Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

Johannes Christian Neupert
MASTER THESIS

Getting Licensing Right

Submitted on February 22nd, 2016
Matriculation Number: 21725360

Supervisor: Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Nürnberg, February 22nd, 2016

License

This work is licensed under the Creative Commons Attribution 4. 0 International license (CC BY 4.0), see https://creativecommons.org/licenses/by/4.0/

Nürnberg, February 22nd, 2016

Abstract

The success of every software project is determined to no small degree by the license under which its creator chooses to publish the software. Free/libre open source software has made free sharing of software and code reuse much easier but has also made the business of software licensing far more complex. This thesis presents a teaching case seeking to introduce the basic concepts required to make licensing choices successfully. The case casts a special focus on free/libre open source licenses, revealing the interwovenness of licensing strategy and the business model of an open source-based company. The 2011 founded startup OwnCloud is a software vendor offering an on-premise file synchronization and sharing solution. The OwnCloud suite is free/libre open source software, but there is also a proprietary edition available aiming to meet the demands of large enterprises. The students have to put themselves in the shoes of one of OwnCloud's co-founders who is in charge of the license management as he faces a key licensing decision regarding one of the central components of OwnCloud's software suite.

Key words: *intellectual property, software licensing, free/libre open source software, single-vendor commercial open source business model, dual licensing strategy*

Contents

I List of Tables....v
II List of Figures....v
1 Introduction....1
2 Teaching Case....2
  2.1 Opening Paragraph....2
  2.2 An Introduction to OwnCloud....2
    2.2.1 OwnCloud — Community Project and Company....2
    2.2.2 The Business Model of OwnCloud Inc.....4
    2.2.3 Why Open Source?....6
  2.3 Software Licensing....8
    2.3.1 Intellectual Property....8
    2.3.2 Free/Libre Open Source Software Licensing....9
    2.3.3 License Compatibility....13
    2.3.4 Dual and multiple Licensing....13
  2.4 Complexity of Licensing at OwnCloud....14
    2.4.1 Overview of the Licenses used at OwnCloud....14
    2.4.2 Overview of the Code Components of OwnCloud....14
    2.4.3 Why the OwnCloud Community Project depends on Licensing....16
    2.4.4 Why the OwnCloud Software Development depends on Licensing....16
    2.4.5 Why the Business Model of OwnCloud Inc. depends on Licensing....17
  2.5 The IOS Client Licensing Issue....19
    2.5.1 The OwnCloud Mobile Clients....19
    2.5.2 The IOS Client Licensing Decision....20
  2.6 Exhibit 1: Outline of OwnCloud Business Model....22
  2.7 Exhibit 2: Comparison of OwnCloud Editions....23
  2.8 Exhibit 3: Different Copyleft Effects for different Types of Modification and Combination of Software....24
  2.9 Exhibit 4: Most important Licenses at OwnCloud....25
3 Conceptual Background....26
  3.1 Intellectual Property Rights....26
    3.1.1 Trademarks....26
    3.1.2 Patents....27
    3.1.3 Copyrights....29
  3.2 Software Licenses....32
    3.2.1 Proprietary Software Licenses....33
    3.2.2 Free/Libre Open Source Licenses....34
  3.3 Exhibit CB-1: Factors to bear in mind when out-licensing....38
4 Teaching Note....39
  4.1 Case Title....39
  4.2 Case Synopsis....39
  4.3 Teaching Objectives....39
  4.4 Immediate Issue....40
  4.5 Basic Issues....40
  4.6 Suggested additional Reading....40
  4.7 Suggested Student Assignment....41
  4.8 Case Analysis and Teaching Plan....41
    4.8.1 Comprising Teaching Suggestion....41
    4.8.2 Recap....41
    4.8.3 The IOS Client Licensing Decision....44
5 Case Difficulty Classification....48
III References....vi

I List of Tables

Table 1: Comparison of OwnCloud Editions 23
Table 2: Different Copyleft Effects for different Types of Modification and Combination of Software....24
Table 3: Most important Licenses at OwnCloud....25
Table 4: Factors to bear in mind when out-licensing....38
Table 5: Overview of expected Case Difficulty according to the Case Difficulty Cube Model of Leenders, Maufette-Leenders, and Erskine....48

II List of Figures

Figure 1: The OwnCloud Trademark as recorded with the United States Patent and Trademark Office ("Own-Cloud Logo", n.d.)....8
Figure 2: Overview of the Components of the Server Edition and the attached Licenses....15
Figure 3: Overview of the Components of the Enterprise Edition and the attached Licenses....15
Figure 4: Setup of the OwnCloud Core 17
Figure 5: Setup of the Android Mobile Client....19
Figure 6: Setup of the IOS Mobile ClientFigure 7: Setup of the iOS Mobile Client....19
Figure 7: Outline of OwnCloud Business Model according to the Business Model Canvas Template by Alexander Osterwalder....22
Figure 8: Possible Outcome of Brainstorming Session about OwnCloud....42

1 Introduction

Software licensing is a key issue for single developers, founders of software startups, or the senior management in the corporate software giants. Basically, in fact, it is important for all software projects. Software licenses are legal tools allowing to grant software usage rights and to specify the terms of what users are allowed to do with their obtained copy of the software. Free/libre open source software is made possible by software licenses that explicitly give users the right to share self-made modifications or copies of the software to which the license is attached. But licensing does not only play a role for software publishing. Free/libre open source software provides a huge pool of software that is free to be reused in new software projects. But when incorporating other software in one's own project, the associated licenses always have to be heeded. The code modules come with licenses attached that impose certain requirements for allowing redistribution. Most especially, free/libre open source software licenses added a new layer of complexity to the issue of software licensing.

This thesis provides a teaching case designed to give an introduction to the complex topic of software licensing with an emphasis on FLOSS licenses. The teaching case illustrates how licensing decisions need to be matched to a company's business model. It shows that the licensing choice for even a single software component can be of major significance for a company's business success.

In the course of the teaching case, presented in Chapter Two, the reader is introduced to a real-life licensing management situation at OwnCloud Inc. OwnCloud is a free/libre open source alternative to mainstream services like Dropbox. Users can install and run OwnCloud on their own network. The on-premise cloud computing model, and the freely accessible source code of the solution, make OwnCloud very interesting for all individuals and businesses concerned about data privacy and control. In addition to the free/libre open source offering, large enterprises have the option to obtain a proprietary edition of OwnCloud, which has some extra features matched to their special requirements. In this context, the case shows how software licenses are key to OwnCloud's business success. Certain criteria made OwnCloud's co-founders question one of their original licensing choices for a central software component. The reader slips into the role of the case protagonist who has to reevaluate the licensing choice in order to arrive at a final licensing decision.

Chapter Three provides deeper insight into the conceptual background of intellectual property, software licensing, and the business of free/libre and open source licenses.

Chapter Four contains a proposal, in the form of a teaching note, as to how the case could be discussed in class. It comprises literature recommendations, a case analysis, and a teaching plan.

Chapter Five provides a classification of the case difficulty in order to provide an estimation of how much time course participants need to allocate to master the case.

2 Teaching Case

2.1 Opening Paragraph

On the morning of October 6, 2014, Mr Holger Dyroff, co-founder of OwnCloud Inc., was on his way to an important meeting. While waiting at a red traffic light, he was outlining for himself the strategy he would shortly be presenting to the other two co-founders of OwnCloud. The meeting was with Mr Frank Karlitschek, who had initiated the OwnCloud project and was in charge of the development, and Mr Markus Rex, OwnCloud's chief executive officer (CEO). The aim of the gathering was to come to a final licensing decision regarding the IOS- client, which allows users to access their OwnCloud's file sync and share server instances from all mobile Apple devices. Dyroff's responsibilities at OwnCloud included the management and governance of intellectual property and licensing. All the three co-founders were convinced that these fields were of tremendous relevance for the success of OwnCloud. Up until this point, the IOS mobile client had been exclusively under a proprietary license, even though OwnCloud was based on open source. In order to be able to distribute it via Apple's App Store it was necessary to have the IOS client under a proprietary license, because Apple does not accept free/libre open source licensed programs. Unfortunately, having the IOS mobile client not open sourced meant two things: First, the OwnCloud executives had to discover that the IOS client made significantly fewer technical improvements when compared to OwnCloud's Android client. The Android client was open-sourced and the community was thus able to work with it. Second, critical remarks inside the community about the fact that the crowd was deprived of the code of the IOS client were becoming louder and was gaining resonance. Waiting at the traffic light, Dyroff was weighing up in his mind different courses of action and what the associated risks and chances would be. Should he advocate to give the IOS client an open source license? The undesired effects resulting from having a component not open-sourced were likely to diminish, but would OwnCloud Inc. then be losing revenue?

2.2 An Introduction to OwnCloud

2.2.1 OwnCloud — Community Project and Company

OwnCloud Inc. was routed in *free/libre open source software (FLOSS)* right from the beginning.[1] When the startup was founded in December 2011 it was already able to draw on the developments of the OwnCloud community. By the time the company was started, version 2.0 of OwnCloud had been launched.[2] Up until this point, development of the software project had been driven solely by the associated open source community under the management of

1 FLOSS: The wording free/libre open source software (FLOSS) tries to give equivalent weighting to both *free software* and *open source software*. Within the FLOSS community, a serious debate exists as to which term should be used to describe software in which the source code is made available and users are assigned the rights to change the software and to redistribute it (Williams, 2002). Libre was added to balance the acronym FLOSS in favor of free software. Libre also expresses the fact that free software is to be understood like freedom, rather than like free in the sense of free beer (Stallman, n.d.). FLOSS, open source software, and free software can be understood to refer to the same concept in the majority of cases. In this piece of work, the terms are used quite interchangeably.

