Open Source Business Strategies in the Domain of Embedded Systems

Konstantinos Theocharidis

Consultor: Enric Serradell-Lopez
Tutor: Gregorio Robles Martíz

Trabajo Final de Máster de Investigación 2013
Máster en Software Libre

Abstract

It is increasingly understood across the domain of embedded systems that engagement with open source software (OSS) development can provide various benefits. Open source can boost the formulation of new partnerships, provide new communication and recruitment channels, introduce efficient software development methodologies and formulate valuable resources and capabilities for companies. The growing global mobile phone market with over one billion units sold annually represents a particularly prominent domain for OSS, constituting opportunities for new entrants and new technologies. Linux, an open-source-based operating system, has been enabling well-provisioned smartphones since 2002, and is currently being deployed by such companies as Nokia, Google and Samsung. Linux offers a lower software bill of materials and faster time to market than many proprietary alternatives.

In order to succeed with their open source business strategies, companies need to assess, recognise and comprehend the various challenges and risks of OSS development. It is also fundamentally important to have the means to generate appropriate dynamic capabilities to benefit from the open source phenomenon.

Research into OSS has concentrated on three research streams so far: the competitive dynamics, the motivation of voluntary contributors and the governance of open source projects. The research into the competitive dynamics of open source has focused on the categorisation of open source business models. Research concerning industry-specific domains, such as embedded systems, including the identification of relevant challenges, resources and dynamic capabilities, is still craving for more in depth research.

This study aims to increase the theoretical and empirical understanding of open source business strategies in the domain of embedded systems by investigating open source business models, challenges, resources, and operational and dynamic capabilities. Several challenges, resources, and operational and dynamic capabilities of importance to business strategies in the domain of embedded systems could be identified. The results indicate that although the idea of exploiting free OSS is extremely attractive, open source on its own is not a business case.

1. Introduction

In the first part, the background of this thesis is described and introduces the research problem and related research questions, as well as the empirical domain which is the sector of embedded systems. The research field of open source business dynamics, to which this study belongs, is also presented, and the research strategy is explained.

The area of embedded systems is converting into an increasingly attractive domain where everything shifts to more innovative means day by day. For example the sector of the mobile phone market in a global scale, is increasing dramatically overcoming one billion units sold annually, thus forming a prominent landscape for all parts included, from newcomers to known companies within the sphere of embedded systems.

In this environment, free and open source software technologies are granted with an important role. From mobile platforms to desktop and server environments, Linux being an open source platform, has managed to spread out to all of these means, currently bearing its system in well known systems and companies of global amplitude. A strong example is the uprising success of Android based devices, proving the world that Linux based operating systems are well worthy and strong competitors.

Of course for these companies to successfully adopt various successful open source strategies, the challenge and uncertainties amplify, demanding more professional handling. With that said, most companies since they seek the best approach for doing good business, they find themselves confronted with the choice of following a purely open source business strategy, or a combination of both open source and proprietary, which in reality gives them the option to use the benefits of different licensing schemes. Additionally, open source projects are usually based on contributions of a great amount of volunteers and online content, who are usually eager to share their knowledge, regardless of their geographic location. Hence, the need for new organisational practices is born, such as governance of open source software projects, and better application of open mentality business practices that align with the philosophy of open source systems. (Rossi, 2006; von Krogh & Von Hippel, 2006).

A great important milestone, was the theory of the Cathedral and the Bazaar, that set a set of business rules if you prefer, that define the way that business politics affect the market, offer and demand. Even since, the Open Source Software phenomenon was a subject of extensive study and analysis from various esteemed researchers. In my current study I will try to remain around three main subjects, as they are defined mainly by von Krogh & von Hippel, 2006.

Those are:

1) Organisational science (governance, organisation and innovation process)

2) Analysis of the motivations of open source developers
(Lerner & Tirole; Rossi, 2006)

3) Open source business models and strategies (competitive dynamics)
(Katsamakas, 2006).

Apart from the above three categories, there will be presented many references from researches based on open source and free software licensing, that made an impact on shaping the ideology of this area. (Rosen, 2005;St. Laurent, 2004).

Without a doubt, a lot of academics have been interested on this field, given the fact that is constantly evolving, presenting their achievements and thoughts on the subject. Like for example Mullerburg and Osterwalder & Pigneur researches.

Therefore what this current research will attempt to do is on the one hand use the already established academic research information, and on the other hand try to give a different aspect of things with a hint of the future of these practices.

This would be important because the majority of the studies, were conducted under the perspective of grand multinational enterprises, like IBM, Red Hat and Sun Microsystems. Additionally to that, most research gives focus on bigger and broader open source development projects such as Linux, MySQL and Apache.

It is hence noted though, that not much attention has been given to industry specific domains, such as embedded systems. Related open system strategies, resources, challenges and dynamic capabilities are also concepts that lacked academic interest and assessment.

Based on the pre-mentioned views, this thesis study will try to focus on the assessment of open source strategies within the targeted domain of embedded systems, investigating the key factors of such processes. By all means, what a company intents to do is successfully implement open source strategies in their fullest. Hopefully what this study will assess will be the feasibility of these strategies before the actual implementation phase. (Ven, K., & Mannaert, H. (2008) and Berger, A. (2002)).

What defines a strategy is a particular company's business plan to naturally make a higher return on investment. Therefore this study will present and relate, among others, the following:

- The ability of companies to assess the suitability of open source business strategies in the domain of embedded systems, which will be answered based on literature review.

- Search and present the existing open source business models, by combining literature review and evidence.

- Define the major open source challenges, as they arise in related theories and findings.

- Outline the majority of resources, operational and dynamic capabilities needed when using OSS-based business strategies.

This study will not try to bring out the whole spectrum of domains where open source strategies are used, but will concentrate mainly on the domain of embedded systems, thus blocking out the rest of domains in which OSS is used. How these business strategies are implemented throughout the areas of usage, will not be extensively analysed as they deviate from the main axis of study. (Fitzgerald and Magretta)

The term open source encapsulates both the intellectual property strategy and the development methodology of Open Source Software. Furthermore, in this context, open source also refers to the involvement of OSS in the world of business strategies.

In order for the previous notions to be realistically projected, we will also be occupied with the examination of some case studies. Since Mobile industry is one of the most rapidly changing industries, we notice that is frequently forced to adopt the open business models in various forms, in order to accommodate business needs, sales targets and better technology that reaches out to more population groups. Android's platform for example, caused a big stir in the industry when it was first announced. The gradual shift from closeness to openness is inevitable in this industry. We can see in the area of mobile industry which support open source embedded systems, other competitors arising except Android, such as Ubuntu Mobile, Firefox for mobile devices etc. This upward spiral is positively pushing technological industries and vendors, to explore alternative means (such as OSS) and utilize their promising potential. (Ubuntu)

Though open source software is turning into a norm nowadays for a plethora of companies, open source hardware design or embedded system is still rare, but does exist. Open source in mobile industry is still a relatively recent phenomenon. To understand how the mobile handset manufactures can ride on the wave of open source movement in mobile industry, how they can capture value from the "openness", we will examine a few business cases outside mobile industry that used the open source or open innovation. Then we will apply the lessons learned to the mobile industry.

The case studies that we will examine, include well known names in the open source industry such as Red Hat, Novel, Sun's open SPARE, DD-WRT-Buffalo technology, XDA-Developers, NSLU2-Linux, CHDK and Rockbox.
All the above fall in the category of open innovation in hardware and embedded system development. After each examination there will be presented some results that can be taken into consideration, directly deriving from the case studies, that they talk about hardware and software in embedded systems, and are ultimately related with business practices.

Recommendations will be shown that are given to the Mobile industry, specifically in the cell phone handset industry, emphasizing the way that mobile device manufactures can be helped to survive in this competitive business environment. Strictly speaking, "cell phone/mobile phone" and "mobile device" were not exactly the same term in the past. "Cell phone" is one type of "mobile devices". While "cell phone" emphasizes the voice function of a device, "mobile device" emphasizes the computing power of the portable device. However there's a clear trend right now that the two terms are converging, finding ourselves in front of highly developed devices that have nothing lacking from personal PCs.

In fact, Microsoft no long makes different Windows Mobile OS for smartphones and PDAs. Starting from Windows Mobile 5 (WM5), there's only one OS for mobile devices, whether they are smartphones or computers. Mobile device manufacturers can customize Windows Mobile operating system to suit a specific device by enabling/ disabling certain features of that OS, or add extra features. We see nowadays with the launching of Windows 8 for mobiles, that the desktop version resembles and interacts with the mobile version, in terms of functionality and characteristics, giving the user a more complete, "one-device" feeling in their every day interaction with these devices.

This concept was adopted by other organizations as well, such as Ubuntu, who are promoting a "one-device" operating system (in this case Linux), that will be equally potent on desktops, tablets and mobile devices. (Ubuntu)

Through the case studies we will also focus on the technology strategy and business model, instead of the technical differences of different platforms in mobile industry. It is more interesting to see the outcome of the results that derive after each case study and their business practices inside or outside mobile industry using open business models, whether those practices intentionally or unintentionally are following the open business paradigm.

The open source or semi open source projects in the embedded system are more common than open source hardware design, but less common than open source software. The reason is obvious. Embedded device has two pieces, the firmware which is the software that enables all the functions of the device, and the hardware piece which support the software. Most open source projects are done on the firmware part with some modifications on the hardware side. Most projects in this domain generally fall into the category of a more user centred innovation paradigm and are done by hacking communities. The lead users in the hacking communities are able to design much better products than the original manufactures using exactly the same piece of hardware. Unlike open source software communities, the motivation of those "hackers" is usually their personal needs. They are not satisfied with whatever the manufactures can provide so they start to invent themselves. Surprisingly, in most cases, manufactures haven't been able to capture the value from users' innovation. Of course, the non-hacking part of the communities still remain one of the most important workhorse of open source development and support. (West & O' Mahony).

Based on the literature review, the existing open source business models will be discussed and re-categorised. It is acknowledged that this categorisation is not comprehensive, as there are indications of novel business models, and the ICT scenery is constantly evolving, especially in the domain of embedded systems. It is therefore suggested that companies should encourage innovation relating to open source. Companies should also be aware of other possible open source business models.

In essence, companies that are willing to benefit from OSS need to evaluate the advantages and disadvantages between two principal models of open source development, of contribution and exploitation. Thus, it would be useful for them to decide whether to release the well established proprietary software to the public and try to build a supporting community around it, or exchange the existing propriety software for community developed software. In either case, however, the open source alternative should lead to various challenges and alter the present competitive environment of companies.

The major challenges of OSS development in the target domain, relate to the companies' key activities, partner network and client relationships. One of the most challenging areas seems to be the management of OSS communities and deciding between open and proprietary alternatives. As a suggestion to follow, companies should identify the pros and cons of various alternative possibilities carefully and transfer the corresponding decision-making to their business-level strategies.

It is also suggested that companies should acknowledge that OSS development matters should be dealt with throughout the organisation and involve departments such as R&D, sales, marketing, human resources, administration and legal.

Challenges, such as the need for additional customisation, should be seen as feasible business possibilities rather than a thread, as, in many cases, customisation provides additional revenue flows and new business opportunities. At the same time, however, companies need to learn to rationalise the extra work for their customers.

It is increasingly evident that OSS is an enabler in the domain of embedded systems. The OSS phenomenon lowers the entry barriers significantly, and there are practically thousands of companies that are financially capable of engaging in community-based development. This in turn means that existing companies must find new and innovative ways to compete in this emerging situation. It is suggested that these companies should look towards indirect revenue flows and consider a transformation towards service or hybrid business models.

As it will be defined in the theory section of this study, operational capabilities are transferable from one company to another. Dynamic capabilities, on the other hand, are non-exchangeable, company-specific routines or processes by which companies arrange different resources in order to match and create market change.

The operational capabilities needed for OSS development are similar to those in proprietary software development. In the case of open source development, however, some new operational capabilities are required that relate to client relationships (sales), R&D, and legal and liability issues. (Riehle)

In order to find the necessary operational capabilities needed for OSS development, it is suggested that companies should evaluate their business model components thoroughly
and concentrate on those areas that need improvement. Consequently, in the case of dynamic capabilities (routines and processes), there is a need for evaluation and generation of new processes. Essentially, companies need to generate and improve such dynamic capabilities as community evaluation processes and OSS-related organisational routines, including strategical processes.

2 Embedded Systems and Open Source Software

This chapter aims to present the characteristics of embedded systems, by giving an introduction and the components of what we call embedded systems. In continuation a typical embedded systems' design is described, followed by a discussion on embedded systems relating to Open Source Software. At the end, the idea of Mobile Linux is introduced, so it can relate to the following case studies.

2.1. Introduction to Embedded Systems

Embedded systems are composed of computer hardware and software, and in many cases, additional parts, either mechanical or electronic, designed to perform a dedicated function (Barr & Massa, 2006). Embedded systems are found in household devices such as washing machines and video recorders, as well as in other consumer products like car electronics and telecommunication systems. Embedded systems are also found in transportation systems, including car navigation systems, airplane and railway control, and autonomous robots, as well as in industrial process control such as chemical and power generation plants (Müllerburg, 1999). In 2005, less than 2% of the manufactured microprocessors ended up in PCs, Macs and UNIX workstations. The rest, about 8.8 billion, became part of embedded systems. Therefore, practically every electronic device designed and manufactured today is an embedded system.

Embedded systems are commonly characterised, as being single-functioned, tightly constrained, reactive and real-time. Single-functioned refers to a situation in which the embedded system is designed to execute only a single program repeatedly (e.g., a digital camera). Embedded systems are typically tightly constrained by such design metrics as cost, size, performance and power. In contrast to traditional software development, embedded system development is based on mass production. While pure software production is characterised by almost zero-level variable costs, embedded systems are produced in many physical instances; thus the unit cost of the individual instance becomes significant. As a result, the cheapest possible parts are likely to be used.

Embedded systems are also usually resource-limited, e.g., they do not employ hard disks, have power limitations and run built-in, application-specific programs instead of user-defined programs (Müllerburg, 1999). Furthermore, embedded systems are often required to provide real-time responses, i.e., sensing and responding to external events simultaneously within milliseconds or microseconds of their occurrence. A real-time response is needed in systems in which the results of the computation are used to interact with a process as it occurs (Embedded System Definition, 2006).

Devices such as mobile phones and set-top boxes, already include embedded operating systems that can be small in size and designed specifically for embedded system use, or to be a stripped-down version of a system that is commonly used in general purpose computers.

OSS-based Linux is an example of such an operating system. Many embedded systems do not need an operating system, however, as the necessary logic can be implemented as a single assembly language program stored in the device's memory (Embedded

System Definition, 2006).

2.1.1. Components of Embedded Systems

Usually, one can say that an embedded device is only designed to perform a particular task, which leads to a situation in which the expected functionality has clear and well-bounded requirements. As the functionality of the device is so intently defined, the system design can be highly optimised. Thus, it is important to find the optimal ingredients from the plethora of different peripherals (Berger, 2002).

User interface

The user interface in embedded systems vary from no user interface, to advanced graphical user interfaces that bear a resemblance to computer desktop operating systems. While simple embedded devices use buttons, LEDs, or graphic and character LCDs with straightforward menu systems, more sophisticated embedded devices employ advanced graphical screens with touch sensors (such as Android phones, Touch mobile devices, iPhones etc).
Some embedded systems also employ web-based user interfaces that may be used remotely through a network..

Embedded Processors

Embedded processors can be divided broadly into two categories:

- *microprocessors* (μP)
- *microcontrollers* (μC).

The difference between the two processor categories is sometimes blurry. In general, micro-controllers have more peripherals integrated on a single chip and are designed to require a minimum complement of external parts, while microprocessors are individual central processing units (CPU) without any additional peripherals. Micro-controllers have advantages such as lower BOMs, greater reliability, better performance and higher speed (Berger, 2002; Embedded System, 2006).

Compared with personal computers, there is a great number of different CPU architectures for use in the field of embedded systems. Commonly used architectures include Von Neumann, Harvards, RISC, non-RISC, and VLIW. Within the various architectures, the word lengths vary from 4 to 64 bits and beyond. Hence, several architectures come in a large number of different variants and forms manufactured by different companies including AMD, Intel and Texas Instruments (Embedded System, 2006).

Computer Boards

Standard computer boards are mostly x86-based and designed for small, low-volume embedded systems. These boards often use operating systems like DOS, Linux, NetBSD or an embedded real-time operating system, including MicroC/OSII, ONX or

VxWorks.

The components used in embedded systems may be compatible with those used in general purpose x86 desktop computers, especially when small size and power efficiency are not critical. Some boards offer PC compatibility in a highly integrated manner yet have attributes (e.g., small size) that are attractive to embedded engineers, which means that the manufacturer may acquire low-cost commodity components for use with the same software development tools as those used for general software development.

Other Ingredients

In addition to user interfaces, processors and computer boards, embedded systems include ingredients such as memory, software and algorithms. Of course there are many other peripherals to be integrated into embedded systems.

2.1.2. Embedded Systems Design Processes

The conventional design process for embedded systems follows the traditional waterfall development model. This is divided into two branches: one deals with the hardware element and the other with the software element of the system. This is followed by
integration once both streams have been finished. Berger (2002) emphasises how the early introduction of hardware/software integration shortens the design cycle
time.

Much of the software development time is spent on integration after the hardware has been finished. Thus, significant time-saving may be gained by employing virtual hardware/software integration (Berger, 2002).

2.1.3. OSS in Embedded Systems

Several studies suggest that Linux is the most commonly used operating system by far in various embedded devices. So we meet the term *embedded Linux,* which denotes variants of the Linux operating system that are adapted for embedded systems.

It is interesting to observe that an annual survey conducted by Venture Development Corporation (VDC) in 2008 revealed even then, that Linux was used as the target operating system by 18 % of the embedded engineers who responded.

Consequently, Linux was the most widely used of the free and commercial operating systems. Other open-source-based operating systems such as eCos, BSD, FreeRTOS and TinyOS were used collectively by another 5 per cent of the respondents (LinuxDevices.com, 2008).

Regardless of its historical development as an operating system for the desktop and server environment, GNU/Linux has become an attractive choice in the field of embedded operating systems. Linux is a multitasking, multi-user, and

multiprocessor operating system that support various hardware platforms. Should hardware support is provided, its kernel is able to increase system reliability, as one failing application is unlikely to cause the kernel, or any other applications, to fail. Furthermore, Linux has basic features such as no licensing fee, open source specifications and standards, reliability, scalability, a large programmer base, support, standards and portability.
Such features are attractive in the field of embedded systems, as they support rapid application development with a reduced TTM .

2.1.4. Mobile Linux

The growing global mobile phone market with over one billion units sold annually is a prominent domain for embedded Linux. This market offers opportunities for new entrants and new technologies. Linux has enabled well-provisioned smartphones since 2002 and is currently being deployed by companies like Google, HTC, Samsung, Motorola etc.
Linux offers benefits such as lower software BOMs than proprietary solutions, options for user interface customisation, extensions and differentiation, and native support for IP networking and desktop peripheral interfaces. Consequently, Linux-based mobile deployment has been unstoppably growing. Examples of Linux-enabled open source projects within the mobile phone industry include Android (Google), Maemo (Nokia), Ubuntu (Canonical), bada (Samsung Electronics) etc.

Android is a Linux-derived platform supported by Google, along with hardware and software developers such as Intel, HTC and ARM, which form the Open Handset Alliance. In 2007, Android has already given developers access to every aspect of the phone's operation, which led to a boom of new customized Roms for phones.

Maemo is a software platform for smartphones and Internet Tablets. It is based on the Debian operating system and mostly on open code. Maemo is developed by Maemo. Devices within Nokia in collaboration with many open source projects such as the Linux kernel, Debian and GNOME. Lately it has gradually being given up, as Nokia is mostly occupied with Windows Mobile platforms.

Canonical has proceeded in creating an Ubuntu version for mobile phones, which promises a new experience to the users, from usability, effectiveness and innovation. It has presented a sample of what is yet to follow, released its code, enabling developers to work on its fast evolution and adaptation in many handsets. It will be interesting to see how this move will end up.

Samsung's bada mobile phone operating system has been mainly developed during 2010, presenting the first handsets bearing this OS. Samsung is now collaborating extensively with Android, giving mobile devices with a competitive edge in the market, with great development and innovation works.

Intel and Nokia had joined forces in the past by merging their open-source-based platforms Moblin and Maemo as MeeGo. Linux Foundation hosted MeeGo and its first release around 2010, with devices launching in the same year.

2.2. Open Source Software

During this part, we will analyse the characteristics of Open Source Software. It begins laying out the relevant history, and will continue to matters of OSS development. Then, the OSS licensing schemes are categorised and explained. Open Source Software is discussed in the context of embedded systems, emphasising the topics of embedded Linux and Mobile Linux. Lastly, we get a glance on the future prospects.

2.2.1. The OSS Phenomenon

Open Source development is inextricably linked to software development whether it involves proprietary or free software. During the decades of 60s and 70s, the burden of software development was mainly on the shoulders of scientists and engineers in academic environments, such as the MIT. At the MIT'S AIL (artificial intelligence laboratory), the communal hacker culture was strongly present among a group of programmers and *software hackers*. For these individuals, open sharing of software code was common practice simply because commercial software solutions and support were not available at the time.

Moving on in the 80s, it is noted the first licensed code by the MIT intended for a commercial company, leaving a group of hackers facing a major setback, especially because that code was created by themselves. In accordance with normal commercial practice, this new licensee then withheld access to the source code of the licensed software and prevented MIT hackers from continuing to use it as a platform for further learning and development (von Hippel & von Krog).

Richard Stallman, a programmer at MIT's Artificial Intelligence Laboratory, was extremely discontented and offended by the loss of access to communally developed source code. Stallman saw these restrictive practices as morally wrong and, in response, founded the Free Software Foundation (FSF) in 1985. Stallman also introduced a political aspect to the matter, claiming that "information wants to be free" and software should not only be owned by companies, but rather be a public good. To implement his ideas, Stallman designed a legal mechanism, the General Public License (GPL) - also referred to as *copyleft.*
The Linux kernel is a prominent example of software licensed under GPL.

Despite Stallman's efforts to proclaim the message that free software is about freedom not price, the success of FSF was somewhat self-limiting. In part, this was due to flexibility issues with GPL, including restrictions on using proprietary code along with free code. The significant part of the problem, as suggested by Bruce Perens and Eric Raymond, lay in Stallman's concept of *free software*, which some companies found suspicious from the business point of view.

As a result, Raymond and Perens in collaboration with some other well-known hackers, founded the Open Source Software Movement, which eventually led to the Open Source Initiative (OSI) and finally to the Open Source Definition (OSD) in 1999.

The primary difference between *open source* and *free software* is based on philosophical grounds. It prefers to emphasise on the practical benefits to issues regarding moral rightness.

Although the opening quote of this section gives a rough idea about OSD, it still contains a plausible promise of other characteristics. OSD presents the criteria for whether the software licence can be considered open source. According to OSD (1999), the distribution terms of OSS must comply with the following criteria:

1) *Free Distribution* - The licence shall not restrict any party from selling or giving away the source code.

2) *Source Code* - The program must include source code, as the evaluation and modification of the software must be made as easy as possible.

3) *Derived works* - The licence must allow modifications and derived works.

4) *Integrity of The Author's Source Code* - Changes to source code must be clearly distinguished from the work of the original author.

5) *No Discrimination Against Persons or Groups* - The author may not restrict any particular people or organisations from using or modifying the software.

6) *No Discrimination Against Field of Endeavour* - The licence must not restrict anyone from making use of the program in any specific field of endeavour. For example it may not restrict the program from being used equally for business or for genetic research.

7) *Distribution of Licence* - The rights attached to the program must apply to all to whom the program is redistributed.

8) *Licence Must Not Be Specific to a Product* - The licence may not limit the use of the software to any particular usage environment, such as Debian, Ubuntu or Windows.

9) *Licence Must Not Restrict Other Software* - The licence must allow the software to be distributed with non-free software.

10) *Licence Must Be Technology-Neutral* - The licence must allow the software to be distributed by different methods, and the download of the source code must not be limited to a particular interface, such as a graphical user interface. It must therefore also be possible to acquire the software over non-Web channels.

To summarise, a user who possesses a copy of OSS has a legal right to use it, study its source code, modify it and distribute modified or unmodified versions to others (Raymond, 1999).

In this thesis, the term *open source software* (OSS) is used to refer to the following concepts and terms: *free software*, *free and open source software* (FOSS) and *free/libre/open source software* (FLOSS). Despite sharing similar development models, these terms represent slightly different philosophical approaches.

2.2.2. OSS Development

Raymond then, based on his examinations of the Linux kernel development project, introduced for the first time the terms *cathedral* and *bazaar* to contrast two different software development models.

The *cathedral development model* is based on small teams, easily understandable, comprehensive project plans and few release cycles, and feedback is only rarely acquired. In this model, the source code is only available once the software is released and is restricted between releases. *GNU Emacs* and *GNU Compiler Collection* projects are examples of this approach.

The *bazaar development model*, on the other hand, is the kind of software development in which the source code is constantly visible to the public. Anyone, anywhere, with a practical idea can contribute to the project. Feedback is sought as early and frequently as possible, and the maintainer confirms the software release when he or she determines that it is 'good enough'. Raymond credits Linus Torvalds as the inventor of this approach. The *Fetchmail* project is an example of the bazaar development model.

According to Fitzgerald, in traditional software development, the development life cycle normally consists of four general phases:

Planning/ Analysis/ Design/ Implementation.

In open source development, the first three stages are chained and typically executed by an individual or small core group.

The implementation phase consists of several sub-phases (according to Fitzgerald):

1) Code - Writing code and submitting it to the open source community for review.

2) Review - The Strength of open source is the independent, prompt peer review.

3) Pre-commit test - The negative implications of breaking the build ensure that contributions are tested carefully before being committed.

4) Development release - Code contributions may be included in the development release within a short time of being submitted - this rapid implementation is a significant motivator for developers.

5) Parallel debugging - In terms of Linus's Law, "given enough eyeballs every bug is shallow", it means a situation in which a large number of debuggers, using different platforms and system configurations, ensure that bugs are found and fixed quickly.

6) Production release - A relatively stable debugged production version of the system is released.

OSS development projects can generally be placed into two subcategories:

- *community- founded projects* (launched by individuals)
- *sponsored projects (*launched by commercial companies, government agencies or non-profit-organisations (West & O' Mahony).

Community-founded projects, the more common of the two, are normally launched by one or more individuals who recruit developers to contribute to new-born software, still in its infant stage. Linus Torvalds's ignition of the Linux kernel is an example of such a project. Community-founded projects typically remain managed by the community and are usually protected from takeovers by companies or other organisations (West & O' Mahony). In the case of the Linux kernel, Torvalds reserved clear leadership of the project. Instead of writing a large amount of code, he quickly moved to coordinating the development project, making decisions on contributions and settling disagreements (Lerner & Tirole). In community founded projects, the contributors do not normally share the same employer. Additionally, community-founded projects may have sponsor contributors, but they are in the same line as other contributors, meaning that they have to earn and maintain their roles in a project, in a similar way to other contributors (West & O' Mahony).

Sponsored projects are launched by commercial companies, government agencies or non profit organisations through the release of their internally developed software code to the public under an OSS compliant licence. Community building is performed by inviting external developers to join the founded open source project. Examples of this include IBM (Eclipse), Sun (OpenOffice) and Netscape (Mozilla) (West & O' Mahony). Sponsored projects do not progress without contributions from the sponsors. On the contrary some arguments claim that the launching of sponsored open source projects requires significant investment by the launcher.

2.2.3. Governance of Open Source Projects

The OSS development based on the bazaar model, can be described according to Raymond as a chaotic, almost anarchic, exchange of ideas with differing agendas and approaches. According to Rossi now, the bazaar metaphor refers to a system of distributed innovation in which great numbers of developers are involved. Bazaar-type development is characterised by the absence of a centralised decision-making unit to define the direction of software development, like for example, concurrent design and debugging, integration of users into the production and self-selection of programmers to the tasks that best benefit from their abilities. The anarchistic image of OSS development above does not accurately explain the phenomenon however. Rossi continues by stating that leadership and authority are essential to effective OSS development. So in a way it depends on the perspective of looking at things.

Open source governance can be grouped into three major types of leadership for OSS development projects (Rossi):

- *benevolent dictatorship*
- *rotating dictatorship*
- *voting committee*

The most prominent example of benevolent dictatorship can be associated with Linus

Torvalds and the Linux kernel project. Torvalds was the founder of the Linux kernel development project and therefore the leader of the community. He gained reputation and authority within the community by showing his initial commitment to the project, as well as his continued interest and involvement even today.

An important advantage was also the fact that Torvalds is a talented programmer and can make contributions of a high quality to the project. Torvalds is also capable of delegating tasks and leadership to others thus maintaining better control over the course of the project.

The rotating dictatorship style of leadership is adopted less in practice, but it has been used in the Pearl development project. The Debian development project is a famous example of the project leader being elected periodically by all the official Debian contributors. Elections are held once a year, so the leadership changes regularly. A downside to this approach is the lack of consistent project management, which may harm the credibility of the community.

The last form of leadership in OSS projects is the voting committee. The Apache Web server project is a good example of this practice in which the decision process is based on e-mail voting. In the beginning, there was a voting committee with eight members. The committee grew steadily over the years, as regular contributors committing patches and bug fixes were invited to the voting committee.

2.2.4. What motivates developers

Over the past few years, analysing what motivates OSS contributors has been an interesting field for attention when we refer to the OSS phenomenon. The motivations of OSS contributors have been studied from different viewpoints of various research disciplines, including cultural anthropology and organisational science (economics).

The main motivations can be broadly put into 2 groups:

endogenous and *extrinsic* motivations

Extrinsic motivations include motivations such as *reputation*, *user needs*, *learning* and *performance*. In other words, extrinsic motivations can be related to immediate or delayed benefits, generally through monetary compensation.

Lerner and Tirole state that: "a programmer participates in a project, whether commercial or open source, only if they derives a net benefit (broadly defined) from engaging in the activity. The net benefit is equal to the immediate payoff (current benefit minus current cost) plus the delayed payoff (delayed benefit minus delayed cost)" and continue by suggesting that the OSS phenomenon can be explained (almost entirely) by economic theories.

Rossi then reports that the reputation aspect, in the extrinsic motivation category, has been acknowledged as a reason for individual developers to join OSS projects, as the characteristics of OSS projects allow developers to gain benefits in terms of reputation. Empirical analysis however, suggest that reputation by itself is by no means the most prominent motivation for developers to contribute to OSS projects. User needs, fixing a bug, solving a problem, make something function correctly, constitute a powerful incentive to create software code.

Moving forward to endogenous (intrinsic) motivations, we can define it as "the doing of an activity for its inherent satisfactions rather than for some separable consequence. When intrinsically motivated, a person is moved to act for the fun or challenge entailed rather than external prods, pressures or rewards." Rossi explores with an academic point of view, classifying intrinsic motivations as hedonic motivations, altruistic, and under a general reciprocity.

Of course there's the other point of view that supports that the members of the community do not require any type of motivation for participation. They can be motivated by a combination of endogenous and extrinsic factors with a personal sense of creativity being an important source of effort.

Other researchers were occupied with how different motivations of contributors are related, and if they are independent, complementary or contradictory. It's been noted that developers' motivations are not independent but related in complex ways. Mixed motives, for example, developers' paid participation (as employees) and status motivations lead to rather high contribution levels, while use-value motivations lead to mediocre contribution levels. Consequently, endogenous motivations do not have a significant impact on average contribution levels.

Cultural anthropology suggests that complementary social norms, such as familiarity and kinship, amongst contributors in projects are important for motivation. Bazaar-type communal software development may be compared with the situation in a family in which calculated economic relationships do not exist.

2.2.5. What motivates commercial companies

Companies are in a non-stop race of finding new ways of generating revenues, reducing costs, maximising profit. In order to achieve these objectives, several multinational companies, (IBM, HP, Sun etc), are increasingly including OSS in their business strategies.

There are several motivations for commercial companies to engage in open source activities. A common economic motivation for software vendors is to save development costs by open sourcing and letting others participate in the software development. Participating companies may be able to generate revenues through provision and by selling complementary products or services. In addition, by establishing a successful open source platform, companies may operate more efficiently across technology stacks and thus broaden their markets. Furthermore, as open source is a common good, active participation in open source activities generates goodwill for the parties involved.

2.2.6. Licensing

Although the guideline by which software licences qualify as OSS licences is defined in the OSD, companies applying open source face a plethora of different licensing options, and are consequently challenged to make important strategic decisions.

A software product has a very specific nature and production attributes. It can therefore be considered an intellectual expression of ideas that are coded using a specific programming language. As a result, its intellectual property rights (IPR) fall in the field of copyright (instead of the field of patents).