2 Please note, that OwnCloud has several meanings: First, it stands for the product OwnCloud, which was available in version 8.2 by the beginning of 2016. Second, OwnCloud is the name of the community project associated with the product. Third an OwnCloud Inc. and an OwnCloud GmbH exist. The GmbH is based in Nuremberg and is a subsidiary of the OwnCloud Inc., which is headquartered in Lexington, Massachusetts.

Frank Karlitschek. It was in January 2010 that Karlitschek had kicked off the OwnCloud project at the K Desktop Environment (KDE) conference. He had been a dedicated FLOSS enthusiast for over 10 years by this time, and had also been involved in several management roles in the KDE community.

Like most successful FLOSS projects, OwnCloud was the result of a need for which no other solution thus far existed. Mr Karlitschek was searching for a way to sync photos and other files to all of his devices and to share them with his friends and family. Also, larger files like videos — which are too big to send via e-mail, due to the attachment size restriction — should be sharable without the need to fall back on a thumb drive.

At that time, various *file synchronization and sharing (FSS)* solutions already existed. One very prominent example was Dropbox a startup launched in 2007 which very quickly evolved to become brand leader, setting a benchmark for all other services which provided similar functionality. Dropbox very intuitively allowed users to store and share data. It also allowed them to sync data between all the user devices, enabling them to keep their content and folders in the same state everywhere. Also, Dropbox came with several other practical features such as automatic syncing of pictures users took with their smartphones.

For Karlitschek, though, using this services was out of the question. He had the wish to conveniently transfer his data, but not at the cost of entrusting giving his data to a third-party service that did not guarantee him certain freedoms (Karlitschek, 2014). The mainstream FSS solutions, including Dropbox, were mostly offered as *Software as a Service (SaaS)* and via the *public cloud* distribution model. This *cloud computing* set-up implies that the software is hosted in data centers controlled by *cloud providers* offering it to different customers at the same time (NIST, 2011). Users can access the provisioned software and online storage on demand, for example via the web browser, like opening a web page. They do not need to install it on their own machine or to provide the necessary computing power by themselves. For example, Google Docs users do not have to install an office suite on their computers anymore, but when using Google Docs all data processed is saved on servers maintained by Google. Customers have no chance to work out where exactly their data is stored or processed. For them, the data is simply somewhere out there in the cloud. As services are provided via a network open for public usage, public cloud architectures are often associated with security concerns.

These security and non-transparency concerns may be an especially acute issue for companies that want to provision an *enterprise file sync and share solution (EFSS)* for their employees. This can become especially critical in combination with concerns about *data privacy*, which are a particularly sensitive issue in Europe, where companies need to take special care that the personal data of their employees does not get stored or processed under laxer privacy measures than the provisions of the *European Data Protection Directive*. Due to regulations such as these, it can be crucial that companies are able to track at which place, or at least in which legislation, their data is processed or stored.

Karlitschek's aim was to provide an intuitive and consumer-friendly FLOSS alternative to Dropbox and other mainstream file sync and share providers — an alternative especially interesting for all those who wanted to maximize control over their data and have full knowledge of where, in which jurisdiction, and for how long their data was stored.

Thanks to open source, OwnCloud's community version quickly became a success story. In 2011, just one year after the launch of the project, OwnCloud estimated it had around 350,000 users worldwide (Endsley, 2011). By September 2015, OwnCloud Server had already reached the 2.5 million user mark ("OwnCloud", 2015).

OwnCloud Inc. was founded to complement the OwnCloud community in December 2011 by Frank Karlitschek, Holger Dyroff, and Markus Rex. The three German co-founders picked the United States — OwnCloud Inc. is headquartered in Lexington, Massachusetts, close to Boston — for incorporating the startup, because new companies there are provided with a better ecosystem for IT startups. For example, they found it easier to raise funding by venture capitalists. Dyroff had a long FLOSS experience in regards to business development and became vice president strategic partners at OwnCloud. He also became executive director of OwnCloud GmbH, the German subsidiary of OwnCloud Inc. Markus Rex, who had a long technology management background at FLOSS organizations became chief executive officer (CEO) of OwnCloud Inc. Frank Karlitschek remained community leader of the OwnCloud community project. In 2012, he additionally stepped into the role as chief technology officer (CTO) to also drive forward the product development at the enterprise.

OwnCloud Inc. was able to continuously grow its customer base. By the end of 2015, it had more than 300 enterprises subscribing as customers. That was equivalent to more than 900,000 users of the Enterprise Edition alone, working with it across 47 different countries. These impressive figures also resulted in financial success. In January 2016, CEO Markus Rex was able to joyfully announce that OwnCloud Inc. had managed to grow over 100 per cent during the course of 2015, bringing OwnCloud the most successful financial year ever. Also, he gave OwnCloud good chances of doubling its revenue again during 2016. OwnCloud was thus aiming for realizing annual revenue of US$16 million in 2016 (Rex, 2016).

OwnCloud had successfully entered a rapidly growing market. The overall worldwide FSS market was forecast, by research and intelligence firm International Data Corporation (IDC), to grow from US$805 million at the beginning of 2014 to a US$2.3 billion market in 2018 (IDC, 2014).

2.2.2 The Business Model of OwnCloud Inc.

OwnCloud Inc. is a *single-vendor commercial open source* software company providing an EFSS solution in return for payment in the form of a subscription model.[3] Additionally, it offers service and support for OwnCloud users.[4] OwnCloud Inc. does not itself host its EFSS solution, so it does not act as a cloud provider.

Exhibit 1 gives a comprehensive overview about OwnCloud's business model according to the Business Model Canvas Template by Alexander Osterwalder.

The OwnCloud software suite is offered in two different editions. The *OwnCloud Server Edition* comprises the gratis FLOSS offering whereas the *OwnCloud Enterprise Edition* is an offer for paying customers in the form of larger enterprises. With the Enterprise Edition, the company specifically aims to win larger organizations as customers in need of a secure EFSS solution that does not come at the cost of loosing control over their data. In order to meet customer demands, OwnCloud Inc. extends the software of the community project by additional features, which are mainly of interest for larger enterprises.

3 The company's strategic setup matches the *single-vendor commercial open source business model* as outlined in the paper by Riehle (2012).

4 Service and support activities are more representative of the classical business model of open source firms (Fitzgerald, 2006). Also, when a company does not fully own the rights to a FLOSS project, it is still possible to create a revenue stream around community open source by offering service and support.

OwnCloud Server is a software suite of client-server software that enables users to establish their own file synchronization and sharing service by hosting it on their own hardware. It can be the ideal solution for individuals, a group of users, or for a small to medium-sized businesses. The OwnCloud Server is developed and maintained via the OwnCloud community under the leadership of Frank Karlitschek.

The latest release of the FLOSS server software package can be downloaded for free at the website of the OwnCloud community to set up an on-premise cloud solution (https://owncloud.org). A network attached computer, server, or network attached storage (NAS) is sufficient for users in order to install and run their own FSS cloud. Organizations can run OwnCloud Server in their own network.

A crucial part of OwnCloud's own aspiration and value proposition is allowing for a high degree of privacy and control regarding user data, whilst establishing universal file access and the breaking down of data silos. OwnCloud manages this by providing FLOSS software for download, so that users can install it upon their own infrastructure. *On-premise* software describes software which is hosted on the users own hardware and where it is thus in their sphere of control. Thanks to its on-premise setup, users of OwnCloud know where their data is stored. In addition, they have more control over what happens with their data compared to the situation if they would use a FSS solution offered by an external cloud provider in form of SaaS. Furthermore, it is possible to mount external storage to OwnCloud. With this functionality other cloud services can also be integrated, like for example Dropbox. OwnCloud's *federated cloud* feature enables the combined usage and management of multiple internal and external cloud computing services. This way, also data stored in different places can be universally accessed via OwnCloud's FSS solution without data transferring data control to an external service.

The software suite also comprises sync clients that enable OwnCloud to be accessed from different devices. The *desktop client* seamlessly integrates with a user's personal computer (PC) and permits very intuitive handling. *Mobile clients* available for the different mobile operation systems are downloadable from the different vendors' app stores. These facilitate file access via smartphones or tablets and allow data to be continuously synchronized on all user devices. By virtue of its cleanly designed look, OwnCloud creates a Dropbox-like user experience.

Due to its modular architecture, OwnCloud Server is extendable via apps that can be downloaded from the *OwnCloud App Store* for free (https://apps.owncloud.com). Developed by members of the community, all apps add extra features to OwnCloud. Hundreds available apps provide functionality that, in part, goes far beyond sole FSS. For example the app Documents allows for collaborative editing and creation of text files.

The Enterprise Edition is basically the same software suite as OwnCloud Server. As the Enterprise Edition targets organizations with approximately 500 users, and a minimum of 50 users, it has to satisfy special customer expectations. Mainly, it is important to offer better integration with the extended system landscape that can be expected in large organizations. For example, the Enterprise Edition offers Microsoft Sharepoint and access to Windows Network drives. Additionally, larger companies need to put special emphasis on data management, and a suitable EFSS solution has to enable special corporate policies. So a high level of control over company data must be provided. For example, OwnCloud *File Firewall* allows to govern file access according to fine grain rules. Therefore the Server Edition gets extended by special Enterprise Applications, developed at OwnCloud Inc.

But it is not only this extended set of features that customers unlock with their subscription. For example, they are also provided with all the clients they need for their employees and can create their own branded versions of them. This allows them to replace the OwnCloud logo with their own. One of the customers of OwnCloud is, for example, the German railway company — Deutsche Bahn (DB). The DB subscribed to the Enterprise Edition and installed OwnCloud on their network in order to offer an on-premise hosted EFSS solution to their employees, named DBBox. The branding of the app with the DB logo completes the feel that DB employees do not use an external service but an exclusively DB tool.

The ability to brand the clients is especially important for one group of potential customers of OwnCloud, the *Original Equipment Manufacturers (OEM)*. OEMs are companies that redistribute another company's product under their own branding. Blaucloud, for example, is a German cloud provider that hosts OwnCloud on its servers in order to offer FSS solutions as SaaS to their own customers (https://www.blaucloud.de). For OEMs like Blaucloud, it can be important to have their own logos on their clients so that their customers are able to recognize their product offerings. To offer this ability exclusively for the Enterprise Edition helps to make it become more interesting than the gratis Server Edition.