An owner of an intellectual property can grant licences to allow others to exercise property rights that would otherwise be exclusive to the property owner (Rosen).
We will be focusing on the most common OSS licences, since there are hundreds of them.

OS Licensing Taxonomy

The term *licence* is used to describe the legal way a copyright and patent owner gives permission to others to use his or her intellectual property (Rosen).

An *open source licence* is a licence that has been approved by the OSI based on the OSD. It is the way the copyright or patent owner grants permission to others to use their intellectual property in such a way that software freedom is protected for all (Rosen).

A *proprietary licence* is the way the copyright or patent owner grants permission to others to use their intellectual property in a restricted way so that software freedom is not protected. The term proprietary is not a synonym of commercial, as commercial licences can be either proprietary or open source. A *commercial licence* is just a term used to describe a licence used in commerce (Rosen).

Copyleft is a licensing scheme that facilitates open and decentralised software development. To be able to modify and distribute software with a copyleft licence, the software developer agrees not to impose licensing restrictions on others by signing the GNU GPL. According to the GPL, everyone has a right to use, modify and distribute the software. Furthermore, all enhancements to the code have to be licensed on the same terms. Code that mixes with the cooperatively developed code also has to be licensed accordingly (Lerner & Tirole). Also, copyleft licences can be divided into *strong* and *weak copyleft licences*.

Lerner divides OSS licences into three classes according to their restrictiveness:

Highly restrictive licences, such as the GPL,
Restrictive licences, such as the Lesser General Public Licence (LGPL),
Non-restrictive licences, such as the Berkeley Software Definition (BSD).

Highly restrictive licences can be associated with strong copyleft licences and non restrictive licences with weak copyleft licences, while restrictive licences remain somewhere in between these contrary licensing categories.

Highly Restrictive Open Source Licences

The classic example of highly restrictive licences is the GNU GPL. The word GNU is a play on the words "GNU's Not Unix" by the licence author, Richard Stallman. GPL originates from the early days of the FSF and was introduced in 1989.

Highly restrictive licences, such as the GPL, state that once a developer licenses the software, consequent derivative programs based on the original must also be licensed in the same way (Rosen). The most restrictive idea of the GPL lies in its second chapter, which states that the licensee must: "cause the whole of any work that you distribute or publish, that in whole or in part contains the Program or any part thereof, either with or without modifications, to be licensed at no charge to all third parties under the terms of this General Public License." (GNU General Public License, version 2, Section 2, 1989) .

The Linux kernel project is the most famous example licensed under the GPL. Initially, in 1991, Linus Torvalds distributed Linux under a licensing agreement that restricted any payment for the program and required all programs distributed or used with Linux to be freely available. He soon relaxed these restrictions, however, and Linux was licensed under the GPL in 1992 (Lerner & Tirole).

Restrictive Open Source Licences

The LGPL is the best-known licence in this category. Like the GPL, once the developer licenses the program, the LGPL also requires the derivative programs based on the original to be licensed. The LGPL allows greater flexibility in regard to the 'mixing' requirement however: LGPL-licensed libraries can be linked to non-GPL licensed programs, including proprietary software. Moreover, under certain conditions, the libraries can be licensed under different licensing scheme (Lerner; St. Laurent).

Other OSS licences that can be categorised as restrictive are the Mozilla Public Licenses.

Non-restrictive Open Source Licences

The BSD and MIT licences were the first OSS licences and they demonstrate some of the very basic principles of OSS licensing. These licences are extremely non restrictive (permissive). They allow the original code to be used in proprietary software and do not require open source versions of the code to be distributed. Furthermore, code created under BSD or MIT licences, or derived from such code, may go 'closed', and development can be made under that proprietary licence, which means that the derived source code will not necessarily be available to the open source community any more. These licences are also compatible with almost all other OSS licences, including the GPL (St. Laurent).

Apple, for example, has used the BSD in the development of its Mac OS X kernel. The kernel, called Darwin, is based on BSD UNIX and re-licensed under Apple's own terms under the Apple Public Source License. While the user interface of the Mac OS X is proprietary, the source code of Darwin is open. Apple may therefore benefit from large community networks around different BSD variants. Apple cannot benefit from the vast Linux development directly, however, Linux is licensed under the GPL and therefore not compatible with Darwin.

Benefits of OS Licensing

St. Laurent suggests that open source licensing enables three significant improvements over traditional proprietary licensing models.

The first benefit is the improved grounds for *innovation*.
Programmers of OSS are willing to contribute to OSS projects for no reward other than making the program more useful. Therefore, the greater the number of contributors working on a project, the more possibilities of generated value.

The second benefit is *reliability*.
Linus's Law tackles the issue illustratively: "given enough eyeballs, every bug is shallow". Accordingly, St. Laurent implies that thousands of eager and technically capable users are more likely to find bugs and make improvements than the employees of the creator of the original proprietary software.

The third benefit is *longevity*.
In many cases, proprietary software comes to its evolutionary end and is no longer supported or updated by the original software vendor. It may even be impossible to purchase the product after a given stage. In the case of open source, however, the software can be updated, rewritten or modified by any technically capable user who needs it, even after a period of silence.

2.2.7. Future Prospects of OSS

Software, (proprietary of open), is extraordinary in nature. It has exciting features, ground for constant evolution, including high fixed development costs as opposed to almost zero-level variable costs and collaborative development approaches. Compared with commercial software, however, OSS has some more distinguishing features. OSS features a plethora of different licensing schemes, it is typically developed using a bazaar-type collaborative development methodology, and it has unique philosophical perspectives and variable developing motives. So before implementing OSS-based business strategies, companies should carefully assess those characteristics in order to be successful.

Although there are challenges in OSS development, the future prospects of the open source phenomenon seem extremely strong. We see nowadays OSS-based operating system platforms, such as Android and other Linux-based operating systems, to largely dominate the smartphone market, following an uprising way towards smart development and innovation.

3. Business Models and Strategies

In order to predict the future opportunities and risks of a business unit, it is important to study the history as well as the present. It is also essential to have the necessary concepts and means for analysis. The *business model* as a concept offers an adequate conceptualisation and thus provides the means for dividing a company's underlying business logic into various sub-components or sub-elements. Furthermore, in order to obtain an even broader picture, emphasis should also be pointed at the complex web of relationships, i.e., the *ecosystem* to which the company is attached . The value-adding activities within the ecosystem, i.e., the *value chain,* are consequently a relevant target of investigation when exploring business units such as an ICT company.

First, the concepts of *ecosystem* and *value chain* are discussed.
Second, the different definitions of *business model* are explored.
Third, some common frameworks for business models are presented briefly.
Fourth, a summary of this chapter.

3.1. Ecosystem and Value Chains

The sector of embedded systems is becoming an increasingly interesting domain in which OSS can provide a significant boost to newcomers as well as established companies. In essence, the growing global mobile phone market with over one billion units sold annually represents a prominent landscape for OSS, constituting opportunities for various players within the ecosystem of embedded systems.

There are suggestions that an industrial organisation is comparable to software architecture. While software architecture is concerned with the decomposition of the system into modules, the functionality of the modules and the interaction between them, the organisation correspondingly decomposes the industry into partners and competitors, identifies the separate and overlapping responsibilities of the individual companies, and acknowledges the relationships between the companies.

3.1.1 Software Ecosystem

The software ecosystem typically consists of the following players:

1) *End-user organisations* represent parties with domain-specific needs for software-intensive products and services.

2) *Industry consultants* analyse and convey vertical and horizontal industry needs and transform these needs into application software features and capabilities.

3) *Application software suppliers* develop software applications emphasising their core competencies, for example, project management skills.

4) *Infrastructure software suppliers* concentrate on the needs of various application

software developers and operators and are therefore able to benefit from economies of scale.

5) *System integrators* specialise in provisioning by integrating software from application and infrastructure suppliers, and infrastructure equipment, i.e., hardware.

6) *Business consultants* concentrate on organisational issues by offering adaptations for particular end-user organisations, for example, domain-specific training.

7) *Application service providers* (ASPs) license and operate the application.

8) *Infrastructure service providers* purchase or license and operate the hardware and software infrastructure, including computers, storage, network and operating system.

3.1.2. Open Source Ecosystem

OSS has altered the traditional rules of the software business significantly and changed the economic interaction among various players within the ecosystem (Riehle).

Riehle suggests that *system integrators* (solution providers) gain most benefits from OSS. System integrators are able to profit through direct cost savings and attract more clients due to increased pricing flexibility. In addition, system integrators may gain significant advantages by using OSS as part of their offering, as it allows them to increase their profits and/or sales volumes through direct cost savings. The system integrator's profits depend on how much he or she spends on each layer of the offering stack.

Riehle points out the differences in costs and pricing between closed source software and OSS. In closed source software business, *Independent software vendors* (ISVs) operate in a situation in which the profit-maximising price is mainly not dependent on the cost. Price competition is not very likely because of the entry barriers to the market. In essence, the high initial investments needed to produce a software product form entry barriers.

In OSS business, competition is fierce, as anyone can easily enter the marketplace. Pricing is based on mark-up over cost. If the mark-up is too high, competition increases, and if it is too low, competition decreases. Companies do not sell the software itself but rather its provision, maintenance and support services.

Different open source ISVs have different cost structures depending on the amount of resources they use in OSS development, i.e., the amount of contributions to open source projects. The increasing number of contributions to open source projects allows companies to charge higher prices, however, as they now possess new capabilities that may be offered as services (Riehle).

According to Riehle, there are two different kinds of *open source service companies.*

First, the group offers first-level support and implementation services, and second, it provides second-level support, training and development services. Open source service companies need to recruit and retain the right people. The employee's technical skills and knowledge about open source products determine his or her value to these companies. These companies also need to be able to set up and execute customer specific service processes and generate expert domain knowledge and unique intellectual property reliably.

3.1.3. Software Value Chains

Various discussions value chains primarily from the perspective of software development. In this study, the idea of software value chains is broadened to also include embedded systems development. There are two value chains in software development: the *supply value chain* and the *requirements value chain*. The general idea is that participants in one chain add value to the other.

The supply value chain relates to the execution phase, starting with the software vendor and ending by providing valuable functionality and capability to the user.

The typical supply chain has four main stages:

1) The *Implementation* stage includes detailed requirements, architecture, programming and testing of the results into a working software product that can be sold.

2) In the *Provisioning* (acquisition) stage, the required facilities (e.g., network, servers and desktop computers) are acquired and deployed. This stage also includes the integration (installation and testing) of software from various suppliers. The organisational and process changes must also be planned and implemented in this phase. The outcome of provisioning is an operational application. System throughput and growth needs must be considered, however, in order to maintain performance attributes in the future.

3) In the *Operation* stage, the application and supporting infrastructure are kept running reliably and securely. This stage also involves user support and reporting of defects back to the suppliers.

4) At the *use* stage, the application functionality provides direct value to end-users and organisations.

The requirements value chain relates to the software implementation phase, starting with business and application ideas, collects functional objectives from users and ends with documented requirements for implementation.

3.2. Definitions of a Business Model

"The definition of a business model is murky at best. Most often, it seems to refer to a loose conception of how a company does business and generates revenue. Yet simply having a business model is an exceedingly low bar to set for building a company. Generating revenue is a far cry from creating economic value " (Porter, 2001)

In their study, Morris, Schindehutte, & Allen (2003) point out that the business model definition, in the applicable literature, can be divided into three categories based on its principal emphasis.

1. *Economic* - At the most basic level, a business model is defined exclusively in terms of the company's economic model with the core logic of profit generation (Morris, 2003). Stewart and Zhao (2000, p. 290) see the business model as *"a statement of how a firm will make money and sustain its profit stream over time."*

2. *Operational* - At this level, the business model embodies an architectural configuration, and the focus is on the internal processes and design of the infrastructure that enable the creation of value. Mayo and Brown (1999, p. 20) point to *"the design of key inter-dependent systems that create and sustain a competitive business."*

3. *Strategic* - At the strategic level, definitions emphasise the overall direction of the company's market positioning, interactions across organisational boundaries and growth opportunities. The main concerns include competitive advantage and sustainability. Consequently, Slywotzky (1996, p. 4) refers to *"the totality of how a company selects its customers, defines and differentiates its offerings, defines the tasks it will perform itself and those it will outsource, configures its resources, goes to market, creates utility for customers and captures profits."*

3.3. Business Model Frameworks

In general, a business model illustrates two important roles: value creation and value capture. First, a business model defines a series of activities that produces a new product or service in such a way that the net value is created through the various activities. Second, a business model captures the value from a portion of these activities for developing the model (Chesbrough, 2006a). Some well-known business model frameworks are presented briefly below.

3.3.1. Functions of a Business Model

According to Chesbrough and Rosenbloom (2002), a business model includes six functions:

1) Articulate the *value proposition* - that is, the value created for users by the offering.

2) Identify a *market segment* - meaning the users to whom the offering and its rationale are useful.

3) Define the *structure of a value chain* required by the company to create and distribute the offering and determine the complementary assets needed to support the company's position in this chain.

4) Specify the *revenue generation mechanisms* for the company, and estimate the cost structure and profit potential of producing the offering, given the value proposition and value-chain structure chosen.

5) Describe the position of the company within the *value network* (ecosystem), linking suppliers and customers, including identification of potential third-party actors and competitors.

6) Formulate the *competitive strategy* by which the innovating company will gain and hold an advantage over rivals.

3.3.2. Strategic Framework for Conceptualising a Value-Based Company

Morris proposed a strategic framework for conceptualising a value-based company. Their framework can be used to design, describe, categorise, critique and analyse a business model for any type of company. At the foundation level of this framework, the business model consists of six components and related key questions.

In addition to the foundation level, the above framework has two further levels. These layers are a *proprietary level* for creating unique combinations for a particular venture and a *rules level* for ensuring that the model's foundation and proprietary elements are reflected in the on-going strategic actions.

3.3.3. Seven Generic Business Model Components

Here it's presented a generic business model that includes the following causally related components, starting at the product market level:

1. Customers
2. Competitors
3. Offering
4. Activities and organisation
5. Resources, and
6. Supply of factor and production inputs
7. Scope of management

3.3.4 .Nine Business Model Building Blocks

Osterwalder employ an ontological approach to conceptualise the business model components. They suggest that the business model is a conceptual tool containing a set of objects, concepts and their relationships with the objective of expressing a company's business logic. The business model includes four core components: *product*, *customer interface*, *infrastructure management* and *financial aspects*.

In his blog, Osterwalder refined some of his earlier studies and presented a version of the business model ontology that had been updated to some extent.
According to his updates, the nine building blocks are:

1) The *value proposition* of what is offered to the market
2) The *segment(s) of clients* that is/are addressed by the value proposition
3) The *communication and distribution channels* to reach clients and offer them the value proposition
4) The *relationships* established with *clients*
5) The *key resources* needed to make the business model possible
6) The *key activities* necessary to implement the business model
7) The key partners and their motivations to participate in the business model (*partner network*)
8) The revenue streams (*revenue flows*) generated by the business model (constituting the revenue model)
9) The *cost structure* resulting from the business model

Value Proposition

The formulation of the value proposition is based on various other business model components, including capabilities, key resources, client relationships, partner network and key activities. In other words, the value proposition is based on the whole value creation system of the company's value network.

According to Osterwalder, the value proposition "describes the way a firm differentiates itself from its competitors and is the reason why customers buy from a certain firm and not from another".

The concept of value proposition is divided into three sub-elements:

1) The cost element (price, effort and risk)
2) The role of the client (buyer, user, co-creator or transferor of value)
3) The performance of the value proposition

Differentiation exists when a company's offering is preferred over rival companies' offerings on some bargaining incident or in the long run. Furthermore, differentiation is about making the offer different in response to differences in demand and this may be achieved through altering any aspect of the offering (not just produce features), including the financial cost of acquisition (i.e., price charged).
Osterwalder and Pigneur suggest three paths to differentiation:

1) Innovation through new, complementary or customised offerings
2) Providing a lower price than the competition
3) A premium customer service level and customer relationship excellence

Client Segments

In general, companies select specific target customers by segmenting. Segmenting enables companies to allocate resources to the clients that are most likely to be interested in their value proposition. Targeting considers at which clients the value proposition is targeted, which geographical areas are included and what product segments are offered. A broad distinction between target customers exists between, for example, business-to-business (B2B) and business-to-consumer (B2C) markets (Osterwalder & Pigneur).

Channels (for communication and distribution)

Channels may be understood in a broad sense - while there are *distribution channels*, there are also several *communication channels*, which are used when dealing with customers and other stakeholders. A distribution channel describes how the company delivers its value proposition to its target customers, thus channels may differ depending on the nature of the offerings (e.g., physical vs. electronic) and client segment (Osterwalder).