OwnCloud Inc. generates income by offering its customers subscriptions. The subscriptions are user-based and involve a subscription fee. Whereas the Enterprise Edition is available on subscription only, OwnCloud Server is in general free to use. If however, smaller organizations need more support than they can obtain gratis via the community online forum, they also have the option to select the Standard Subscription offering service and support for OwnCloud Server.

Exhibit 2 shows a comparison of the two different Editions of OwnCloud and highlights important attributes.

As a company based on an open source community project, the business model of OwnCloud Inc. had, for one thing, to take account of this special circumstance while also leveraging that unique characteristic in order to be successful. Firstly, that had implications for the open source development model and the special relationship with the OwnCloud community. Almost two years prior to foundation of the company, the community had laid the cornerstone by starting to realize the OwnCloud project. And development was continued to be conducted by the community after the foundation of OwnCloud Inc. Secondly, *licensing* was a special concern that had to be considered when engineering OwnCloud's business model.

2.2.3 Why Open Source?

Back in 2011, it was not only due to Karlitschek's FLOSS background that OwnCloud became an open source project. For him, it was the obvious choice in order to create a FSS solution that could be trusted.

All source code of FLOSS is open for anybody to review or to modify. The software is even free to be redistributed. Everyone who has a copy of a FLOSS program can share it with others, with or without modifications. In order to be considered as FLOSS, the source code of the computer programs must be made available. These features set FLOSS apart from so-called *proprietary software,* also known as *closed source software*. Source code is not made available, and modification or distribution of copies is mostly prohibited, in order of maximizing monetary exploitation. Sharing of proprietary software is mostly seen as an act of software piracy due to copyright infringement, unless explicitly permitted by the rights holder.[5]

Open source also describes the special *development model* that underlies the creation of FLOSS. The specific FLOSS attribute — that any interested person can alter the source code where desired — allows for a special form of collaborative and decentralized development. Often, users contributing to the development users self-organize in the form of a *community* around the project.

Karlitschek knew from his previous FLOSS experience that an open source approach would be more beneficial for realizing OwnCloud than the creation of proprietary software.

A first category of benefits Karlitschek wanted to leverage for his project are rooted in the special process of open source software development. As the source code is made available everyone with an interest and a certain basic skill set is able to contribute. Also, open source development is a highly cooperative process. Due to the number of helping hands, the productive discussion with like-minded skilled people, and the possibility to cover different areas of knowledge, the development will be faster than when working with a small set of employees. Open source development does not only speed up the development, it can also result in a more qualitative and error-robust product, making the software especially secure. This can be explained with the effect got known as *Linus's Law*.[6] The greater the number of people dealing with the source code of a program, the more bugs are likely to be detected and resolved and thus the likelihood for security breaches is reduced (Raymond, 2001).

Second, the freedom of redistribution and modification makes it easy to incorporate other FLOSS programs into the architecture, provided that the associated licenses are compatible. With simply reusing code from other FLOSS projects, certain features can be implemented without having to reinvent the wheel. This characteristic of FLOSS can save valuable development time and speed up projects significantly. For example, one of many programs OwnCloud was able to use as a module was the open source web server framework SabreDAV. By incorporating SabreDAV, OwnCloud could easily establish communication according to the WebDAV protocol. WebDAV is an open protocol by which OwnCloud enables file access and synchronization. FLOSS helps to easier implement the usage of open standards and protocols — which are important for OwnCloud in order to make it as compatible with as many different platforms file transfer platforms as possible.

Third, the previously named benefits of open source development accumulate to produce a major additional advantage. The FLOSS character of OwnCloud is key to satisfying the high demands that Karlitschek imposed on the project with regard to privacy and user data control. Basically, being open source adds transparency to the equation, as users have the opportunity to look inside the source code and to understand what the software really is doing. It can thus be excluded that programs have backdoors incorporated that give others access, unauthorized by the user, to the uploaded data (Stallman, 2009). Also, vendor lock-in can not occur in FLOSS, as a missing export feature could be added to enable the transfer o f user data to another service.

5 There are certain forms of proprietary software that also allow for free redistribution of copies. For example, *shareware* is proprietary software which is permitted to be redistributed, but free usage may only be granted for a trial period and users are not free to inspect or modify the source code (Lerner, Tirole, 2003).

6 Eric S. Raymond, the first developer who examined the reasons causing open source development to be especially efficient, named this effect Linus's Law as a homage to Linus Torvalds, who initiated and governs the development project of the Linux-Kernel.

But open source did not only help Karlitschek and the community to develop OwnCloud. It is also a strategic asset for the OwnCloud Inc. A FLOSS version promotes use of a product much more efficiently and cost-effectively than marketing by the company could ever do (Olsen, 2005) . Also, OwnCloud's FLOSS background successfully helps to distinguish OwnCloud Inc. from the roughly 300 competitors in the marketplace. In 2014, the company was classified by market research institute Gartner as a niche player for not coming close to the market share of mainstream EFSS solutions (Arlotta, 2014). Yet, when compared to other FLOSS solutions, OwnCloud clearly is the leading open source FSS alternative to the mainstream services, to which all users who care about their data privacy can switch.

2.3 Software Licensing

2.3.1 Intellectual Property

Licensing within the context of software creation and business is based on the concept of *intellectual property.* Intellectual property defines exclusive ownership of an intangible good like man-made creations and inventions. Authors and inventors are granted exclusive rights in the form of *intellectual property rights (IPR)* by the legal system, enabling them to use and benefit from their creations. The public does not have the same set of rights as the creators of intellectual property. For example, persons other than the creator of a drawing are not allowed to sell copies of the drawing. If others wish to use intellectual property, they will have to request usage rights from the IPR holder. IPR holders have the possibility to grant others some of the rights, otherwise exclusive to them. This permission is issued in the form of *licensing*. Usually, such permission involves payments from the licensee in form of license fees, also known as *royalties*. The three primary types of IPR are Trademarks, Patents and Copyrights.

Trademarks focus on the protection of identifiers that can be used to distinguish products or businesses, like names, logos, slogans, or even tunes, for example, created to be associated with a certain company. OwnCloud is a recorded trademark registered on OwnCloud Inc. That way OwnCloud has secured the right to prohibit unwanted usage of its product's name and symbol, and to prevent users from being misled by software unlawfully branded as OwnCloud.

Figure 1: The OwnCloud Trademark as recorded with the United States Patent and Trademark Office ("OwnCloud Logo", n.d.)

Patents are IPRs used to protect the underlying function of — typically technical — inventions. Patents need to be granted by official patent offices and are given for a limited time only. They represent the strongest form of IPR protection, as they can even serve to prevent usage of products developed completely separately yet which follow the same idea as of an invention recorded as patent. This characteristic enables patent holders to set monopolistic prices, or to earn money by licensing the right of using the patented component to others. A company wishing to use patented software first has to obtain the licenses accorded by the patent holder. In general, software is also protectable via patents, even though patents are only indirectly applicable to software in the European Union. In the United States, software patents are granted if they are filed for the technical conception of an invention. There is a growing trend towards software patents, even though there is an extensive global debate on whether, and to what extent, software patents are beneficial. For example, patents can lead to compatibility issues and also overlap with the copyright protection of software. Software patents also add a further level to the complexity of licensing on the basis of copyrights, as such licensing always has to be audited with regards to possible interference with software patents (Evans, Layne-Farrar, 2004).

Copyrights protect artistic works, like paintings, songs, or writing. Copyright applies by default as soon a creator brings to life a form of individual creative expression in a tangible medium. Basically, owners of copyright have the exclusive right to distribute their creation, be it by copying it or by giving away slightly modified versions. This is mainly designed to ensure that copyright holders are able to make money with their creations, before someone else goes to the market with a copy of it. To be protected by copyright law an author, for example, does not have to request copyright protections for a new book. It is applied automatically, but only for a certain period of time. Unlike patents, copyrights do not protect a complete idea. They only focus on concrete instances as produced by creators. If a songwriter composes a song about the sun, only this piece of music is protected by copyright, not the general idea of the sun itself.

Within the context of IPR, computer software is classified as a piece of literature. Software automatically becomes protected by copyright as long the underlying source code is an original work made by the developer. Thus, no source code exists where the copyright is not initially held by either a person or an entity. All those interested to distributing copies thereof have to request permission from the copyright holder. It is common that developers transfer exclusive usage rights to the company that employs them, or to the organization governing the FLOSS project on which the developer is working. Thus, the organizations also have the right to redistribute or license the developer's work. The license to redistribute software again mostly implies costs in the form of royalties. If people were to distribute unauthorized copies of a software program, they would risk prosecution due to copyright infringement, often involving severe penalties. Copyright holders can set the rules with which others have to comply if they wish to use the software.

All three forms of IPR presented allow creators and businesses to protect their market and to generate revenue sources on the basis of licensing usage and distribution rights. But the necessary precondition for those wishing to set the conditions others have to meet is to hold the associated intellectual property ownership.

2.3.2 Free/Libre Open Source Software Licensing

In the context of FLOSS, patents and, for the most part, copyrights play a key role. FLOSS builds on certain principles, such as the fact that everybody is free to study its source code and to modify and distribute the software. At first glance the FLOSS values seem to be incompatible with the concept of copyrights, as copyright law restricts the right of redistribution exclusively to the copyright holders.