Client Relationships

The client relationships component explains the kind of relationships and links a company establishes between itself and its different client segments. All client interactions affect the strength of the relationship a company builds with its clients. The goals (profits) of client relationships may be achieved through the acquisition of new clients, the enhancement of profitability of existing clients and the extension of the duration of existing client relationships. The client lifetime value (LTV) reflects a desire to leverage customer acquisition investments by building long-term customer relationships.

Key Resources

Resources are the cornerstones of any business model, as they allow companies to create and offer their value propositions, reach client segments and markets, maintain client relationships and finally generate revenues. "Key resources can be physical financial, intellectual, or human" and they "can be owned or leased by the company or acquired from key partners" (Osterwalder & Pigneur).

Physical resources include assets such as manufacturing facilities, buildings, systems and distribution networks. Financial resources include, for example, cash, lines of credit and stock option pools for hiring key employees. Intellectual resources are assets such as brand names, proprietary knowledge, patents and copyrights, partnerships and customer databases. Human resources, i.e., people, are important in any business model, especially in knowledge-intensive and creative industries.

Key Activities

Key activities are the most important activities a company has to perform in order to operate successfully. Similarly, like key resources, key activities are required to create and offer a value proposition. In this business model framework, key activities may be divided into three categories: *production*, *problem solving*, and *platform or network*. Production-related key activities include designing, manufacturing and delivering the products. Problem-solving key activities relate to new innovations, knowledge management and continuous training. Platform or network-based key activities are essential to ICT companies, especially to companies with OSS-based business models, as platforms and networks form essential key resources for these companies.

Partner Network

Osterwalder defines *partnership* as "a voluntarily initiated cooperative agreement formed between two or more independent companies in order to carry out a project or specific activity jointly by coordinating the necessary capabilities, resources and activities."

Partnering may be looked at from four different perspectives. From the first perspective, partnerships may be seen as optimisations of resources - companies focus on their core competencies and rely on partner networks and outsourcing for non-core competencies and activities. From the second perspective, partnerships are looked at from an RBV of the company in which the contribution of partnering is gained by acquiring resources that the company lacks. This may include resources such as a customer database, brand name or patents and technology that a firm does not possess. From the third perspective, partnering and cooperation are closely linked to organisational learning, such as market knowledge or management skills. A fourth perspective on partnering emphasises joint ventures in order to conquer new foreign markets and create completely new markets from scratch. Partnering, even between competitors, is not uncommon today.

In their book, Osterwalder and Pigneur further refine the partner network module by renaming it *key partnerships*. They distinguish between the key partnerships of four different types of partnerships:

1) Strategic alliances between non-competitors
2) Coo-petition: strategic partnerships between competitors
3) Joint ventures to develop new businesses
4) Buyer-supplier relationships to assure reliable supplies

Revenue Flows

A company's revenue model (the revenue flows-component) includes all arrangements that permit the participants in business interactions to charge fees that are covered by one or several other participants in order to cover costs and add a margin to create profits.

Furthermore, revenues can be generated in a *direct* or *indirect* way. In the direct model, revenues come directly from the customer. Direct revenue flows are typically generated from *sales*, *transaction fees* and *subscriptions*. In sales, revenues simply come from products or services. Transaction fees are gained when, for example, an e-payment service provider deducts commissions for monetary transactions between the client and the service provider. Subscription revenues may be acquired from one-time charges or periodic charges.

According to Osterwalder and Pigneur, a business model can involve two different types of revenue flows: *transaction revenues* resulting from one-time customer payments and *recurring revenues* resulting from on-going payments to deliver a value proposition to customers or to provide post-purchase customer support.

In the indirect model, the actual product will be provided gratis and the revenues generated through a third party interested in the diffusion of the product. Indirect models involve advertisements, affiliate models or bundling.

Cost Structure

"The cost structure describes all costs incurred to operate a business model" Osterwalder and Pigneur. In other words, the cost structure defines the monetary value of all the resources, assets, activities and partner network relationships within the company. Companies may generate significant cost savings by focusing on their core competencies and activities and relying on partner networks for other non-core competencies and activities.

Business model cost structures may be divided in two broad sub categories:

cost-driven and *value-driven*.

In cost-driven models, the focus is on cost-minimisation, wherever possible, enabled by means such as automation and outsourcing. In value-driven models, the focus is on value creation with a high degree of personalised service and high profile, exclusive products and services (Osterwalder & Pigneur).

Furthermore, cost structures may have the following characteristics: *fixed costs, variable costs, economies of scale* and *economies of scope*. '*Fixed costs*' stands for costs that are independent of the production volume. In other words, *fixed costs* remain the same whether zero or a thousand units are produced. *Variable costs* are a kind of opposite of *fixed costs*, as they depend proportionally on the production volume. *Economies of scale* give advantages, especially to large companies, as they can benefit from lower purchase rates when buying large amounts of production factors. Likewise, *economies of scope* usually give benefits to large companies, as they may use the larger scope of operations by, for example, marketing multiple products concurrently on the same media.

The business model ontology designed and refined by Osterwalder is proposed as the chosen business model framework to be used in the empirical part of this study. The following section provides the reasoning behind the selected business model framework.

3.4. Open Source Business Strategies

"Yet to an economist, the behaviour of individual programmers and commercial companies engaged in open source processes is startling" (Lerner & Tirole).

This chapter first discusses the concept of *business strategy* briefly.
Secondly, it distinguishes between the concepts of *business model* and *business strategy*.
Then, a literary review on OSS-related business models and strategies is provided.
Finally, the chapter is summarised.

3.4.1. Business Strategy

Strategy as a term is old with a variety of different implications, though it generally refers to a *careful plan,* a *method* and to an *art of employing plans and methods* . In the field of economics, there is a plethora of terms relating to the concept of *strategy*: corporate strategy, business strategy, functional strategy, competitive strategy, etc.
A relevant choice for this study, from the legion of strategy-related terms, is *business strategy*. Business strategy can be viewed as "positioning of resources of the firm relative to its environment, all to achieve desired performance outcome".

Strategy is generally divided into three strategy levels: *corporate level*, *business level* and *functional level*.
These strategy levels may be arranged in hierarchical order from general to least general as follows:

1) Corporate level
2) Business level
3) Functional level

Beard and Dess define *corporate level strategy* in terms of "variation in the deployment of a firm's resources among the portfolios of industries within which all business firms compete." Hofer and Schendel offer the following definition for corporate-level strategy: "corporate-level strategy is concerned primarily with answering the question of what set of businesses should we be in. Consequently, scope and resource deployment among businesses are the primary component of
corporate strategy".

According to Hofer and Schendel, at *business level* "strategy focuses on how to compete in a particular industry or product-market segment. Thus, distinctive
competencies and competitive advantage are usually the most important components of strategy at this level." Beard and Dess define business level strategy in terms of "variation in firm characteristics relevant to competitive success or failure within a given industry."

Functional level strategy is when the company is establishing the required action programs in order to fulfil the *corporate objectives* and *business objectives*. The corporate objectives and business objectives refer to the upper strategy levels.

When distinguishing between the definitions of *corporate level* and *business level* strategies, the former concentrates more broadly on the overall setting of different businesses and considers the different businesses in which the company is involved at present and in the future, while the latter concentrates purely on one business area through competencies and competitive advantage. In turn, at the functional level, strategy means *action* in order to reach the strategic goals set at the upper levels.

In this study, the OSS business strategies are explored primarily at the business level of the target companies. This study targets open source business strategies from the viewpoint of companies that do not base their businesses entirely on OSS but rather see OSS as an emerging opportunity and promise to gain a competitive advantage. In this study, the business level strategy is therefore an appropriate starting point for investigating OSS-based businesses strategies.

3.4.2. Business Model vs Business Strategy

"Today, 'business model' and 'strategy' are among the most sloppily used terms in business; they are often stretched to mean everything - and end up meaning nothing." (Magretta)

In spite of a large volume of research on the distinction between the concepts of business model and strategy, there is no consensus on the issue. Stähler points out that both concepts are often used to refer to everything that is believed to give a competitive advantage. A practical distinction describes the business model as a system that shows how the pieces of a business fit together, while strategy also includes competition (Magretta). Osterwalder distinguishes between the two concepts by stating that strategy includes execution and implementation, while the business model is more concerned with the way a business works as a system. They also note that the business model implementation or execution is a widely neglected issue in current literature.

Seddon and Lewis conclude that "business model is an abstract representation of some aspect of a firm's strategy; it outlines the essential details one needs to know to understand how a firm can successfully deliver value to its customers."

In the following section, some selected open-source-based business models and strategies are presented.

3.4.3. Open Source Business Models and Strategies

Since the turn of the century, a great deal of research has emerged in the field of open source business dynamics. Most researchers (Hecker, Raymond) have concentrated on the classification of different open-source-based business models. Very few surveys concentrate on the business strategies within the OSS phenomenon however.

In this section, two sets of OSS-related business models and one set of OSS-related business strategies are presented in detail. These models and strategies partly overlap. It is admitted here that these are not the only possible models or strategies for OSS-based business, but the chosen examples are enough to provide a comprehensive picture of the current research on open source business dynamics and possibilities.

A. Heckers' Open Source Business Models

Following Raymond (1999), Frank Hecker was among the first researchers to categorise and present open source business models. His business model classification was probably the most classic at the time.

According to Hecker (1999), the models are:

1) *Support Sellers* - in this model, revenue is generated by promoting a free product and selling related supporting services such as distribution media, branding, training, consulting, custom development and post-sales support. According to Weber, it is almost always more beneficial to outsource support services than to organise comparable functionality in-house. Red Hat is probably the best known company to use this model by providing support for open source solutions such as Linux (Fitzgerald).

2) *Loss Leader* - in this model, a free open source product is used as a loss leader for commercial software and revenues are generated from the sales of commercial software. The idea is to generate interest through a free product and to sell related proprietary products for revenue. For example, Sendmail acts as a loss leader for Sendmail Pro, a commercial product with extra functionality (Fitzgerald).

3) *Widget frosting* - in this model, a company primarily sells hardware and provides OSS such as drivers and interfaces to make their hardware more compatible and consequently more valuable. This model is especially relevant to open source business strategies in the domain of embedded systems.

4) *Accessorising* - a company generates revenues by selling complementary products (or services) such as books, manuals and other physical items associated with the OSS (Mustonen, 2003). One example is O'Reilly & Associates, whose core business is in publishing books and manuals that support different OSS packages.

5) *Service enabler* - software is promoted free in order to enable access to an online service (typically owned by the software producer). Revenues are generated from the use of this enabled service. For example, Spotify, an OSS-based digital music player, enables online access to a comprehensive music library. The basic service is free, while revenues are generated by selling a premium version with a higher transfer rate of 320kbs without advertising. This business model is also known as SaaS.

6) *Brand licensing* - the company owns the brand name of an OSS product but does not own the source code (Weber). The owner of the brand name generates revenue by selling trademarks and brand name rights to customers who create derivative products.

7) *Sell it, free it* - similar to the *loss leader* model but extended to a product life cycle in which the release of a new commercial version opens the source code of the previous commercial version. In other words, the OSS product functions as a loss leader for the next-generation of commercial product (Weber).

8) *Software franchising* - combines several of the preceding models, especially *brand*

licensing and *support sellers*. A company authorises others to use its brand names and trademarks to create custom software development, in particular geographic areas or vertical markets. The offering may also contain training and other related services.

In addition to the above models, Hecker presents two hybrid models that he does not explicitly name. In the case of *dual licensing*, the company releases two versions of the software under different licensing schemes: open and proprietary. The basic idea lies in differentiating between users and types of use. In the second, *restricted redistribution* model, the company opens the source code and allows evaluation of the software but restricts the redistribution of modified versions in some way.

The above business model concepts are ideal types, and many companies combine elements of two or more of these models. For example, Bitkeeper.com with its Bitkeeper code base management tool applies elements of the *loss leader*, *service enabler* and *sell it, free it* models. The basic idea lies in the smart licensing strategy, which enforces the modified versions of the code to pass a regression test to demonstrate compatibility. In the case of failure, the modified version cannot be used at all. The open source version of Bitkeeper has a logging system that is openly available on the web. Commercial Bitkeeper users are willing to pay to keep their logs closed to the public. In other words, the core idea is to build an excellent product with strong demand and then identify a feature that benefits the OSS world and that customers find disturbing. Revenues are generated as the disturbing feature is removed (Weber).

B. Krishnamurthys' Open Source Business Models

Krishnamurthy states that it is possible to build business based on open source strategies and suggests that OSS communities are remarkable competitors to the commercial companies and can even defeat them. Krishnamurthy proposes three basic open source business models:

distributor, software producer and *third-party service provider.*

Distributor

There are three different variables in this model:

1) *The Retail Service* - The simple idea behind this model is to package the OSS product and sell it. An example of this model is to provide the OSS product on CD instead of as an online download. According to a survey of 113,794 Linux users 37.06% preferred to obtain Linux in the form of a CD to downloading it.

2) *Providing support services to enterprise customers* - large companies that need support for OSS are not willing to wait for community support. First, 'Time is money' and second, community support may not always meet quality requirements. Thus, there is a good chance of generating revenue by providing support services including installation, answering technical questions and training.

3) *Upgrade Services* - by acting as an ASP, distributors can help their clients to obtain the latest product upgrades seamlessly.

Software Producer (Non-GPL and GPL model)

1) In the *non-GPL model,* benefits can be gained in two ways. First, some source code of existing OSS can be incorporated into a larger code base in order to create a new product. Second, the entire open source product may be bundled with existing products. As the licence, instead of being the GPL, is of the unrestricted kind, the source code of the derived software product does not need to be disclosed. One application of this model would be to incorporate code from BSD-licensed OSS into a commercial product.

2) In the *GPL model,* the producer may sell the product but at the same time be forced to release the source code of the derived product. The main benefit of this model is community-accelerated innovation owing to rapid feedback and input.
The main differences between the non-GPL and GPL models may be presented in terms of what the producer expects from the customer. The producer of the non-GPL software expects the customer only to use the product while in the GPL model; the customer is expected to cooperate and engage in a two-way conversation.

Third-Party Service Provider

The basic idea behind this model is to have one single revenue-stream, i.e., *service*. The service provider is not concerned with where the software comes from. For this model, two different strategies are available: selling software with service and selling just services. In the first case, the software producer can combine OSS and propriety software into a well-integrated package and top it off with service. As many proprietary software producers have services as a secondary revenue stream, it provides a potential approach to any kind of software.

Cusumano distinguishes between the software business strategies of *products strategy* and *service strategy*. There is also a continuum of *hybrid strategies* in which companies use strategies somewhere between the two extremes. Cusumano suggests that plain product companies inevitably become service or hybrid solution providers and points out that the service approach could, especially in bad economic times, mean "the difference between success and failure as a company."

Krishnamurthy's business models do not include purely hardware-related models. These models may provide some ideas that are also applicable to the embedded system's domain. For example, the models labelled as *providing support services to enterprise customers* and *upgrade services* may be beneficial in the context of embedded systems.

Γ. Koenigs' Seven Open Source Business Strategies

Open source presents a big potential competitive advantage, not only for traditional software companies but also for hardware vendors and vendors of complementary and substitute services. Moreover, business managers should understand open source business strategies and decide which strategies are feasible. Accordingly, investors should be aware of different open source business strategies while evaluating their target companies. Seven open source business strategies are explained in this section.

1) The *optimisation strategy* is based on Christensen's Law of Conservation of Modularity. According to this law, modular and conformable layers are unprofitable, whereas in adjacent software layers, in which applications are optimised to achieve greater value, better pricing power exists.

2) Under the *dual licensing strategy,* the software vendor typically offers more than one licensing scheme. Customers may choose between commercial and free licensing options. Free licensing may include some limitations or restrictions, for example, distributed modifications must also be kept open sourced, which prevents competition. The commercial licensing option gives the customer commercial distribution rights and a larger set of features, albeit against financial compensation. Furthermore, the dual licensing strategy enables building of a highly defensible market position while offering the benefits of open source development.

3) The *subscription strategy* is parallel to Hecker's *support sellers* model, in which the company offers OSS-related maintenance and support.

4) In the *consulting strategy,* revenues are generated simply by providing OSS-specific consultancy. Companies applying this strategy are not necessarily software producers.

5) In the *patronage strategy,* the company contributes time, energy, developers and code to OSS projects to drive standards adoption or commoditise a particular layer of the software stack in order to eliminate competitors who benefit from that layer. To succeed in this strategy, the company must guarantee community management leadership and consistency. The patronage strategy is used by companies such as IBM, HP, Sun Microsystems and SAP .