But licensing can resolve this conflict, as licenses allow certain exclusive rights of the copyright holder to be passed on to anyone who wants to redistribute them. The developer or the company that developed the software, however, retains ownership of the copyright. Yet everyone wishing to make use of the assigned freedoms needs to adhere to the licensing terms of the IPR holder precisely. Thus, FLOSS does not conflict with copyrights — in fact it is even based upon them. FLOSS licenses demand that the copyright notice of the original work is always passed on when FLOSS is redistributed. This feature is important for FLOSS projects in order to attract developers because they can rely on being empowered to build their reputation in this way. The intention of developers contributing to FLOSS projects is that everybody should be able to copy their software in order to make it available for everyone else. Also, all interested persons should be able to modify the software, so that anyone can adjust it to meet their own needs. In order to make software freely copiable, modifiable, and distributable, FLOSS developers have to explicitly grant these rights to others. *FLOSS licenses* are the tool to generally permit this. Without these licenses, the rights to make and distribute copies would remain exclusive and FLOSS would thus not be able to exist.

A special document, the *Open Source Definition*, details further certain characteristics software licenses have to meet in order for the associated software to qualify as open source. The document is maintained by the *Open Source Initiative (OSI),* an organization founded in 1998 by open source advocates involved from the very inception. The Open Source Definition explicitly lists the condition of *free redistribution*, stating that FLOSS licenses shall not restrict that software can be passed on. All FLOSS licenses therefore have to ensure that the licensee is allowed to redistribute the software. Also, free distribution does not allow the licensor to demand royalties or other payment in exchange for granting FLOSS licenses ("Open Source Definition, n.d.). This implies that no business model creating revenue, with income from licensing that grants usage or distribution rights for single copies, would be applicable around FLOSS. Mainly, because the company would not be allowed to restrict the redistribution and customers could give away the software for free. After a certain period, nobody would any longer pay the company the fees it demands.

That, again, is a major difference to the proprietary software field, where a typical business model is to generate income by selling the usage right for a single copy of the software. Customers do not purchase the software per se but the license to use their copy together with the medium the software is delivered on, like for example a CD-ROM. Granting of usage licenses is typically done via *End User License Agreements (EULA)* which simultaneously restrict all the freedoms granted by FLOSS licenses. *Proprietary licenses* usually grant no more than the permission for users to apply the single copy of a software they have obtained while restricting any further forms of usage. Proprietary licenses therefore aim to limit usage rights whereas FLOSS licenses aim to entitle users to rights otherwise exclusive to the copyright holder.

Two subclasses of FLOSS licenses exist. It is important to distinguish between permissive and reciprocal licenses.

Permissive licenses impose almost no conditions regarding the distribution of the software to which they are attached. Their primary goal is to signal who holds the copyright to the software and to grant users the typical FLOSS freedoms in order that they can freely use the code, make modifications, and redistribute programs. Typically, permissive licenses exclude, via *warranty disclaimers*, all liabilities for any malfunction or damage caused by the software. Finally, they include *attribution clauses* that impose common conditions on the user with regard to redistribution. These clauses ensure that the sources of the FLOSS components can be identified, and that the original licensing conditions and disclaimers are listed together with its derivatives. Due to the attribution clause, users wishing to redistribute software have to incorporate the original copyright notice and the original licensing text.[7]

The *MIT License*, which once originated at the Massachusetts Institute of Technology (MIT), is the most popular FLOSS license ("Top 20 Open Source Licenses", n.d.). Its is also one of the most earliest FLOSS licenses, designed to facilitate in concise manner, the sharing of software for academic projects (Laurent, 2004).[8]

Also one of the first licenses originated from a university background is the Berkley Software Distribution (BSD) license family.[9] The *BSD-2-Clause License* version is very similar to the MIT License. The *BSD-3-Clause License* version contains an additional term, that next to the required attribution the names of the original copyright holders shall not be used for promotion purposes.[10]

The *Apache License 2.0* is maintained by the Apache Software Foundation for the purposes of having a licensing tool for the Apache projects including the Apache HTTP Server.[11] It focuses on software development via large communities. The Apache License 2.0 includes a special paragraph that also transfers usage rights regarding software patents, if the code open-sourced under the Apache License 2.0 should contain patented idea. Even though bearing the name of the Apache software projects it is also used by many FLOSS projects outside of the Apache world which want to open-source their own software under the same terms.

Reciprocal licenses share the attributes of permissive licenses. They also aim to grant users extensive copyright usage rights, and on the other hand guarantee attribution of the source and its copyright holder, as well as its original license. What is more, reciprocal licenses assure that FLOSS source code will remain open and free in the future. This effect is enabled by an additional attribute of reciprocal FLOSS licenses, that has been named *copyleft*.[12]

Copyleft adds more restrictive requirements to the rights to redistribute software. Basically, all those wishing to pass on software that is attached to a reciprocal license have to use the same license for redistribution. This also applies if developers modify the code in order to add any features or to create interoperability with another program (Jaeger & Metzger, 2011). The outcome of modifications is referred to as *derivative work*, comprising a piece of the original software and some extension. Derivative works need to be put under the same reciprocal FLOSS license as was attached to the original element used.

7 In contrast to reciprocal licenses, stating the licensing conditions of the source is for informatory purposes only and does not restrict users when choosing a license for redistributing copies, for permissive licensed software.

8 The MIT License consists of three short paragraphs only: a copyright notice, a permission notice and a warranty disclaimer. It can be viewed on the OSI website (https://opensource.org/licenses/MIT).

9 The template of the 3-clause license can be found together with a comparison to the 2-clause text at https://opensource.org/licenses/BSD-3-Clause.

10 Even though, being a very popular license the third clause is known to be able to cause incompatibilities with other FLOSS licenses like the GPL family (Laurent, 2004).

11 The Apache License 2.0 is already a larger legal document, and can be found on the website of the Apache Software Foundation (http://www.apache.org/licenses/LICENSE-2.0).

12 Copyleft was developed and named by Richard Stallman when he drafted the first version of the GPL. The name is to indicate that no further restrictions may be applied to copies and modifications made from the original, than imposed by the license used for the original. It also is a wordplay referencing on copyrights ("What is Copyleft", n.d.).

Furthermore, copyleft affects *combined works*. A typical way to combine code components is via linking. Here, code components in form of libraries or plug-ins do not get incorporated but combined via references on the single sources. When using a reciprocal licensed component to create a combined work, there is a high likelihood that it has to be licensed under the same reciprocal FLOSS license. Copyleft is sometimes referred to as being *viral, a*s reciprocal FLOSS licenses seem to spread quite aggressively by also affecting other code components. However, *collective works* allow to bundle reciprocal licensed components with other licensed components and to put the result under a software license free of choice. *C*ollective works are created when different software elements are combined to form a larger software package, like done with the different Linux distributions. The single programs incorporated in a collective work can communicate with each other but are not linked. In this set up, it is for example possible to distribute a software suite that consists out of proprietary licensed as well as reciprocal licensed software components.

Exhibit 3 provides an overview of the different copyleft effects on reciprocal software according to the different possible modification and combination types.

The most popular reciprocal license family evolved around the *General Public License (GPL)*, which also was the first copyleft license.[13] Richard Stallman had originally created the license in order to use it for the software components of the GNU project.[14] Version 2 of the GPL follows the MIT license as second most popular FLOSS license ("Top 20 Open Source Licenses", n.d.). OwnCloud uses Version 2 as well as Version 3 of the GPL. Both versions have in common the fact that no further restrictions than those stated in the GPL may be added to a GPL licensed code in the process of redistribution. With *GPLv2* this can cause many license incompatibilities. *GPLv3* was altered to make it more easily compatible with other licenses. For example, GPLv3 allows for compatibility with the Apache license 2.0. Code under the Apache license 2.0 may be combined with code from the GPLv3. But because of the copyleft element i n the GPL, the resulting derivative or combined work must be published underneath the GPlv3. In addition to version 2, GPLv3 includes a patent clause that passes on rights for possible patents of the licensor if licensees were to need these in order to execute the rights granted to them via the GPLv3 with regard to copyrights.

The *Affero General Public License Version 3 (AGPLv3)* is a license based on the concepts of the GPLv3. It has one additional condition that goes beyond the other GPL licenses. GPLv2 and GPLv3 assume that software has to be distributed in the form of copies — copies, for example, put onto a CD-ROM or available via downloads — obtained in order to be applied by eventual users. The conditions of the regular GPL versions are not triggered until such a case of distribution occurs. For example, no source code has to be made available without redistribution. If, however, software is made accessible as SaaS by cloud providers, no actual copies are distributed. With respect to GPLv2 and Version 3, therefore, the cloud providers are not bound to provide the source code of any potential modifications made by them. The AGPL was designed to close this gap of copyleft by adding a condition for usage of a program via computer networks. Cloud providers offering an AGPL protected software on their server also need to provide the equivalent source code. OwnCloud strategically out-licensed the server core under the AGPLv3 in order to have an additional incentive for a certain user group to rather pay for the Enterprise Edition. OEM s wishing to use the

13 The different versions from the GPL license family can be retrieved from the GNU website (http://www.gnu.org/licenses/).

14 In the GNU Manifesto Richard Stallman provides his vision about the project with the main goal to develop a free software operating system (http://www.gnu.org/gnu/manifesto.html).

OwnCloud Server Edition to set up a FSS service hosted on their network resources are, due to the AGPL, required to open source all modifications they may have made to the OwnCloud Core. They can offer a derivative work based on OwnCloud without open-sourcing their source code only if they use the Enterprise Edition, where the core is under the OwnCloud Commercial License.

The *Lesser General Public License (LGPL)* was designed to provide protection for code libraries. The LGPL provides a weaker form of copyleft protection that applies for derivative works only and not for combined works. As a result programs using LGPL protected libraries can be licensed freely without the obligations from copyleft, just like permissive licenses.

2.3.3 License Compatibility

Code components under different licenses may not always be legally combinable, due to the differing licensing terms. When complying with the terms of one license means not to comply with the terms of the other, the situation can only lead to copyright infringement. The GPL, for example, has proved itself to be a license that is incompatible with many other licenses, even other FLOSS licenses (Lindberg, 2008). This can be largely attributed to the clause that no further conditions are allowed to be added to the redistribution of GPL-licensed code. Incompatibility with the GPL can become easily an issue for derived or combined works. For example, the Apache License 2.0 is incompatible with GPLv2, because the Apache License 2.0 has some arrangements against threats to free redistribution from software patents, which the GPLv2 does not have. It is not possible to create derived or combined works with Apache-licensed code and GPLv2-licensed code. The GPLv3, on the other hand, was complemented with a paragraph containing the same restrictions regarding to software patents. So GPLv3 and Apache License 2.0 can be used together.

Due to copyleft, all derivative and combined works resulting from the combination of reciprocal licensed-code components have to be under a reciprocal license again. This is especially interesting to picture from the point of view of a proprietary software project that wants to incorporate FLOSS components but not loose its ability to publish the result under a proprietary software license. From this view point, reciprocal licenses may not be evaluated as compatible with the proprietary licensed code. The viral character of copyleft can put a risk to the aim of publishing the end result again under a proprietary license. For sure, if the result is a derivative work, and with a high likelihood if it is a combined work. The safest way would be by bundling it to a software package in form of a collective work. This way no viral effect would be likely and compatibility would be assured best. Incorporating LGPL licensed libraries into the proprietary software project would also work without the risk to loose the ability to choose the license freely. The weak copyleft of the LGPL does not effect combined works. Permissive licenses, on the other hand, can be combined with proprietary licenses freely, as even a derivative work can be licensed without a restriction. Exhibit 3 also indicates how combinations with reciprocal licensed FLOSS affect the compatibility.

2.3.4 Dual and multiple Licensing

The issue of software licensing can become rather complex, especially when involving FLOSS. At OwnCloud, licensing is deeply interwoven with the whole company's business model. This fosters complexity but also offers great chances — such as, for example, by allowing for leveraging open source but at the same time charging for licenses.

Dual licensing provides a way to put one software product under two different licenses. For this purpose, dual licensing requires full control regarding the copyright of the software module that is to be put under two licenses (Olson, 2005).

Dual licensing unlocks new business opportunities for organizations governing FLOSS projects, because it allows them to distribute a single software product under a FLOSS and a proprietary license. It is thus key to creating software licensing revenue for businesses based on FLOSS and to enabling single-vendor commercial open source firms (Riehle, 2012).

Where useful, software can also be offered under more than two different licenses. That action is referred to under the term *multiple licensing* (Rosen, 2005).

Giving more than one license to a program is not only a basis for the single-vendor commercial open source business model. Dual licensing can also be important for software development in order to avoid complications with non-matching licenses of single software modules that are to be combined. Developers who want to reuse code parts but are hindered because of their licenses can approach the copyrights owners and ask wether they could put the code underneath an additional license that would allow usage. This is also known as *re-licensing*.

2.4 Complexity of Licensing at OwnCloud

2.4.1 Overview of the Licenses used at OwnCloud

OwnCloud makes strategic use of the different licenses to achieve different goals. Alongside the already introduced FLOSS licenses, two types of proprietary licenses are also important for OwnCloud.

The *OwnCloud Commercial License* is a proprietary license specifically designed to fit the dual licensing strategy of OwnCloud Inc. It is the most important license for the Enterprise Edition. Dyroff and the other co-founders created the commercial license in order to be able to offer a proprietary license to their customers that would nevertheless include as many of the FLOSS-typical freedoms as possible. The commercial license by OwnCloud grants customers the right to examine the underlying source code and to modify it. However, this makes it a non-FLOSS license as it only allows these freedoms to paying customers and restricts redistribution.[15] Customers are not allowed to pass on copies. These license attributes are very beneficial for customers as it enables them to seamlessly integrate the OwnCloud EFSS solution into their systems. In addition, customers do not have to be afraid of accidentally open-sourcing their intellectual property, which might happen if they were to use a FLOSS software instead, via the properties of copyleft. In addition, *OwnCloud EULAs* grant usage rights for the proprietary version of the Android client and of the IOS client.[16]

Exhibit 4 gives a comprising overview of the most important licenses at OwnCloud.

2.4.2 Overview of the Code Components of OwnCloud

OwnCloud's different code components played an important role for the three co-founders when setting up their licensing strategy. Each Edition of OwnCloud basically has three main component types: the core module, apps, and three different clients.

15 The OwnCloud Commercial License can be inspected on the website of OwnCloud Inc. (https://owncloud.com/de/licenses/owncloud-commercial/).

16 OwnCloud's EULA for the Android and the IOS Client can be inspected on the website of OwnCloud Inc. (https://owncloud.com/de/licenses/owncloud-android-application/).

- The OwnCloud Core represents the heart of the software suite and is the server program that the users install into their network in order to host their own FSS service.
- The modular design of OwnCloud allows users to complement the core on demand with extra functionality in the form of apps. For the Server Edition, the apps are offered via the OwnCloud App Store. These apps can be developed by all community members who want to integrate some additional features to the core, and OwnCloud does not necessarily have any control over the copyrights.
- The clients are applications the users can install on their different devices to access the core and to provide synchronization with the core. When using clients users do not have to log into their own FSS server via their web-browser. Furthermore, the clients allow for additional functionality on the device side, such as seamless integration into the filing structure of the operating system.

As part of the dual licensing strategy all these single subcomponents can have more than one license. The license attached to a component can differ according to wether it is used in the Server or in the Enterprise Edition.

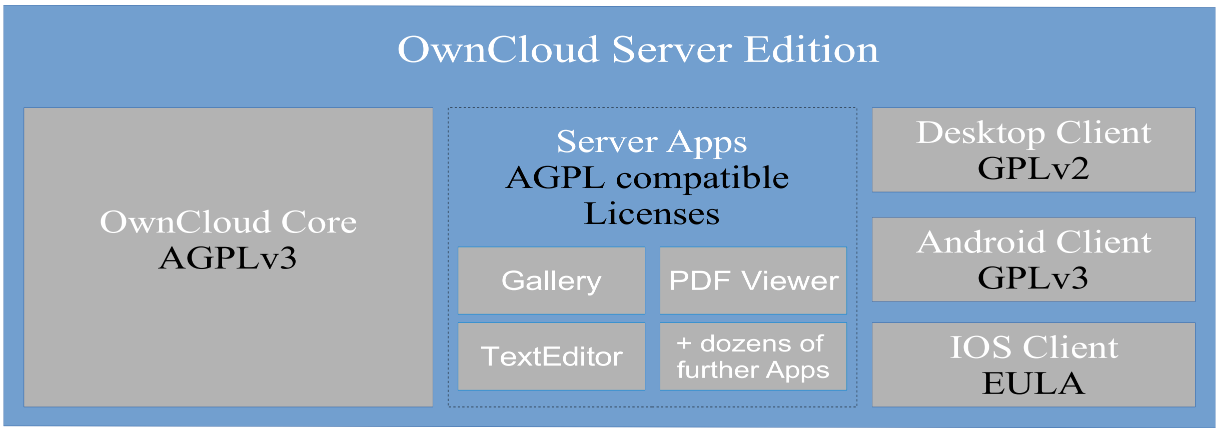

Figure 2: Overview of the Components of the Server Edition and the attached Licenses

The components of the Server Edition — being OwnCloud's FLOSS offering — are largely put under FLOSS licenses. As the AGPLv3 is used for the OwnCloud Core of the Server Edition, the whole Edition is referred to as being licensed under an AGPL. The only component of the Server Edition not under a FLOSS license is the IOS client.

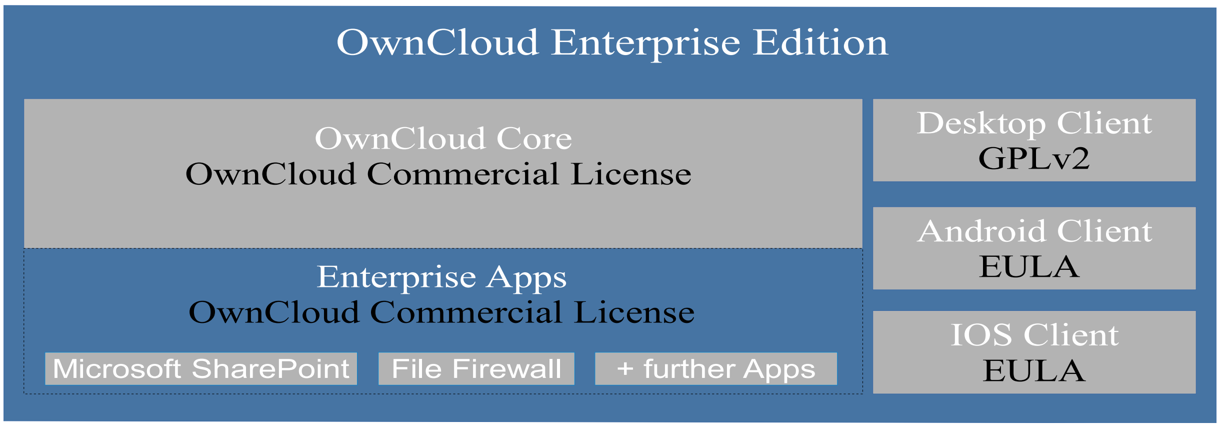

Figure 3: Overview of the Components of the Enterprise Edition and the attached Licenses

As the Enterprise Edition targets potential EFFS customers required to pay fees in order to be able to access it, the components are for the most part under proprietary licenses. However, the technical set-up of the single components does not differ from the Server Edition, with the exception of the apps offered. In order to position the Enterprise Edition on the EFFS market, the Enterprise Apps are especially developed to meet the demands of large enterprises. The Enterprise Apps therefore differ from the Server Apps offering. In addition, the Enterprise apps are distributed together with the core of the Enterprise Edition and do not represent an auxiliary offering, as they do in the Server Edition.