6) The *hosted strategy* is categorised further into three subcategories: *ASP*, *transaction* and *advertising*. In the case of *ASP*, often referred to as SaaS, the service provider may use GPL-licensed OSS internally without restrictions and without sharing the modified source code back to the OSS community.

7) The general idea of the *embedded strategy* is to gain cost advantages by using OSS in embedded systems. Linux is the best-known example of software used in embedded systems. With low costs, a large supporting community and technical advantages such as stability, compatibility and advanced network capabilities, Linux can deliver great value for embedded markets.

Δ. Wests' Open Source Strategies

West studied three multinational companies and their software-related business strategies. He recognised that IBM, Apple and Sun used different strategies, to some extent. While IBM adopted Linux on all its hardware platforms, Apple and Sun adapted hybrid strategies, which are presented briefly next.

1) *Opening parts* - "waving control of commodity layer(s) of the platform, while retaining full control of other layers that presumably provide greater opportunities for differentiation". Apple used this strategy by basing its Mac OS X operating system on open-source-based Free BSD. Free BSD acts as a commodity layer, while the application layer remains closed source.

2) *Partly open* - "disclosing technology under such restrictions that it provides value to customers while making it difficult for it to be directly employed by competitors"
Sun, in turn, used this strategy as it opened up the entire source code of Java. It also opened its Solaris operating system but under a restrictive SCSL licence.

3.5. Summary

Amongst the presented business model frameworks, there are several that can be associated with this study. When comparing possible candidates, it soon became clear that Osterwalder's conceptualisation offers somewhat better grounds for the empirical part of this study than the other options.

The number of different variables or *components* in the business model framework is also meaningful. In terms of complexity, Osterwalder's framework is suitable for this study. It encompasses enough components to provide a reasonably broad variety of perspectives when examining an ICT company. Concurrently, it is not too complex in terms of having too many components.

Osterwalder original business model has been used previously in the research context of the OSS business dynamics by several authors, for example, Seppänen, Helander, and Mäkinen (2007) in their study on business models in the context of OSS value creation. They propose that there is no need for a context-specific business model and that, regardless of industry, a generic business model should engage the same components.
In addition, Feller, Finnegan, and Hayes (2008) used the e-business model ontology by Osterwalder and Pigneur as a lens to investigate business models in the context of OSS development.

The above open source business models and strategies are presented according to the interpretations by their original authors, which means that some of them are labelled as strategies and some as business models. In this study, however, they are all considered business models, as they are general in nature and therefore suitable for several companies. Furthermore, as the focus of this study is primarily on strategies, business models are merely considered frameworks. Meaning, the business model comes first and the strategy follows. The strategy thereby constitutes action, implementation and competition in order to gain a competitive advantage.

4. Open Source Challenges and the Resource Based View

The last theory chapter presents the theoretical framework of this study along with a brief analysis of open-source-related challenges, resources and capabilities.

Firstly, key literature findings on open source development and management-related challenges are presented here.

Secondly, the RBV, including the dynamic capability approach, is discussed.

Thirdly, the idea of answering open-source-related challenges with resources and capabilities is presented.

Lastly, the theoretical framework for analysing the data and answering the research question posed in this study is constructed.

4.1. Open Source Challenges

There is no explicit theory relating to the terms *challenge* or *business challenge*. The noun *challenge* relates to "something needing great mental or physical effort in order to be done successfully and which therefore tests a person's ability" (Cambridge Dictionaries Online, 2010). By the term challenge we can also refer to an issue or problem that a company must somehow overcome in order to incorporate OSS beneficially into its own application and domain, i.e., in the context of embedded systems.

We will try to identify the open-source-specific challenges that have not yet been comprehensively categorised or summarised in preceding researches.

4.1.1. Getting Started with OSS

In their extensive case study, Ven and Mannaert (2008, p. 995) investigate the use of OSS by ISVs and propose that companies decide carefully in which projects they invest time. They categorise these challenges under "becoming acquainted with OSS projects":

1) *Finding a way to the OSS community* - OSS projects are well governed and have a clear hierarchy. It is not always easy to start, as OSS projects have formal and informal policies that determine such issues as how to become a member and how patches should be submitted.

2) *Time investments* - There is a plethora of different open source components and each one has its own specifics. The time investment factor should therefore be considered, especially by companies with limited resources.

3) *Learning period* - Even for experienced programmers in commercial companies, the various OSS community procedures are challenging in the beginning. It can be difficult to obtain patches that meet the quality requirements set by the community.

4.1.2. Associating Business Activities with Open Source

Companies, and even governments, are seeking opportunities to release their existing proprietary source code to create new open source projects. Examples include IBM (Eclipse), Sun (OpenOffice) and Netscape (Mozilla). The changing of established proprietary projects leads to different challenges to those faced by community-founded projects. In their study, West and O'Mahony contrast community building in sponsored and community-founded OSS projects. They conclude that "to the degree that sponsors assert unilateral on-going technical leadership and control, sponsored projects may have difficulty recruiting external contributors." The relationships between the sponsor and the community are likely to raise questions about ownership, rights and control. The sponsor's incentive to control the open source project should therefore be weighed carefully against providing incentives for participants to contribute to the community.

4.1.3. Revealing the Source Code

While planning their open-source-based strategies, companies need to decide how much or which part of the software is appropriate to reveal. While they want to protect their IPR, they find the benefits of OSS development (maintenance, development support, etc.) rather appealing. Henkel executed a quantitative study revealing the pattern of company developed innovations within embedded Linux. The study revealed that companies contribute about 50% of their own development back to the Linux community and subsequently gain development support from other companies. Companies address the issue in the form of *selective revealing*, as they are concurrently in need of protecting their intellectual property. Furthermore, the conflict between the downsides and benefits of openness appears to be manageable, and the revealing is intentional.

4.1.4. Legal Challenges

"These licenses, it should be acknowledged, are complex legal documents that have not yet been tested in court." (Lerner)

Several cases have raised questions concerning the legal use of GPL-licensed software. For example, the distinction between static and dynamic linking is often fuzzy, which could be the case with the GPL and the LGPL. In some situations, the FSF has brought possible violators into compliance by threading them with a lawsuit. Issues have often been settled behind closed doors, however, outside the courtrooms. Bargaining outside the courts has led to a lack of generalised information on possible legal consequences and sanctions, thus increasing the uncertainty surrounding the use of GPL-licensed software.

The situation has evolved slightly since the time Lerner stated the opening quotation in this section. In fact, more recently, there has been at least one incident in which the exploitation of GPL-licensed source code in proprietary products has led to a court case. This episode occurred between D-Link and the gpl-violations.org and the result was favourable to the latter.

Issues may also rise when software components of two or more programs, under different OSS licensing schemes, are fused together into a new program. The integrator must investigate carefully whether the licences are actually compatible, as the distribution of the new program may result in copyright infringement. The compatibility of different OSS licences in several scenarios is obvious, although "it is much easier to describe those licenses that are incompatible than to assert with any assurance that two licenses are compatible." Consequently, it is endorsed consulting attorneys in complicated licensing situations for a thorough examination of the applicable licences (St. Laurent).

4.1.5. Motivation of the Contributors

The motivation of the contributors constitutes a core issue in the research into the OSS phenomenon (von Krogh & von Hippel). As discussed earlier in this study , there are a number of different motives, intrinsic and extrinsic, that make the programmers tick. For a commercial company, motivation-related issues therefore form various challenges, which may be difficult to overcome.

The different motivation bases, e.g., companies are primarily profit-driven and communities are motivated by social factors, may generate problems between these parties. In order to maintain good relationships with communities, companies may be prepared to invest considerable resources, e.g., time and money in the matter.

According to Katsamakas and Georgantzas, intrinsically motivated developers tend to switch between different projects and extrinsically motivated developers tend to exit the projects entirely once they reach their goals, e.g., gain enough of a reputation etc. Consequently, this may implicate that developers are perhaps less committed to their work in OSS communities than developers in commercial companies.

Moreover, if a community has several unsuccessful projects, it may be difficult to keep the community attractive to developers. In the case of success, however, the exit of extrinsically motivated developers may have a negative effect on the long-term stability of the community.

4.1.6. Maintaining and Contributing OSS Component Modifications

When using OSS, companies need to learn new organisational practices such as, for example, how to manage (maintenance) open source code. Basically, companies are not eager to invest too much effort in OSS component maintenance and prefer to contribute modifications to the OSS projects, as some maintenance tasks will be undertaken by the OSS community on behalf of the company. In his article on the use of embedded Linux, Henkel notes that companies tend to contribute modifications on the basis of their usefulness to others and keep specific modifications private.

Even if the modification is accepted by the OSS project, however, additional effort is often required from the company. In some cases, the patches must be made more generic to be compatible with the OSS project, though the components should still remain compatible with company's own application or use. Another case is when the company contributes a specific patch or component and becomes the modifier within the OSS project.

OSS component maintenance can be challenging for software vendors for two contradictory reasons.
First, there are a number of barriers that prevent companies from contributing the modifications back to the community.
Second, there are possible operational and business risks in the case of not contributing the code back to the community. As a barrier, especially in the case of smaller companies, the modifications may be specific and targeted at niche markets, which may lead to the modifications ending up being unsupported by the OSS project.

Risks of not contributing the code include a situation in which the OSS project finds an alternative route around the problem. This alternative route may be incompatible with the company's solution and the company may therefore be forced to maintain a parallel development or change its product to be compatible with the OSS community branch. Either case incurs additional work and costs. Another case when not contributing raises problems is obsolete APIs, when the OSS project decides to alter or remove some legacy APIs and the company is cut out of future versions of the OSS component. This may lead to a situation in which the company once again ends up with a parallel development branch and consequently additional costs.

Companies often develop their own applications on top of OSS software. These OSS components are not always suitable for the project in hand however. In some cases, OSS does not provide all the required functionality for a specific market niche in which the vendor is active. A particular 'glue code' is sometimes required to make OSS components compatible. Hence, the OSS component needs to be customised by adding the required functionality.

4.1.7. Projects' Forking

Forking of OSS projects is a situation in which disagreements over product design cause a certain project to divide into two or more competing branches, i.e., different variants. This may be a result of weak leadership or a lack of credibility in project management. Developers may have conflicting interests concerning the evolution of the technology. It is also possible that the project group's internal egoistic interests prevent it from admitting that the *competing* approach is more promising.

Ways of avoiding forking include strong leadership with accurate milestones, regular meetings and other project-leading attributes. The key to successful leadership is the programmers' trust in the leadership. This means that the programmers must believe that the motives of the leadership are in line with theirs and not, for example, commercially or politically driven (Lerner & Tirole).

4.1.8. Embedded Systems Specific Challenges

The remarkable growth of the computational power of computers and the increased capacity of flash memory cards have boosted the space available for operating systems (i.e., Linux) in the domain of embedded systems. Performance trade-offs are the most common constraints on the development of embedded systems however. The computing power available in an embedded device is typically low due to hardware architecture characteristics and issues pertaining to power management, especially in battery-operated devices. ARM architectures have become popular in such devices due to their low power consumption.

Furthermore, as an UNIX-like kernel, which is designed to provide fair sharing of resources among many users and applications, Linux does not fully guarantee real-time performance requirements. In real-time robotics studies, however, discoveries showed that there are currently commercial as well as open-source-based solutions to real-time performance issues.

4.2. Open Source Resources and Capabilities

In this section, the concepts of resources, RBV, capabilities and dynamic capabilities are explained briefly.

4.2.1. Resource-Based View

The origins of the RBV may be traced to three different research perspectives: the *traditional strategy*, *organisational economics* and *industrial organisation research*.

The RBV describes the relevance of resources to a company to gain a sustained competitive advantage. A company's resources include all assets, capabilities, organizational processes, firm attributes, information, knowledge, etc. controlled by a firm that enable the firm to conceive of and implement strategies that improve its efficiency and effectiveness.
A sustained competitive advantage derives from the resources and capabilities a firm controls that are valuable, rare, imperfectly imitable, and not substitutable. Also, resources and capabilities can be viewed as bundles of tangible and intangible assets, including firm's management skills, its organizational processes and routines, and the information and knowledge it controls.

A company's resources can also be categorised into three groups:
physical capital resources, human capital resources and *organisational capital resources*. While physical capital resources include the technology and equipment used, human capital resources are the characteristics related to managers and personnel, such as training, experience, relationships and intelligence. In turn, organisational capital resources include a company's formal and informal structures of reporting, information planning, and internal and external relationships.

According to others, the concept of resources is divided into two separate concepts of *resources* and *capabilities*. Resources are defined as "stocks of available factors that are owned or controlled" by the company and state that resources are tradable and non-company specific. In contrast, capabilities refer to a company's capacity to deploy resources, usually in combination, using organizational processes, for achieving desired goals. Furthermore, in contrast to resources, capabilities are company-specific. Moreover, resources consist of factors such as tradable expertise, financial and physical assets, and human capital.

Depending on how one looks at it, a company's resources can have many categorisations. Like for example into four classes: human, physical, technological and financial capital available to the firm at any moment in time for its objectives. This classification attempts to explain a company's behaviour in any one specific situation, not in multiple situations.

4.2.2. Capabilities and Dynamic Capabilities

The concept of *capability* is commonly defined as "the quality of being capable" (Dictionary.com). The ones who first introduced the concept of *dynamic capabilities* and defined it as a "firm's ability to integrate, build and reconfigure internal and external competencies to address rapidly changing environments". Some argue that the dynamic capability view could overcome the limitations of the RBV.

Accordingly, Eisenhardt and Martin define dynamic capabilities as company's "processes that use resources - specifically the processes to integrate, reconfigure, gain and release resources - to match and even create market change". Furthermore, dynamic capabilities are the "organizational and strategic routines by which firms achieve new resource configurations as markets emerge, collide, split, evolve, and die".

According to Winter while operational capabilities enable an organisation to earn a living in the present, dynamic capabilities are required in the case of change. Moreover, dynamic capabilities are those that are required to change the product, production and the scale of the markets.

More recently, Teece categorised the main dynamic capabilities into three groups:

- *sensing opportunities and threads,*
- *seizing opportunities,*
- *managing threads and reconfiguration.*

The first group, *sensing opportunities and threads*, involves investment in research and related activities, such as scanning, creation, learning and interpretive activity.

Seizing opportunities comes into the picture once a new (technological or market) opportunity is sensed. Seizing should be addressed through new products, processes or services. Once again, seizing often requires investments in development and commercialisation activity.

The third group of *managing threads and reconfiguration* involves "successful identification and calibration of technological and market opportunities, the judicious selection of technologies and product attributes, the design of business models, and the commitment of (financial) resources to investment opportunities", leading to enterprise growth and profitability.

In the context of the ICT industry, it is important to acknowledge the dynamic capabilities approach, because the industry is typically characterised by rapid technological changes. Moreover, as the focus of this study is on open source business strategies, which in turn often involves changes in the environment, it is appropriate to relate the dynamic capabilities approach to this study. In other words, the evaluation of dynamic capabilities is important for companies that consider OSS-based business strategies.

4.3. Association of Challenges with Resources and Capabilities

Current literature seems to lack to-the-point research on resources and (dynamic) capabilities in the contexts of open source business strategies and embedded systems. Crowston and Scozzi base their study of the identification of success factors of OSS projects on the theory of competence rallying (CR). They tackle the issue purely based on success factors, however, which could be related to resources and capabilities. Similarly, this study tries to explore the success factors of open source business strategies by first identifying the relevant challenge areas and then the corresponding resources and capabilities. In this study, a company's OSS-related resources, capabilities and dynamic capabilities are the vital ingredients for the successful application of OSS-based business strategies.

4.4. OSS Business Strategy Assessment Model in the Domain of Embedded Systems

In this section, the theoretical framework, i.e., the model for open source business strategy assessment in the context of embedded systems is constructed. The primary purpose of this model is to analyse the empirical data and consequently answer the research questions. As an application or tool, this model should also be able to provide means for ICT companies to assess their OSS-related business strategies, whether the company already uses OSS or is about to do so in the near future.

4.4.1. Definitions

The various definitions used in this study have already been discussed in the corresponding theory chapters. The chosen definitions relating to the theoretical framework in this study are however presented and summarised in this section.

Business Model

The chosen definition for the term *business model* is adapted from Osterwalder and will be defined as:

> *Business model is a conceptual tool that contains a set of elements and their relationships and allows expressing the business logic of a specific firm. It is a description of the value a company offers to one or several segments of customers and of the architecture of the firm and its network of partners for creating, marketing, and delivering this value and relationship capital, to generate profitable and sustainable revenue streams.*

Strategy

The concept of *strategy* is derived by distinguishing between the concepts of *business model* and *strategy*. Whereas business model relates to a conceptual tool that contains a set of elements, strategy includes implementation and competition. Following Magretta, it will be defined as follows:

> *Strategy includes the practical means of enforcing and implementing individual business activities for competitive advantage by using the business model as an underlying framework, where the business model is of general in nature and therefore suitable for several companies.*

The emphasis is on the word *individual*, as strategy is considered company-specific.

Challenge

The concept of *challenge* (Cambridge Dictionaries Online, 2010) is discussed in Chapter 6. It will be described in terms of the following definition:

> *Challenge constitutes an issue or problem a company must somehow overcome. In this context, the emphasis is in OSS specific challenges, which are critical for a company to tackle in order to beneficially incorporate OSS in its own application and in the domain of embedded systems.*

Resource

Resources will be defined by following Amit and Shoemaker :

> *Resources are tradable, non-company specific stocks of available factors (assets), owned or controlled by the company. Resources consist of such factors as tradable know-how, financial and physical assets, and human capital.*

Furthermore, the distinction between resources and capabilities will be applied according to Amit and Shoemaker (1993, p. 35). The following distinction is equal to the one applied by Väyrynen :

> *While resources are tradable and non-specific to a company, capabilities are company-specific.*

Operational Capability and Dynamic Capability

The distinction between operational capabilities and dynamic capabilities will be made according to Winter:

> *While operational capabilities enable an organisation to earn a living in the*

> *present, dynamic capabilities are required in the case of change. In addition, dynamic capabilities are required in order to change the product, the production, and the scale of the markets.*

Dynamic capabilities are further defined according to Eisenhardt and Martin as:

> *Company's processes, that integrate, reconfigure, gain and release resources, in order to match and create market change. Furthermore, dynamic capabilities are the organizational and strategic routines by which firms achieve new resource configurations as markets emerge, collide, split, evolve, and die.*

4.4.2 Business Model Framework

In this study, the business model framework by Osterwalder will constitute the underlying core logic for the assessment of open source business strategies. The framework will be used in order to understand a company's underlying structure and make sure that every meaningful aspect of a company's business logic will be covered. The nine building blocks or components of the model constitute a comprehensive and sufficiently broad picture of the company's business structure, being neither too complicated nor oversimplified for the task in hand. It is acknowledged that Osterwalder's business model framework is not the only possible solution for this study.

4.4.3. Business Model Components

The nine business model components (Osterwalder):

1) The *value proposition* of what is offered to the market
2) The *segment(s) of clients* that is/are addressed by the value proposition
3) The *communication and distribution channels* to reach clients and offer them the value proposition
4) The *relationships* established with *clients*
5) The *key resources* needed to make the business model possible
6) The *key activities* necessary to implement the business model
7) The key partners and their motivations to participate in the business model (*partner network*)
8) The revenue streams (*revenue flows*) generated by the business model (constituting the revenue model)
9) The *cost structure* resulting from the business model

4.4.4. Challenges, Resources and Capabilities

The term challenge is defined as an issue or problem a company must somehow overcome in order to incorporate OSS beneficially in its own application and domain, specifically within the domain of embedded systems. Open-source-specific resources will be considered important assets that are needed to overcome open-source-related challenges. Equally, capabilities, whether operational or dynamic, are essential abilities and processes that help companies achieve their open-source-specific goals and match changes when using OSS.

5. Case Studies of Open Innovations Outside the Mobile Industry.

Though open source software is the norm today, open source hardware design or embedded system is still rare but does exist. Open source in mobile industry is still a pretty recent phenomenon. To understand how the mobile handset manufactures can ride on the wave of open source movement in mobile industry, how they can capture value from the "openness", we can examine a few business cases outside mobile industry that used the open source or open innovation. Then we will apply the lessons learned to mobile industry.

5.1. Open Source Development (outside the mobile industry)

Since our primary interest in this study is technology strategy and business model, the history and evolution of open source software development is out of scope for this discussion. There are plenty researches that have been conducted or are being conducted, and we will only touch the area of motivations and business impacts because this study is about technology strategy. Early studies have found three general categories of contributor motivations;

- Direct utility: e.g. the end product can be directly used by the contributors
- Intrinsic benefit, such as learning a new skill or developing a network
- Personal satisfaction by signaling one's capabilities to gain respect from peers; self fulfillment; altruism; pleasure derived from "playing".

Those early studies failed to point out real motivations behind the involvement of some companies with commercial interest lately. Those firms' incentive to participate in the open source software development is quite different from those of individuals. Their decision is purely based on their technology strategy and business strategy.

Some companies participate in the open source community to win adoption of their technology and to gain first-mover advantage. Sharing may lead to the establishment of a de facto standard or the generation of network effects; some companies may contribute to create demands for other products in their core business. On the other hand, some companies participate because they have no choice. Other benefits for the firms include symbiotic or parasitic gains from community, shaping industry and pre-empt competitors and benefit spillover to participants and non- participants alike (West).

We will examine only a few cases to see why some commercial software firms adopted the open source model and how the open source model helped their business. There's no question about the social effect of the open source community, however we are more interested in the business models, specifically the lessons that can be applied to mobile industry.

5.1.1. Red Hat: Building New Business from Nothing

Red Hat, Inc. is the market leader in open source software systems for mainframes, servers, workstation and embedded device. Red Hat's Open Source strategy offers customers a long term plan for building infrastructures that are based on and leverage open source technologies with focus on security and ease of management. Red Hat also offers support, training and consulting services to its customers worldwide and through top-tier partnerships. Red Hat was incorporated in 1995 by Bob Young and Marc Ewing. Red Hat went public on August 11, 1999, the eighth biggest first-day gain in the history of Wall Street. In August 2002 Red Hat announces the Red Hat Alliance. Committed partners include AliaslWavefront, BMC, Borland, Checkpoint, Computer Associates, Dell, HP, IBM, Legato Systems, Novell, Oracle, Rogue Wave, Softimage, Synopsys, TIBCO, and VERITAS. Red Hat officially announces its entry into the enterprise software market with the "Enterprise-Ready Linux" event with partners Oracle and Dell. Red Hat achieved first profitable quarter in 2001. Red Hat stock became part of the NASDAQ-100 on December 19, 2005.

In their very early days, Red Hat distributed and sold the shrink-wrapped CDs of tRed Hat Linux distribution. Even though the same software could be downloaded free from their servers and many mirror sites, people opted to buy the CDs for the convenience. As the internet bandwidth and storage capacity increased dramatically, it's more convenient for users to download the new CD images from internet. As a result, that stream of revenue to sell packaged Linux diminished with the bust of dot com bubble. Today Red Hat partly operates on a professional open-source business model (free software with paid professional services) based on professional quality assurance services, and subscription based customer support. Customers pay one fixed price for unlimited access to services such as Red Hat Network and up to 24x7 supports.

Now the interesting question is, Red Hat does not have any proprietary technology and it doesn't provide services that the OEM (IBM, HP, Dell) can't provide, then how could it build a business from nothing to the size that almost matches Sun Microsystems?

The story written by the founder Robert Young "How Red Hat Software Stumbled Across a New Economic Model" in 1999 may shed some lights (Young 1999). At the time of founding, Young didn't understand that "bizarre economic model" either but decided to give it a try anyway. In the next couple of years, Red Hat developed a business plan that was compatible with the "bizarre business model" by trials and errors. To answer the question how to make money in free software, Young explained "While making money with free software is a challenge, the challenge is not necessarily greater than with proprietary software. In fact you make money in
free software exactly the same way you do it in proprietary software: by building a great product, marketing it with skill and imagination, looking after your customers, and thereby building a brand that stands for quality and customer service."

Another advice from Young in term of finding a new business model is, "You can't compete with a monopoly by playing the game by the monopolist's rules. The monopoly has the resources, the distribution channels, the R & D resources; in short, it just has too much strength. You compete with a monopoly by changing the rules of the game into a set that favors your strengths"

At this writing, Red Hat has a market capitalization of $3.7 Billion. For comparison, Sun Microsystems' market capitalization is $6.68 Billion (they have about the same market capitalization of $2.7 Billion at the lowest market level of January of 2009). To investors, Red Hat's business model is transparent and understood, whereas Sun's is not. As Red Hat's direct competitor, Sun Microsystems also has an open source support operation but only account for a small portion of Sun's total revenue. It's also interesting to note that Red Hat's another direct competitor in the open source arena, IBM adopted a different strategy. Unlike Red Hat which has no proprietary system, IBM still maintain a large portfolio of proprietary products despite its significant participation in open source software community. IBM has not given away the value farm by creating open source software stacks that cannibalize or replace its proprietary value stack sales. How long they can maintain that advantage is a good question to ask.

5.1.2. Novell

Just like IBM which had its near-death experience in 1990s', Novell also had similar experience when it failed to anticipate the market change in 1990s. Today's Novell is a complete new company operating on a completely new business model.

Novell was found in 1983 primarily as a hardware company but soon abandoned hardware business and focused on software for networking. At that time PC industry was booming. Firms became increasing interested in ways to connect their PCs. Netware was released in 1983 and soon became the leader of networking market. By 1988 Novell had a 50% share of PC networking market. Novell did extremely well throughout the 1980s, acting aggressively to increase its market share initially by selling the expensive Ethernet cards at cost; by the early 1990s, Novell NetWare held nearly 70 percent of the network operating system market. During that period, Novell took advantage of its position in the computer networking industry and created a massive distribution network involving 13,000 independent distributors. Value-added resellers range from mass-market discounter to high-level system integrators. Novell intentionally left some gaps in the NetWare product line so their partners could fill the gap and capture the value. As a result their channel partners had vested interest in Novell's success. These massive channels turned out to be the best asset of Novell in 2000s when Novell reinvented itself with a totally new business model.

Despite its success and monopoly status in the networking industry, Novell failed to recognize the change in its market in early 1990s. Their market share plummeted because NetWare missed out on the huge server application wave. The release of Microsoft's Windows NT with built-in networking capability posed perhaps the largest threat Novell had ever faced. Novell tried to diversify by purchasing WordPerfect and Quattro Pro but that strategy did not work out well.

Seeing its market diminishing, Novell decided to reincarnate itself at the end of 1990s and soon moved to Linux domain. In 2003, Novell acquired SuSE, a developer of a leading Linux distribution and Ximian, a developer of open source Linux applications. However Novell adopted a completely different open source strategy and business model compared to those of Red Hat or IBM. Today Novell is a mixed-source company mixing open source with proprietary software to architect their product line. The company no long relies on the traditional open-source business model that depends on support and service revenue streams. The company also made it clear its plans to shift from direct sales and service to a channel-based business model. The company is no longer the same company at its peak time, but it survived. So we can say the mixed-source and channel-based business model are the strategies Novell is betting on today and so far they are working.

Lessons Learned

1. Change the mentality. Company should not assume it is easier to make money selling proprietary software. The IP model of software development and marketing is proven to hard to make living. So companies should get used to the free software, and participate in the open source development. During the process, they may be able to figure out a business model to make living. Novell didn't realize the value of its channels established in the past when it entered the open source software domain. But they figured it out eventually. In the past, the vertically integrated R&D worked well, but companies shouldn't feel comfortable staying in the same environment. Try not to be the frog in the "boiling frog" story. As an example, Motorola felt so comfortable from its past achievement, it didn't move fast enough in react to the market change. As a result, its market share dropped significantly, and that trend is still continuing.

2. Embrace the change. It's quite common that companies get trapped by their own success. Once they establish a money making business model, they will only seek additional opportunities that fit that model. Companies not only need to get used to changes, but also need to welcome and seek changes. Some mobile handset companies are bound to face the same challenges Novell faced before. If they learn to embrace changes like Novell did, they may be able to survive.

3. Be proactive. Even though IBM survived after its transition of 1990s, it came to what it is today not at a cheap price. If they could react to the change in the industry earlier, their price would be much lower. The same principle applies to the cell phone industry. If manufacturers are proactive, they will anticipate the industry change, and then design a strategy according to the change. For example, if the consumers' buying choice is not based on technology or feature/price ratio, but the fashion or culture, then companies should shift their focus quickly from the technology centered R&D process to the user centered process.

5.2. Open Standard and Open Source in Hardware Development

While there are several hundred open source software projects in the world, there are very few open hardware projects. In the past, the lack of standard digital form to transfer information significantly hindered the collaboration between designers. With the adoption of standard hardware description language (HDL) such as Verilog, designers from anywhere in the world are able to design chips together. Still, compared to the popularity of open source software projects, open source hardware projects are still isolated cases probably due to the much smaller number of developers that are capable of designing chips compared to the number of software developers.

5.2.1. Sun's OpenSPARC

The OpenSPARC project (http://www.opensparc.net/), launched by Sun Microsystems to create an open source community centered around its "Niagara" Sparc T1 processors, is probably the most famous open source hardware design project up to now. SPARC stands for Scalable Processor ARChitecture. The technology is based on RISC. SPARC International is currently managing SPARC related IPs.

Sun Microsystems began shipping the UltraSPARC TI chip multithreaded (CMT) processor in

December 2005. Sun surprised the industry by announcing that it would not only ship the processor but also open source that processor. By March 2006, UltraSPARC T1 had been open- sourced in a distribution called OpenSPARC T1. In 2007, Sun began shipping its newer, more advanced UltraSPARC T2 processor, and open-sourced the bulk of that design as OpenSPARC.

The official goals of OpenSPARC Initiative state (www.opensparc.net):

- To significantly increase participation in processor architecture development and application design by making cutting-edge hardware intellectual property freely available.
- To eliminate barriers to the next big build-out of the Internet.
- To improve collaboration and cooperation among hardware designers.
- To enable community members to build on proven technology at a markedly lower cost.
- To encourage innovation.
- To foster bringing bold new products to market

However the real goal of Sun in disclosing microprocessor design data, which took its engineers significant efforts to develop, is to increase the amount of equipment using SPARC and the already disclosed OpenSolaris operating system (OS) by tapping the open course community. By leveraging the open source community, developers will be able to rapidly create new and more tightly integrated thread-rich applications at a markedly lower cost. Sun senior vice president and Fellow, Scalable Servers Group chief technologist Mike Splain, commented, "We wanted to lower the barriers to introducing the SPARC architecture to boost the number of system-on-chip (SoC) and other integrated circuits (IC) developed. If these chips are used in various products, it will increase our business opportunities.”

Lessons Learned

By giving away the technology that is not in the core business of the company does not necessarily cause the loss of the asset. On the other hand, with careful planning, by giving away a technology to build a community that has the potential to be the customer of the complement products. Manufacturing microprocessor is not the core business of Sun but the processor is a key component of its server that can distinguish Sun from its competitors. Sun derives its revenue from the sale of servers, operation systems and service contract. With a fraction of its original development cost, Sun can get better processors at cheaper price to boost its servers' performance. The mobile can adopt the same strategy. If the company bets on the mobile web as the next generation source of revenue, then it can design a mobile OS specific for mobile web. Since the hardware part will become commodity anyway, the company can open up the design of device, either the key components that are specialized for the mobile web OS, or the whole device.

5.2.2. Personal computers' Open Standard

The whole personal computer (PC) industry is about thirty years old. No other industry can match the innovations happened in PC industry in such short a period. Most early players in PC market no long exist today. Some such as Texas Instruments, Commodore, Amiga and Wang never adapted their software and hardware to the IBM standard. The proprietary system just didn't win the heart and soul of average PC users, so the market. A few survived reinvented themselves and now they are not the same companies as they used to be.

In the very early days, all the big players are vertically integrated companies. They designed and manufactured every component of a PC, from hardware to software and applications. IBM is a typical example. During 1990s, the competition forced the industry to shift from vertically integrated business model to horizontally integrated business model. The Firms won the competition are the firms which are specialized in PC components. Intel and Advanced Micro Devices (AMD) are specialized in CPU and flash chip design and manufacturing. They are the two CPU manufactures survived today. Nvidia and ATI (was acquired by AMD in 2006) are the only two graphic chip manufactures left today. IBM lost its edge during that period and was forced to exit PC industry. What brought IBM's success is exactly what almost killed IBM, the standardization of PC.

During 1980s, the IBM design was accepted as the industry standard which all manufactures essentially copied. IBM opened the interfaces and standards of all PC components. After 1987, IBM overplayed its role in defining the industry, and lost it position in marketplace. The open standards defined by IBM encouraged competition and innovation. The innovation encouraged by the open standard is also open innovation. Even though formal collaboration was not that common between competitors, they could easily learn others' design by reveres- engineering because all components conform to the same standard. Those competitions and innovation drove up features and drove down prices of PC components and PC itself dramatically in a short period. Soon PC became commodity. Nowadays, almost every family in United States owns at least one PC.