2.4.3 Why the OwnCloud Community Project depends on Licensing

FLOSS licenses are crucial to FLOSS and open source development. Without the explicitly granted rights, OwnCloud could not have been shared freely and no one would have been able to contribute to the OwnCloud community project when it was launched back in 2010. Karlitschek wanted to make sure that OwnCloud would not only start as FLOSS but to remain freely accessible to all interested parties in the future. Due to the copyleft attribute the GPL license family is especially well suited to assuring this.

Karlitschek chose the AGPL for licensing the core because the OwnCloud core is a server software application that users could utilize for offering their own hosted FSS service. Only the AGPL's network reciprocity feature was able to assure that, within this possible setting, the source code of possible derivatives would stay open. In addition, AGPL and GPL sent a strong signal that managed to attract likeminded people wishing to contribute to a durable FLOSS alternative to mainstream FSS services. For Karlitschek, it was also important that OwnCloud would be compatible with the KDE solutions and the overall Linux family.

2.4.4 Why the OwnCloud Software Development depends on Licensing

Just like mot other software projects, OwnCloud reuses code written by others. This is supported by the common software design technique of modular programming. Software is designed on a modular basis, with the functionality being partitioned into different modules. This makes code parts more easily replaceable and facilitates the process of creating new software by recombining the single components. Due in particular to the free redistribution condition many FLOSS components are available for reuse. Yet developers have to respect the terms of the different licenses attached to the single programs. When the different modules are combined, the compatibility of the single licenses also has to be borne in mind.

OwnCloud leverages the benefits of reusing FLOSS components. Some programs are even offered solely to work as modules in different programs in order to enrich them with a certain functionality. SabreDAV, for example, is such a module that allows OwnCloud to use the open WebDAV protocol for file access. A particularly efficient way to reuse code is offered via libraries. Using libraries means that pre-written code does not have to be duplicated and physically incorporated into different parts of a program in order to make it work. Instead, the program can simply call the library to retrieve its functionality whenever this is needed. Thus the code of the library can be provided next to the program instead of being actually incorporated. OwnCloud uses many different libraries as a basis for its different components.

Whenever OwnCloud falls back on code created by others, the licenses attached to the single modules and libraries have to be evaluated. The situation when developers evaluate and accept the licensing terms of desired code modules can be referred to as the *in-licensing* of software (Rosen, 2005). Modules that inherited in this way are referred to as *third party software.* For example, when OwnCloud uses a LGPL licensed library developed by another

FLOSS project can be addressed as third party FLOSS. When in-licensing, developers should bear in mind the possible consequences that acceptance of the licensing terms will have on the further development and — most especially — the distribution of their software.

2.4.5 Why the Business Model of OwnCloud Inc. depends on Licensing

Leveraging its single-vendor commercial open source business model, OwnCloud Inc. creates revenue not only with service and support. One important revenue source is the subscription fees paid by the businesses the firm has been able to secure as customers of the Enterprise Edition. Enterprise customers pay subscription fees for being allowed to use the Enterprise Edition because that is the only way to gain access to additional value in the form of extra functionality, brandable clients, and the avoidance of licensing issues.

In order to realize the planned business model, the co-founders of OwnCloud had to establish a matching solution with regard to licensing and the dual licensing strategy. On the one hand OwnCloud Server was kept under the AGPL license and, as such, offered all the FLOSS-typical freedoms. On the other hand, the company licensed OwnCloud Server under their own created proprietary license, allowing OwnCloud Inc. to add exclusive extra features and to offer the package as Enterprise Edition via the OwnCloud Commercial License.

The complex dual licensing strategy shows that the co-founders had to put an especially large amount of thought into the process of choosing the best license for their own software — a process also known as *out-licensing* (Rosen, 2005). The choice of course critically depends on the degree of freedom left by the terms of in-licensed software. Most especially, if a reciprocal FLOSS licensed module was modified, almost no freedom of choice remains. In principle, the licensor first needs to meet the obligations imposed by the licenses of reused third party components before making any own plans. Another major factor is the degree to which the owner intends to share IPR — OwnCloud's decision to establish a proprietary software branch by using the dual licensing strategy shows this to be a major factor.

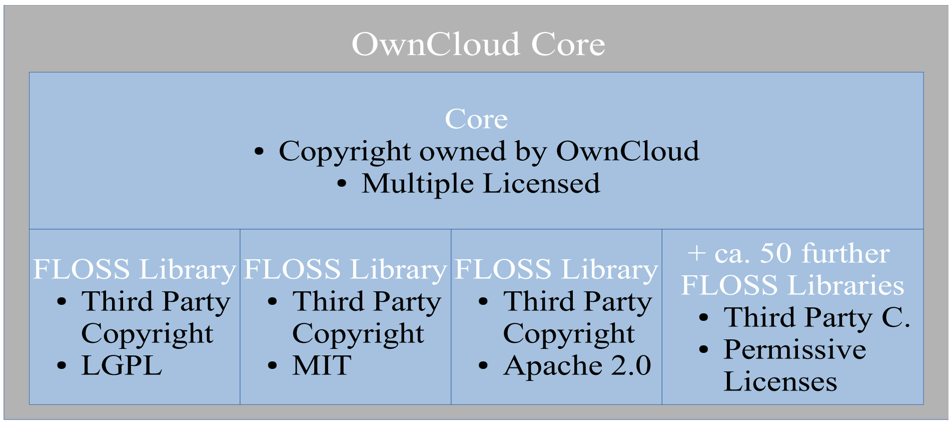

Figure 4: Setup of the OwnCloud Core

Also for the Core OwnCloud in-licenses several components and applies the dual licensing strategy for out-licensing. Figure 4 provides insight into the next deeper layer of the Core. On this level it consists of the core program developed by OwnCloud that works with different code libraries, all providing certain functionalities needed by the core. The code libraries used represents third party open source code for which OwnCloud does not hold the copyright.

The reciprocal AGPL was selected to make the software available to open source users. This is important in order to incentivize OwnCloud Server users to change to the Enterprise Edition, which requires a subscription fee. But customers who plan to make modifications to

OwnCloud, or to incorporate it into their existing software, may prefer to pay money for the Enterprise Edition. This is due to the fact that, when using the gratis Server Edition, the strong copyleft requirement of the AGPL and GPLv2 may imply that customers would have to publish their code under the same reciprocal license. This means they would need to open source code they prefer to keep closed. When using the Enterprise Edition instead, customers are protected — by OwnCloud's Commercial License — from the viral-effect of reciprocal FLOSS licenses, but also have the freedom to make modifications or additions.

As the example of the OwnCloud Core shows, different priorities apply with regards to in-licensing. OwnCloud prefers to use permissive licenses for in-licensing components, as they do not have to fear restrictions on their dual licensing strategy imposed by these licenses. The only reciprocal license OwnCloud uses for in-licensing is the LGPL, as linking LGPL-licensed libraries does not prevent the dual licensing strategy, as long as the libraries get not modified by OwnCloud.

However, permissive licenses or the LGPL do not meet OwnCloud's requirements for out-licensing. For the Server Edition they offer an insufficient protection, as the code they are attached can easily be incorporated into proprietary third party software. On the one hand permissive licenses do not guarantee that the Server Edition remains opens source. On the other hand, they allow competitors to reuse the innovation developed for OwnCloud without any conditions being imposed. So when out-licensing, only reciprocal FLOSS licenses can be used strategically as a barrier, the result being that larger enterprises prefer to avoid any risks by subscribing to the Enterprise Edition.

The AGPL choice specifically focuses on potential OEM users of the Server Edition. If the Core of the Server Edition were to be offered under a GPL version, OEMs could exploit the gap in the copyleft protection by not directly distributing the software and offering it as SaaS. But the network reciprocity feature of the AGPL assures that also OEMs also have to open-source the modifications the might have made. Also, the AGPL is an incentive for this special customer group to switch to the Enterprise Edition instead. Here, the Commercial License would allow users to make all necessary changes but would not require them to open source the respective code.

The co-founders had to tackle a final complexity in order to be able to out-license the single components of the FSS software suite under the dual licensing strategy. Licensors need to have ownership of the copyright of components whose rights they wish to grant to others, or at least the allowance to redistribute the software. Companies need to establish this control for their product in two ways. First, they have to make sure that they have the right to use and distribute all code contributions from all developers involved. This can also have an impact on open source development. Usually, all developers planning to commit code to the project are required to license usage rights regarding copyrights and patents to their employers or organizations governing the project. OwnCloud Inc. covers this *contributor agreements* which have to be signed by every developer who wants to contribute to the development of the core or clients.[17] Second, companies need to make sure that they have permission to redistribute and license possible third party software components reused for the software. This is already to be borne in mind when in-licensing components.

Because of the enormous impact of licensing on a company's success, software companies need to be in control of in-licensing decisions on the one hand and out-licensing on the other.

17 The OwnCloud Contributor agreement can be inspected on the website of OwnCloud Inc. (https://OwnCloud.org/contribute/agreement/).

They need to have their developers trained so that they are capable of make decisions in accordance with the wishes of their employers. It is also important to establish policy guidelines (Helmich & Riehle, 2012).

At OwnCloud, key licensing decisions are always made by the board, just as the co-founders developed together the basic licensing concept for OwnCloud Inc. Regarding in-licensing of third party FLOSS, only modules and libraries with permissive or, at most weak reciprocal licenses are accepted. On the other hand, as part of their dual licensing strategy the co-founders made the decision that, for the FLOSS part, always strong copyleft licenses have to be chosen, when it comes to distributing their product. Firstly, this ensures that the software always stays FLOSS, which was already the goal of Karlitschek in the early days of the OwnCloud Community. For the company, too however, this is of strategical relevance as only a GPL or alike can prevent proprietary companies from evolving into competitors on the basis of the groundwork as developed by OwnCloud.