The mobile handset industry is experiencing exactly the same revolution at even faster speed. The standardization is inevitable. The open standard will encourage open innovation which in turn will turn cell phone components and cell phones into commodity. Most cell phone manufactures are still vertically integrated company. Sooner they realized that they have to abandon their vertically integrated business model, the better chance they can survive the competition. The current movement of open source mobile OS is just the beginning of the revolution that will disintegrate the industry giant like Nokia, Motorola etc.

5.3. Open Source in Embedded System Development

The open source or semi open source projects in the embedded system are more common than open source hardware design but less common than open source software. The reason is obvious. Embedded device has two pieces, the firmware which is the software that enables all the functions of the device, and the hardware piece which support the software. Most open source projects are done on the firmware part with some modifications on the hardware side.

IMost projects in this domain generally fall into the category of von Hippel's user centered innovation paradigm and are done by hacking communities. The lead users in the hacking communities are able to design much better products than the original manufactures using exactly the same piece of hardware. Unlike open source software communities, the motivation of those "hacker" is usually their personal needs. They are not satisfied with whatever the manufactures can provide so they start to invent themselves. Surprisingly, in most cases, manufactures haven't been able to capture the value from users' innovation.

5.3.1. DD-WRT and Buffalo Technology.

Home network equipment business today is commodity business. The US market is dominated by a few players, such as LinkSys (a Cisco subsidiary), DLink, Netgear etc. Buffalo Inc or Buffalo Technology (wholly owned subsidiary of Melco Holdings Inc, headquartered in Japan) is a multinational provider of innovative network solutions for the home and business - from wireless networking and storage to memory and multimedia devices. Buffalo branded routers are very popular among advanced users in the US due to certain models' compatibility with DD- WRT firmware.

DD-WRT (www.dd-wrt.com) is a free third party developed firmware released under the terms of the GPL for many ieee802.1 1a/b/g/h/n wireless routers based on a Broadcom or Atheros chip reference design. DD-WRT is a Linux-based firmware designed to replace the firmware that ships pre-installed on many commercial routers. This is done for a variety of reasons; including but not limited to the addition of many features which are not typically included in a manufacturer's router firmware but required by advanced users. Used and tested by hundreds of thousands users, DD-WRT firmware has much fewer bugs in many cases, DD-WRT firmware also adds stability over the stock firmware. In every aspect, DD-WRT is a much better firmware regarded by majority of users over the stock firmware. It is estimated that one million DD- WRT® based routers are already in use worldwide, DD-WRT versions up to v22 were based on the Alchemy firmware from Sveasoft, which in turn is based on the original GPL'd Linksys firmware and a number of other open source projects.

DD-WRT versions from v23 onwards, are almost completely rewritten. The Linux kernel part is based on the OpenWrt kernel which is a similar open source project. The project was created directly from Sveasoft's decision to start charging for their firmware, closing the door to open source. The firmware is currently maintained by BrainSlayer and he's drafting a different business model in order to support himself through this project. (source www.www-wrt.com).

The partnership with Buffalo Technology probably is in that plan. Despite the superior feature of DD-WRT firmware and it's compatibility with wide range of router hardware, the major market players haven't been able to capture the potential value except Buffalo Technology.

In October 2007, Buffalo Technology announced a strategic partnership with DD-WRT community to develop 802.11 g routers to provide small businesses and ISPs with a high-performance solution that is covered by Buffalo's two-year manufacturer's warranty and full tech support (Buffalo 2007). The partnership addresses the shortcomings of each side so they complement each other perfectly. On the Buffalo Technology side, they get the best firmware for their routers without development effort, and they also get the free technical support from the community. The technical support is usually a major headache for consumer network equipment manufacturers. On the DD-WRT side, the agreement addressed the hardware warranty issue which is the major concern for end users using third party firmware.

The question is, why only Buffalo Technology is willing to tap into the DD-WRT community to create win-win business model? The answer probably lies in the market position of each company. The larger companies, like LinkSys, DLink, have much larger product portfolio, and much larger distribution channels. When one product line is at the end of its life cycle, companies will abandon the support of that product and try to move users to a new product in order to generate new revenue. For those companies, value captured from DD-WRT may cannibalize the revenue from other product line. Buffalo Technology does not have that concern because they have much smaller market share.

For mobile industry, the same story may happen in the future. Smaller market players will jump on open source projects to increase their competiveness. If the dominant players do not participate early to pre-empt competition, then they are bound to lose market share in the long run. DD-WRT is a rare case that manufacturer and hacking community made a good business case. There are many hacking projects similar to DD-WRT that create great value for consumers, but they are ignored by the original equipment manufacturers, or sometimes their interest are not aligned with manufacturers interest. The following section will list a few projects if someone is interested in further reading.

5.3.2. XDA-Developers

XDA-Developers.com isa community dedicated to improving HTC branded smartphones. The website was started as a result of the lack of support from both service providers and manufactures. "Since we develop software for it, we need information, and nobody seemed eager or ready to give us what we needed. So we 'reverse-engineered' the devices, found a lot of information, and shared it with the world. But as our site grew we realized that lots of ordinary users were also suffering from a lack of support. They started using the xda-developers forum to communicate and before long the forum was as much a user forum as it was a developer forum".

The ROM (the firmware of a cell phone) developers are often called "chef' because they "cook" the customized ROMs for various phones. This community is not exactly an open source community because some applications or ROMs are not open-source but nonetheless they are all free and open to public. Because the existence of this community, HTC branded phones become highly sought-after smartphones among advanced users. It's not clear how many innovations HTC has assimilated; HTC has been unquestionably benefited by the existence of this community, albeit there are concerns about IP issue there because Windows Mobile OS is not open source.

5.3.3. NSLU2-Linux

NSLU2-Linux (www.nslu2-linux.org) is a development group and user community dedicated to the improvement, development and modification of the firmware and hardware of the Linksys NSLU2, the Synology DS101, the Iomega NAS 100d, the D-Link DSMG600, and other ixp4xx- based devices with large attached storage. The firmware developed by this community dramatically extended the functionality of original NSLU2 devices. LinkSys is aware of the activities of this community and DD-WRT community, but they didn't do anything to discourage them. It would be interesting to quote a comment from Mike Wagner, director of marketing for Linksys regarding all the activities going on with their NSLU2. "While Linksys does not support any of the alternate firmware available for the NSLU2, we are always delighted to see a product gain such widespread acceptance. Like the similar community that emerged to enhance the WRT54G before it, the creativity and ingenuity of Linksys customers inspires us to continually improve our products" (Buzbee 2006) . Nonetheless, LinkSys is benefited from those users innovation.

5.3.4. CHDK

CHDK (stands for Canon Hack Development Kit) is free software released under the GPL. CHDK is a firmware enhancement that operates on a number of Canon Cameras (Canon Point and Shoot digital camera only at this moment). CHDK gets loaded into your camera's memory upon boot up (either manually or automatically). It provides additional functionality beyond the currently provided by the native camera firmware (source http://chdk.wikia.com). The enhanced capabilities that CHDK provides are most likely to be of interest to experienced photographers, but they are not to the interest of Canon. Canon deliberately disabled certain features in some product lines in order to differentiate the market. The development of CHDK is likely to cannibalize Canon's sale in higher end camera model. So far Canon hasn't taken any action against the CHDK community.

5.3.5. Rockbox

Rockbox (www.rockbox.org) is an open source firmware for mp3 players, written from scratch. It runs on a wide range of players including Apple's iPod, Sandisk's Sansa series. The developers feel the original firmware is lacking some features and contains a number of annoying bugs that they don't want to live with. The Rockbox firmware was written from scratch so there's no IP issue. Not every manufacturer is lenient with Rockbox community. Some companies like Iriver and Archos specifically state that using Rockbox firmware will void warranty. Apple hasn't done anything against the project, but Apple is certainly not in favor of Rockbox. Apple is a company that doesn't want users to tell them what to do. They believe they can tell users what they really need.

Lesson Learned

From the open innovation cases we have examined in the embedded device industry, we can say that values created by user innovation haven't been captured in most cases. Smaller companies should actively participate in and guide the user innovation and seek the right opportunity. Dominant players should give the freedom to the user innovation. They may not see immediate commercial return from those innovations, but they never know if someday user innovation turns to disruptive technology.

5.4. Open Source Development (inside the mobile industry)

The industry trend is quite clear right now that the cell phone devices are going to two extreme: the low cost "dumb" phone and the feature-rich smartphone. The smartphone market share sets to surge in the next couple of years. By 2013, almost one out of four phones will be smartphone (See chapter 5.2). If counting the revenue instead of units sold, then smartphones will account for much larger market share due to the higher price. This growth, coupled with a fast pace of innovation, presents an excellent opportunity for handset manufactures to increase revenues and profits with higher prices higher margin devices.

5.4.1. Open Source Mobile Platforms

For the smartphone market, the ongoing battle is about the mobile operating system. Just like the early days of PC industry, there are many players in the mobile OS field. In fact the market is more fragmented today than early days of PC industry. We count at least eight OS platforms in today's smartphone, not including different flavors of Linux based mobile platform. Each platform offer different openness, from completely proprietary to completely open source.

1. Symbian: Symbian is an open source platform today. Nokia acquired all the shares of Symbian Ltd. in June 2008 and decided to open the platform. The Symbian Foundation was established as a non-profit foundation to "provide royalty-free open platform and accelerate innovation" with the intent to unite Symbian OS, S60, UIQ and MOAP(S) to create one open mobile software platform. Membership of this non-profit foundation will be open to all organizations.

2. RIM Blackberry: this is the most closed platform in all the smartphone OS. There's no sign that RIM will open the platform. The drawback of this OS is the lack of applications from third party. That landscape may change as RIM plans to launch a BlackBerry Storefront this year.

3. Microsoft's Windows Mobile: Originally designed for PPC/PDA, now one platform for both PDA and smartphone (the line is blurred anyway). Windows Mobile is proprietary system, but not as closed as RIM Blackberry because Microsoft opened some modules. The best thing about this OS is the sheer variety of available applications.

4. Apple's iPhone OS (Mac OS X): The new entrant in the market, but already took a large chunk of the smartphone market. iPhone OS is Unix based but not open.

5. Google's Android: the newest player in this field. Linux based, completely open source platform. Currently have a large developer community.

6. Palm OS (Palm's new webOS): the new Palm OS (named WebOS) for its Palm Pre smartphone is open source platform. Whether Palm is too late in the game is still not clear at this moment.

7. LiMo (Linux Mobile): LiMo platform is based on Linux and developed by LiMo Foundation. LiMo Foundation was founded in January 2007 by Motorola, NEC, NTT DoCoMo, Panasonic Mobile Communications, Samsung Electronics, and Vodafone with the

goal of establishing a globally competitive, consistent Linux-based operating system for mobile devices for the whole mobile industry.

8. Various Linux base OS from Trolltech, Access, Purple Labs, Open Plug, Ala Mobile, OpenMoko and Mizi Research. None of them have significant market share. Completely open source or semi-open source platform.

5.4.2. ShanZhai Phone Phenomena in China

ShanZhaiPhone, literally translated to "bandit cell phone", refers to cell phones manufactured by unauthorized or small-scale factories on the South East Coast of China. Many factories are family workshops with 5-10 workers. They usually copy the design and the function of branded cell phones with much lower price. In fact, ShanZhai phones have been around for a quite a few years, but it started to gain huge attention and market shares from 2008. The Shanzhai phone also refers to the phones that are partial knock-off or any phone with ingenious design but not from brand-name companies. The word "Shanzhai" now even extended to other fields of people's life and become part of the grass-root culture. It's also been reported that Shanzhai phones have been spotted in some developing countries and we expect to see more reports of Shanzhai phones outside Chinese market. Recently Indian government and service providers banned the Shanzhai phones from the network because some Shanzhai phones may not have unique IMEI number. But shortly the ban was lifted due to the strong demand from consumers.

Researchers in China are also trying to understand this phenomenon to help the government to cope with possible consequences. In the largest electronic market in Shenzhen, China, I personally tested at least five iPhone clones priced from $80 to $120. Some iPhone knock-offs (see Figure 9) I tested are almost identical to real iPhone outside but with more features inside. It's said that those shops could crank out 1:1 knock-off of any new release from any brand name manufacture in two weeks if they can get hold of a sample phone. Knock-offs are not the only product those manufactures are producing. They also make some ingeniously designed phones that no brand name cell phone manufactures have ever made. The IP issues and ethic issues are beyond this study. We are trying to understand why those Shanzhai manufacturers can move so fast on technology innovation while the multinational manufactures' cannot.

There are some explanations for the extremely low cost of ShanZhai (Bandit) phones:

1. The integrated all-in-one cell phone chip from Mediatech; Little investment on R&D. To be fair, there are certain R&Ds on the design and manufacturing process, but not on technology. The cost of components is the major cost of the phone, then the cost of assembly. An experienced Shanzhai phone workshop can come up with a new design fair quickly with little effort, so the design cost is also fairly low compared to brand name manufactures.

2. The capability to source low cost components. For example Shenzhen is one of the manufacturing and distribution centers of Shanzhai phones. All the components, such as screens, keyboards, IC chips are widely available with 100 miles of the city, and very cheap.

3. Capability to tap into the existing supply chain. They are able to utilize the same supply chain established by big companies without previous investment.

4. Extremely low manufacturing cost. The labor cost is extremely low. Not much upfront investment.

5. Evasion of tax. Those small workshops usually don't pay the tax. Compared to the legitimate companies, they have cost advantage

6. No testing, no regulatory compliance. It's been known that most Shanzhai phones do not comply with radiation safety standard. So customer is at the Shanzhai phone manufactures mercy in term of radiation safety.

Despite the popularity and ongoing research on this phenomenon in China, little is known how those small manufacturers could design and manufacture a new phone in such short time frame, regardless whether their phones are knock-offs or original designs. Talking with shop owners revealed that those manufacturers may employ a process similar to open innovation model (but not open source) in their design and manufacturing. Most manufacturers are in the vicinity of Shenzhen. There exists a community very similar to the open source software communities for the Shanzhai phone manufacturers even though there's no official organization. They do share information, not voluntarily but by the "force of nature". Unlike the open source software communities, their knowledge is not documented anywhere but spread from one shop to another shop by the word of mouth.

There's really nothing the individual owner can hide. There's no concept of IP or IP protection. If someone figure out how to source a cheap component, that knowledge can easily be replicated into other factories even the originator doesn't like to share.
That's just the nature of that community. It's said that most Shanzhai phones use Linux operating system (not confirmed). The source codes or libraries are not really shared but copied from one owner to another owner thanks to the lack of IP protection. They don't need to write the application or the operating system from the scratch. They only need to customize the UI of existing applications to look identical to the original devices. If we think the whole community as one company, and the Shanzhai phones are the product portfolio of the whole company, then it's like thousands of engineers are working together to produce cell phones at the lowest cost and fastest turnaround time without anyone to coordinate them. Actually you have to think those thousands engineering are competitors among themselves. Isn't that amazing?

So for the multinational cell phone manufactures, they do have some lesson to learn from the Shanzhai cell phone manufacturers' community. If anyone of them masters a fraction of the innovation speed of the Shanzhai phone manufacturers, then landscape of mobile industry will be changed dramatically.

5.5. Prediction of Future Mobile Handset Market

The most recent data show that 20 percent of US households had only mobiles during the last half of 2008 while only 17 % had landline only. Six in 10 U.S households have both landline and cell phones, while one in 50 has no phones at all (FRAM 2009). The trend will continue at faster speed in the near future. What does that information tell the mobile industry? While the mobile penetration rate is almost 100 percent in United States, mobile service providers should still see this as a positive sign to increase their revenue. When people dump the landline, they have more budgets to spend on mobile service, presumably non voice service.

Though mobile service providers will find it hard to increase the account numbers, they should be able to find room to increase ARPU (Average revenue per user)

5.5.1. Smartphones Set to Surge

Mobile handset manufactures should also see previous news as a positive sign. With the increased spending on non-voice related service, the demand for smartphone will also increase. Industry observers Juniper Research already claimed that worldwide sales of high-end mobile phones should double to 300 million annually by 2013 thanks to rising demand for Web 2.0 apps on the go. The analyst believes 23% of all new mobile phones will be smartphones by 2013. For US market, Gartner estimate that More than 35 million smart phones are expected to sell in the U.S alone this year, up 77% from a year ago, in a market worth $11.8 billion. By 2012, the market will be worth $29.2 billion, with nearly 100 million units sold.