2.5 The IOS Client Licensing Issue

2.5.1 The OwnCloud Mobile Clients

The different clients are key components in the software architecture of OwnCloud, as only they facilitate a true mobile file sync and sharing experience that allows users to pick up their files on one device in the currently updated state as last saved on another. It took OwnCloud almost two years to have the file synchronization up and running as desired. In addition to OwnCloud's desktop client, which is available for PCs running Windows, MacOS or Linux, OwnCloud offers IOS and Android clients to cover most mainstream mobile operating systems. All clients have the same technical construction in both editions, but as part of OwnCloud's dual licensing strategy the licenses they are available under can differ. Both clients are split into a front end and a logical component. The front end covers the representation of the data for the user and was developed by OwnCloud Inc., whereas the logical component enables the synchronization mechanism and was in-licensed under MIT licenses, in accordance with OwnCloud's policy of only reusing components under a permissive license or LGPL.

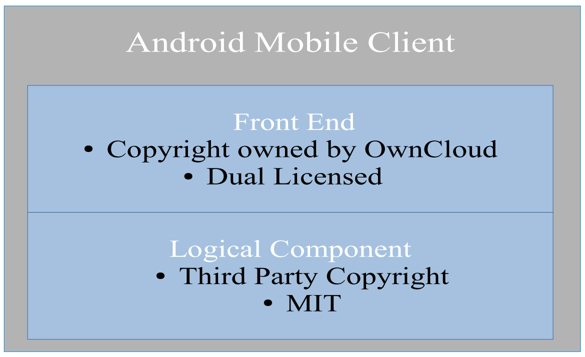

Figure 5: Setup of the Android Mobile Client

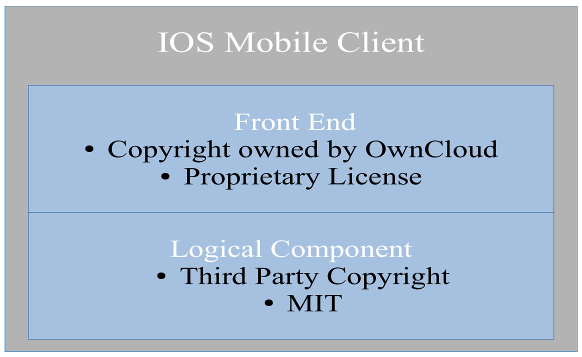

Figure 6: Setup of the IOS Mobile Client

Originally, the development of the clients had been driven by the OwnCloud Inc. In August 2012, when OwnCloud released the first version of the mobile clients, it was decided that only the Android client was to be open-sourced in order to enable the community to participate. In line with the dual licensing strategy the Android client was out-licensed via a GPLv3 for the

Server Edition and under an OwnCloud proprietary license agreement for the Enterprise Edition. Regarding the IOS client, the OwnCloud management decided at that time that they would only be able to out-license it under a proprietary license agreement.

In 2010 Apple, for its part, had stopped accepting any apps licensed under a GPL as incompatibilities between the license and the IOS App Store terms could not have been ruled out (Contributions, 2010). But OwnCloud's distribution strategy of mobile clients — as designed by the three co-founders — distinguished between Server and Enterprise Edition. Users of the Server Edition should be able to purchase clients from the online app marketplaces of Apple and Google only. The ability to redistribute clients freely to their end-users as needed, should be retained exclusively to OwnCloud's Enterprise Edition customers. To keep that strategy intact, the IOS client of the Server Edition had to be compatible with the terms of the IOS App Store. Thus the co-founders evaluated the situation that they would need to go without an open-sourced version of the IOS client to this time.

2.5.2 The IOS Client Licensing Decision

Holger Dyroff, still waiting at the traffic light, recalled the situation in which their decision, made back in 2012, had left them. He now focused on the differences between the Android and the IOS client.

Firstly, the Android client was dual-licensed and available under the GPLv2 and a proprietary license agreement, whereas the IOS client was available only under OwnCloud's proprietary license agreement. Also, the source code of the IOS client was not made publicly available. OwnCloud Server Edition users and community members had no choice but to use this proprietary application in order to get the best experience of OwnCloud on their iPhone or iPad. Android users, however, did not have to leave the FLOSS universe as they got offered the GPLv2 licensed client. Consequently, only Android users where able to review the code of the client and to participate in the development of the software.

Secondly, the development for the clients was done using different programming languages. The IOS client is based on Objective-C technology whereas the Android client is developed in Java, which is more popular in FLOSS communities (Metz, 2015).

A third difference occurred in the manner in which the clients could be distributed. In order to be obtainable by users, the mobile clients had to be offered through the application marketplaces of Google and Apple. The Google Play store accepts GPL licensed applications for distribution. In principle, OwnCloud Server users were able to upload own variations of their OwnCloud Android client, also branded — even though OwnCloud Inc. aims to ensure that only Enterprise Edition customers should be given the option of putting their own logo into the client. That made the Android client better commercially utilizable for OwnCloud Server users. Without the Enterprise Subscription, OEMs could also offer an Android sync client with their corporate logo without needing to subscribe to the OwnCloud Enterprise edition. OEMs needed only to be willing to accept the GPL conditions and to open-source their code if required to do so by the license.

The OwnCloud management team had a certain strategy in mind, when the mobile clients were first introduced. They agreed that the Android app would be more important for the community, as Android had more users in the OwnCloud community than IOS did. Therefore, and in order to comply with their FLOSS values, they did not hesitate to open-source the Android client. But OwnCloud Inc. was also seeking to gain larger businesses as customers for their Enterprise Edition. Apple devices were known to be very popular in that customer segment. Dyroff's, Karlitschek's and Rex's rationale was that the customer group they were targeting could not pass on also having an IOS sync client in their portfolio in order to cover all employer needs for a smooth EFFS solution.

Thus, the IOS client was offered solely under a proprietary license not only because of the incompatibility of GPL and Apple's IOS App Store. This single component was also given a proprietary license agreement to avoid giving the target group another reason to stick with the gratis OwnCloud Server solution. In this way, it was ensured that no potential customer would be able to offer the complete set of sync clients. With this factor the co-founders wanted to give one more incentive for potential customers to choose the Enterprise Edition. Also, OEMs would not have the chance to structure their own offer with their logo on IOS clients. So the strategy not to open-source the IOS client as the only component of the OwnCloud architecture was adopted with an eye to generating revenue and hence commercial success — factors even the most dedicated FLOSS firm finds it hard to neglect.

But the strategic decision came at a price. It did not take long for a lively discussion to be sparked inside the community. Some members just complained that they had no possibility to review the code, or requested open-sourcing of the code in the near future. OwnCloud should also openly publish the IOS source code on Github.[18] Others even questioned the whole OwnCloud solution, because it had a fully proprietary component. Although only a fraction of the large community was really troubled with the setup, the arguments weighed heavily in the minds of the three co-founders.

In addition, OwnCloud management had to realize that the development of the Android application was progressing much faster than that of the IOS client. The IOS app was missing features which already had already been implemented in the Android solution. The IOS client was developed under exclusion of the public. As no user stories of handling by the community could be published, all progress needed to be made by OwnCloud employees. Last but not least, the OwnCloud IOS development team had no assistance when finding, characterizing, and fixing bugs.

Dyroff, wanted to use the meeting with the other co-founders to finally agree on measures to address this issues. On the other hand he knew, they would have to be careful to keep the right barriers in place in order to prevent potential customers from getting too cozy with their gratis FLOSS offering. Like any company, they could not allow their business success to be jeopardized. If they were to open-source the IOS client, which FLOSS license would be involved best? Was a multiple licensing strategy, as with the other clients, ultimately the better way? Could they find a solution that would allow for both: putting an end to the negative effects identified with regard to the IOS Client while still providing an incentive for potential customers to subscribe to the Enterprise Edition?

18 Github is a web based repository hosting service allowing to collaboratively develop software. Since 2012 the development of the OwnCloud Community is performed on Github in a public repository (https://github.com/owncloud).