Some fear the popularity of smart phones will push down the price of higher-end feature phones offered by the large handset makers. The consequence is a challenge to maintaining or boosting margins (Cheng 2008). For this reason, smartphone market should be an interesting battle ground to watch in the next couple of years. This is also the battle field for open source mobile operating system as most smartphones are on open source OS now.

5.5.2. The Decline of the Mid-Market Cell Phone

According to ABI Research, even as the overall mobile phone market surges, sales of mid-market cell phones are expected to crash in the next five years. ABI anticipates that just 441 million such phones will be shipped in 2013, down from 854 million in 2007. Today mid-level phones represented 74% of the market, while low-end phones represented 16% and smartphones 10%. In 2013, ABI expects to see a 46%, 23%, 31% market share for low-end, mid-level and smartphones. The dumb (phone) become dumber, the smart (phone) become smarter, and leaves the middle one nowhere. The smartphones using open source operating systems will have cost advantage over the phones with proprietary system. The decrease of smartphone price also pushes hard on the mid-market phones.

5.5.3. PC Maker entering Mobile Phone Space

As the cell phone is shifting to commodity, the traditional mobile device companies are not the only players in this market. The new trend in the industry is that PC makers are entering the mobile device market either by supplying smartphone or via their 3G-enabled netbook devices. Apple shipped its first smartphone iPhone in the middle of 2007. iPhone is highly regarded by many consumers.

Technically speaking, smartphone is not that different from the PDAs that Dell used to make (Dell exited its PDA business in 2007). Other PC makers follow the suite. The popularity of netbook and its 3G-capable feature also make the line between a portable computer and cell phone blurred.

5.5.4. Industry Calls for Collaboration

Several industry experts already stood out calling for collaboration and open standards. Calls for co-operation from the biggest player in this market probably carry more weight. Speaking at the Mobile World Congress (MWC) conference in Barcelona this February, Nokia CEO Olli-Pekka Kallasvuo told delegates: "The next phase of our industry will require some courage because a change always requires a shift away from what we are comfortable with, what we are used to."

He also added "We will have to work together with competitors, new players and partners in different ways far more than in the past. By sharing resources and ideas, by tapping into each other's expertise we can accelerate our efforts and transform our products and services by leaps and bounds. Mutually beneficial relationships among industries and companies will be more important than ever."

5.6. Discussion and Recommendations

The mobile industry is undergoing fast-paced changes, probably faster than any industry existed before in the history. Motorola already gave away its leadership in only a couple of years though it is still the strongest player in US market. Motorola's loss in cell phone sector widens as its market shares keep dropping. Nokia remains the worldwide leader in mobile handset industry and is confident of maintaining its competitive advantage in technology, innovation, productivity for the foreseeable future. However, history has shown that the status quo does not prevail forever. How to survive in this extremely competitive business environment is the question every mobile handset company wants to understand. Open innovation is certainly one of the answer, but not the only answer, or absolutely the right answer for every company.

5.6.1. Criteria of What to Open, What to Close

From closeness to openness, the trend is clear and inevitable in the mobile industry. For the mobile device manufacturers, if they can't change the trend, then they have to learn to adapt to it. Their challenge is how to decide what area to open, what area to remain proprietary. There are a few criteria that firms can use to make their decision. Some of them have already been discussed in various places in this study.

1. Difficulty of information diffusion: For the technologies that are expected to diffuse rapidly, companies should open them as early as possible to benefit everyone in the field. Holding them in the darkness does no good to anyone. For the technologies that are not expected to diffuse easily, keep them in private domain.

2. Global impact: open the technologies that are not the core business to stimulate demand for other products. Like the shaver and razor business model. i.e. sell the complement product.

3. Initial investment and long term gain/investment ratio: If the initial invest is too high, better to open the technology and make joint development effort. IBM calculated that it costs approximately $500 million annually to maintain an industrial-strength operating system. If IBM invests $100 million of its man power to support Linux development, it will get same quality product with $400 million savings. Other big firms did similar calculation. That's exactly the reason why the big firms are the major contributors to theLinux community today.

4. Breakdown the system or business to components and categorize them. For the commodity category, open the system and license the technology. For the rapidly evolving components such as operating system and applications, better join the open source development. For the components that are hard to imitate, keep them in the house. Change the business model to sell those components instead of the whole system.

5.6.2. A Model Based on Stage of Current Technology

Chesbrough recommended an anti-piracy strategy to software industry based on the stage of the technology life cycle. Similarly mobile industry can adopt similar strategy to decide what to open, what to close.

The life cycle of a technology has four stages, namely the introductory stage, growth stage, maturity stage and decline stage. At the introductory stage, the technology diffusion is very slow. Customers are not willing to try something new unless there are significant benefits. Many wonderful technologies died during this stage before they could even reach the growth stage. In the growth stage the adoption of the new technology grow exponentially until the market is saturated. Then new technology will emerge to replace the old technology.

If the technology in a mobile handset company's portfolio is in the introductory stage and growth stage, company should open the technology. It pays to be open in the very early phase of a new technology to get the information out to help the diffusion. Besides no one really know the best business model for the new technology anyway. During growth stage, company should also adopt the open strategy to drive the growth so the technology can become dominant design of the industry. If a technology is in the maturity stage, company may consider keeping it closed to reap the existing value. Company should also consider licensing the technology to other firms to maximize the value before the technology become obsolete. If the technology is in the last phase, the decline stage, company should keep it close, and aggressively license and enforce the IP.

Company can also consider exiting that technology and only collect the loyalty. For example Lucent is still collecting tons of money from the technologies it created many years ago but no longer pursued.

5.6.3. From Ubiquity to Invisibility

The stages of technology diffusion are critical mass (ownership by 20-30% of the population), ubiquity (30-70%), and finally invisibility (more than 70%). In the last stage, so many people have the technology that it's taken for granted (Brown 2009). "The step after ubiquity is invisibility" was first said by Al Mandel who held high-level positions at Apple and AOL. He pointed out that once a technology had reached the point where everyone had it, people simply forgot about it and from then on assumed it would be there. Clearly PC already reached the "invisibility" stage while mobile devices is still in the "ubiquity" stage. The two industries share so many similarities that what happed before in PC industry will most likely happen again in mobile industry. What we can expect is the "invisibility" stage in the mobile industry.

The cell phone industry is experiencing the same device size change as what happened before in PC Industry. Similarly the transition from proprietary to standard system will also happen in mobile industry.

5.6.4. Change Business Model

Brand Management in the Open Source Model

In the very early days when Red Hat tried to figure out a business model, Robert Young looked at the commodity industry, from bottled water, soap to ketchup and learned some ideas (Young 1999). The brand name behind those commodities stands for quality, consistence and reliability.

So they realized the value in brand management of those commodities that they could emulate. Everyone can make ketchup in their kitchen but no consumer does that because it's cheaper and more convenient to buy ketchup from Heinz or Hunts. Heinz has 80% of the market because they have been able to define the taste of the ketchup in the mind of ketchup consumers. Coca Cola and Evian are able to sell billions of dollars of tap water in the US market because they address the irrational fear that water coming from our tap is not to be trusted. Now the cell phone business is increasingly becoming commodity business, any company can use the free open source OS in its product. What will really differentiate in the market is the company that can define the product in the mind of consumer and can do the brand management well. Those companies will take the market eventually.

Lego Business Model

Every kids grow up in US knows the Lego brand name. They have used Lego kits to demonstrate their creativity. They build houses, machines, animals and anything you can name using many small plastic pieces from that company. Kids even build sophisticated robots for robot competition using Lego kits. Lego does not build any final product, but it supplies the building material. It's up to the user to exhibit his/her creativity using their products.

Mobile industry just started disintegration. If the industry has to go through the process of disintegration, standardization and specialization, then think about the Lego business model. It's not too bad to be Lego. The catch is, probably only a few "Lego" can co-exist.

5.6.5. Disruptive Technology

No one can really predict the upcoming disruptive technology, but company should be prepared to change when the disruptive technologies emerge on the horizon. As Charles Darwin said, "It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change." This statement is especially true for high-tech industry and has been proven by the recent history of this field. For example, IBM transformed itself from a computer vendor to a service provider as of today and survived the harsh environment.

On the other hand, most companies which competed with IBM in the computer hardware business no longer exist. It's not always easy to spot the disruptive technologies, but companies can always bet on a group of technologies to minimize the risk.

6. Future Research Proposals

Although the area of open source business dynamics has been studied widely across the academia, this study provides a good basis for future research.

In current literature, there is lack of research in which the concept of dynamic capability is studied thoroughly in the context of OSS business dynamics. The formation of such dynamic capabilities possesses particularly interesting research topics.

Agile software development methodologies have some characteristics in common with OSS development, for example, small teams and a team-driven approach. Current studies and papers don't get into further detailed analysis or research in which agile software development methodology is taken to the OSS context.

6.1. Conclusions

The domain of embedded systems is an emerging business area in which OSS may be used successfully. The growing smartphone market, in particular, is a sector in which several manufacturers successfully use embedded Linux, generates revenue flows, thus providing possibilities for the whole ecosystem of embedded systems.

Open Source is not a Business Case

On its own, OSS does not generate any direct revenue flows. In fact, OSS in itself has no business value and, quite the opposite; it has a neutralising effect on the markets, as it is also available to competitors. OSS acts as an enabler, however, constituting various ways for revenues for many companies in the domain of embedded systems. These companies generate revenues by customising OSS components or combining different elements, including OSS, proprietary software, hardware, services, etc., but, why does one company succeed while others struggle in their attempts to use OSS?

It is proposed that companies that possess adequate dynamic capabilities, in essence those needed to differentiate and strategise within OSS, are more likely to succeed. It is also suggested that it may be more beneficial to contribute, like in the engagement of two-way conversation with the OSS world, rather than merely exploiting it.

Strategising at business level

The likelihood of an organisation succeeding with open source is dependent on its ability to formulate and align certain key resources and activities to support community driven software development. Thus, in order to succeed, companies need to acknowledge that OSS awareness should be spread through all levels of an organisation, including departments such as R&D, sales, marketing, human resources, administration and legal.

Open Source Strategy Assessment

As an answer to the question of how a company can assess the feasibility of open source business strategies, it is suggested that the company should engage in a workshop in which a thorough assessment of open-source-related challenges, resources and dynamic capabilities is executed. The participants in such a workshop should be chosen carefully so that all the key viewpoints of OSS development in the company are involved. That alone is not enough; however, such a workshop should also include outsiders, preferably people representing areas that are not covered by the company.

In order to implement their open source business strategies successfully, companies should assess their current business models, resources, capabilities and dynamic capabilities carefully in regard to open source. They also need to be aware of the various challenges associated with OSS development.

References

Book and Article References

- Amit, R., & Shoemaker, P. J. H. (1993).
  Strategic assets and organizational rent.
  Strategic Management Journal, 14(1), 33-46

- Barr, M., & Massa, A. (2006).

  Programming embedded systems: With C and GNU development tools.
  O'Reilly Media, Inc.

- Beard, D. W., & Dess, G. G. (1981).
  Corporate-level strategy, business-level strategy, and firm performance.
  The Academy of Management Journal, 24(4), 663-668.

- Berger, A. (2002).

  Embedded systems design: An introduction to processes, tools and techniques. Kansas: CMP Books.

- Chesbrough, H. (2006a).
  Open business models: How to thrive in the new innovation landscape.
  Boston, MA: Harvard Business School Press.

- Chesbrough, H., & Rosenbloom, R. S. (2002).
  The role of the business model in capturing value from innovation: Evidence from Xerox Corporation's technology spin-off companies.
  Industrial and Corporate Change, 11(3), 529.

- Crowston, K., & Scozzi, B. (2002).
  Open source software projects as virtual organizations: Competency rallying for software development.
  IEE Proceedings Software, 149(1), 3-17.

- Cusumano, M. A. (2004).
  The business of software.
  New York, NY: Free Press.

- Eisenhardt, K. (1989).
  Building theories from case study research.
  The Academy of Management Review, 14(4), 532.

- Fitzgerald, B. (2006).

  The transformation of open source software.
  MIS Quarterly, 30(3), 587-598.

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

- Hofer, C., & Schendel, D. (1978).
  Strategy formulation: Analytical concepts.
  West Publishing Co.

- Katsamakas, E., & Georgantzas, N. (2007).

  Why most open source development projects do not succeed?
  In FLOSS'07: Proceedings of the first international workshop on emerging trends in FLOSS research and development (p. 3).
  Washington, DC: IEEE Computer Society.

- Krishnamurthy, S. (2005).
  An analysis of open-source business models
  Cambridge, MA: MIT Press.

- Lerner, J., & Tirole, J. (2002).

  Some simple economics of open source.
  Journal of Industrial Economics, 52, 197-234.

- Magretta, J. (2002).

  Why business models matter
  Harvard Business Review, 80, 86-92.

- Mayo, M. C., & Brown, G. S. (1999).
  Building a competitive business model. Ivey
  Business Journal, 63(3), 18-23.

- Morris, M., Schindehutte, M., & Allen, J. (2003).
  The entrepreneur's business model: Toward a unified perspective.
  Journal of Business Research, 58, 726-735.

- Müllerburg, M. (1999).

  Software intensive embedded systems.
  Information and Software Technology, 41(14), 979-984.

- Osterwalder, A., & Pigneur, Y. (2009).

  Business model generation.
  Amsterdam, Netherlands: Self-published.

- Porter, M. E. (2001).
  Strategy and the Internet.
  Harvard Business Review, 79 (3), 63-78.

- Riehle, D. (2007).

  The economic motivation of open source: Stakeholder perspectives.

IEEE Computer, 40(4), 25-32

- Rosen, L. (2005).

  Open source licensing: Software freedom and intellectual property law.
  Upper Saddle River, USA: Prentice Hall

- Rossi, M. A. (2006).

  Decoding the open source puzzle: A survey of theoretical and empirical contributions.
  The Economics of Open Source Software Development. Amsterdam: Elsevier.

- Seddon, P. B., & Lewis, G. P. (2003).
  Strategy and business models: What's the difference.
  Proceedings from the 7th Pacific Asia Conference on Information Systems, Adelaide, South Australia.

- Slywotzky, A. J. (1996).
  Value migration.
  Boston: Harvard Business Review Press.

- Stewart, D., & Zhao, Q. (2000).
  Internet marketing, business models, and public policy.
  Journal of Public Policy & Marketing, 19(2), 287-296.

- St. Laurent, A. (2004).

  Understanding open source and free software licensing.
  Sebastopol, CA: O'Reilly Media, Inc.

- Teece, D. J. (2007).
  Explicating dynamic capabilities: the nature and micro foundations of (sustainable) enterprise performance.
  Strategic Management Journal 28, 1319- 1350

- Väyrynen, K. (2009).
  Evolution of software business in industrial companies: resources, capabilities and strategy.
  Doctoral thesis, University of Oulu, Department of Information Processing Science.

- Ven, K., & Mannaert, H. (2008).

  Challenges and strategies in the use of open sources software by independent software vendors.
  Information and Software Technology, 50, 991-1002.

- von Hippel, E., & von Krogh, G. (2003).

  Open source software and the "private- collective" innovation model: Issues for organization science.
  Organization Science, 14(2), 208-223.

- West, J. & O' Mahony, S. (2005).

  Contrasting community building in sponsored and community founded open source projects, System Sciences, 2005.
  Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS '05), 196c-196c.

- Winter, S. G. (2003).
  Understanding dynamic capabilities.
  Strategic Management Journal, 24(10), 991-995.

Online References

- Embedded System Definition. (2006).

  The Linux information project.
  Retrieved from: http://www.linfo.org/embedded_system.html

- Koenig, J. (2004).
  Seven open source business strategies for competitive advantage.
  IT Manager's Journal.
  Retrieved from:
  http://management.itmanagersjournal.com/ article.pl?
  sid=04/05/10/2052216&tid=85&tid=4)

- LinuxDevices.com. (2008).
  Linux still top embedded OS.
  Retrieved from: http://www.linuxfordevices.com/c/a/News/Linux-still-top-embedded-OS/

- Osterwalder, A. (2008b).

  Business model alchemist.
  Retrieved from:
  http://www.businessmodelalchemist.com/2008/07/what-is-business-model-2.html

- Stähler, P. (2002).
  Business models as an unit of analysis for strategizing.
  Retrieved from:
  http://www.staehler.info/english/definitions.htm

- Ubuntu (2013)

  Ubuntu on devices
  Retrieved from: http://www.ubuntu.com/devices/phone