2.6 Exhibit 1: Outline of OwnCloud Business Model

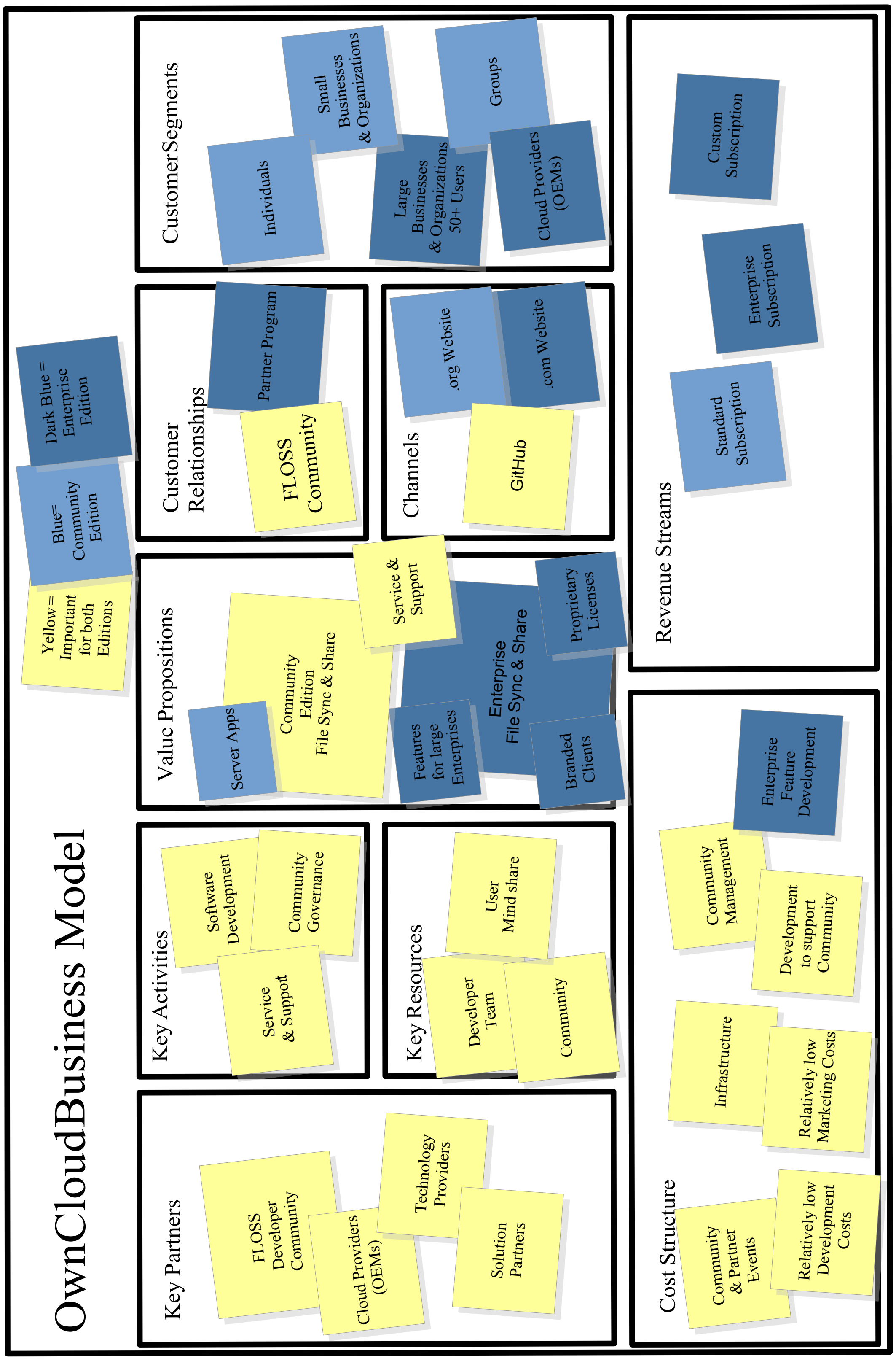

Figure 7: Outline of OwnCloud Business Model according to the Business Model Canvas Template by Alexander Osterwalder

2.7 Exhibit 2: Comparison of OwnCloud Editions

	OwnCloud Server Edition	**OwnCloud Enterprise Edition**
Type of Licensing	FLOSS licensed	Proprietary licensed
Governance	OwnCloud Community	OwnCloud Inc.
Targeting	FSS market • Individuals • Small to medium organizations	EFFS market • Large organizations • OEMs
Based on		OwnCloud Server Edition
Distribution of Mobile Clients	Google Play and IOS App Store • Users need to purchase mobile clients from the online application market places of Apple and Google.	Companies can redistribute clients according to their terms • Customers can obtain mobile clients directly • Companies can redistribute clients to their employees or own customers • End-users do not have to purchase mobile clients on Apple's or Google's online application markets
Additional Value for Users	Community Applications • Available for a fee via the OwnCloud App Store • Offering additional features • OwnCloud as a platform for functionality beyond FSS • Sample apps: TextEditor, Anti-virus, Gallery, etc.	Enterprise Applications • Distributed as bundle with the OwnCloud Core • Features typically interesting for large companies • Sample apps: Microsoft Share Point Integration, Logging, File Firewall, etc.
		Commercial License • Enterprises can avoid FLOSS risks of sharing IP unwantedly • Commercial License offers almost the same freedoms as FLOSS licenses with the exception of redistribution
		Rebranding of Clients • Customers can use their own logo to integrate OwnCloud into their Corporate Design
Availability	Freely downloadable and usable • Support via the community forums	Enterprise Subscription • 5X12 email and phone support • From 50 up to 100,000 Users
	Standard Subscription • 5x8 email Support • From 50 up to 1,000 users	Custom Subscription • 24x7 email and phone support • From 10,000 users

Table 1: Comparison of OwnCloud Editions

2.8 Exhibit 3: Different Copyleft Effects for different Types of Modification and Combination of Software

	Derivative Work	Combined Work	Collective Work
Type of Modification/ Combination	Modification/ incorporation	Linking	Mere aggregation of single components
Example Application	Modification of reciprocal licensed source code to add a new feature	Usage of libraries or plug-ins which are under a reciprocal license	Combining different programs to a software suite and one component is under a reciprocal license — like a Linux-Distribution
Copyleft Effect	Resulting software is to be put under same reciprocal license	Whole Combination is to be put under same reciprocal license — Exception: LGPL	No Effect, whole combination is free of licensing choice — single components keep their original licenses
Likelihood of viral Effect	Certain	High	Low
Compatibility	All other licenses need to match the terms of the involved reciprocal license	All other licenses need to match the terms of the involved reciprocal license	Unproblematic

Table 2: Different Copyleft Effects for different Types of Modification and Combination of Software

2.9 Exhibit 4: Most important Licenses at OwnCloud

License Name	License Type	Usage at OwnCloud	Key Properties	Derivative Work	Combined Work	Collective Work
GPLv2	Reciprocal License	Out-licensing	Strong copyleft Attribution	Derivative must be released under GPL	Combination must be released under GPL	No restrictions
GPLv3	Reciprocal License	Out-licensing	Strong copyleft Enhanced compatibly Software patents Attribution	Derivative must be released under GPL	Combination must be released under GPL	No restrictions
AGPLv3	Reciprocal License	Out-licensing	Strong copyleft Network reciprocity Software Patents Attribution	Derivative must be released under AGPL	Combination must be released under AGPL	No restrictions
LGPL	Reciprocal License	In-licensing	Weak copyleft Attribution	Derivative must be LGPL or GPL	No restrictions	No restrictions
MIT	Permissive License	In-licensing	Attribution	No restrictions	No restrictions	No restrictions
BSD-2-Clause License	Permissive License	In-licensing	Attribution	No restrictions	No restrictions	No restrictions
Apache 2.0 License	Permissive License	In-licensing	Attribution Software patents	No restrictions	No restrictions	No restrictions
OwnCloud Commercial License	Proprietary License	Out-licensing	No redistribution allowed Otherwise granting rights similar to the freedoms of FLOSS			
OwnCloud EULA	Proprietary License	Out-licensing	No rights granted — no free use, no open source code, no modification, no free distribution — unless explicitly permitted by OwnCloud			

Table 3: Most important Licenses at OwnCloud

3 Conceptual Background

3.1 Intellectual Property Rights

Intellectual property (IP) is an umbrella term that comprises all creations and innovations of the human mind. This includes all kinds of inventions and literary and artistic works. Authors and inventors are granted exclusive rights by the legal system to permit and control the use of their creations. Intellectual property rights (IPR) shall ensure that authors and inventors are able to benefit from the work and investments they put into their creations. Following the definition as provided by the World Intellectual Property Organization (WIPO), this protection will encourage further innovation and thus lead to further economic growth (Idris, 2003).[19] However, the provided rights are limited in scope, duration, and geographical reach. The limitations and characteristics of intellectual property law can vary from country to country, according to the respective countries different legislations. Also, jurisdiction is different for each country, and case law derived from court decisions in one country may not necessary lead to a similar enforcement in another country. Observed on a global level, IP law cannot, therefore, be seen as homogenous. Nevertheless, international partnerships und agreements exist to narrow down the differences for the associated countries and enable an international protection of IP rights. A prominent example for an agreement that is very powerful, since it reaches out almost globally, is the Agreement on Trade-Related Aspects of Intellectual Property Rights (TRIPS), which was negotiated in 1994 by the members of the World Trade Organization (WTO).[20] The three primary types of IPR are Trademarks, Patents, and Copyrights.

3.1.1 Trademarks

Trademarks protect the logo, slogan, or even tune used as an identifying tag for a company or a product. A trademark's main purpose is to make ones company's product or service easily distinguishable from those of the competition, and to prevent misleading signaling by having third parties using one's logo unlawfully, for example. This also means that trademarks are owned exclusively by one entity in order to ensure the clear association between trademark and right holder.

To gain the full set of rights for a trademark, it has to be registered with the patent and trademark office. A standard registration will give protection only for the area in which the trademark was registered. For example when registering a trademark at the German Patent and Trade Mark Office (DPMA), protection will be granted for Germany only. So trademarks filed in Germany will not offer protection in the rest of the European Union or the United States for example. If broader protection is desired, cross-border protection can be achieved based on international partnerships and agreements. In order to ensure protection for all jurisdictions in the European Union, a *community trade mark* can be requested at the European Union's Office for Harmonization in the Internal Market (OHIM). To achieve

19 WIPO is an agency of the United Nations (UN) which promotes the protection of IP and governs most of the treaties for international property law. WIPO was created in 1967 and is based in Geneva. WIPO offers a wide set of publications regarding IP. The book by Idris is one of those publications.

20 TRIPS became effective at the beginning of 1995 and provides minimums standards for IP regulations to which the WTO member states guarantee to adhere. An important amendment was passed in 2005 standardizing regulations on patents. The WTO currently has 162 members including the United States and the countries of the European Union. An overview over the single member states is provided on the website of the WTO (https://www.wto.org/english/thewto_e/whatis_e/tif_e/org6_e.htm).

protection in the United States as well, additional registration at the United States Patent and Trademark Office (USPTO) would be an option. Maximum global protection could be achieved by filing an international registration with the WIPO offering protection for all jurisdictions that signed the *Madrid Agreement c*oncerning the international registration of marks, including the United States and the countries of the European Union.[21] For registrations usually a specialized attorney is needed and a considerable fee has to be paid, one that increases in proportion to the size of the protection area.

The term of protection for trademarks is not limited if these are being actively used by the right holder and extensions of registrations are maintained on time. Also, trademarks have to be actively defended in the event of infringement in order to maintain their protection.

In the software industry, just as in any industry, trademarks are used to indicate who made the product. *Trade dress* protection, which Graham and Deepak (2004) classify as a variety of trademark protection, can also play an important role in the software industry. Trade dress focuses on the protection of a product's visual appearance. Applied to a computer program, it can be used to protect the *look and feel* elements of, for example, a graphical user interface (GUI).

3.1.2 Patents

Patents are designed to protect inventions and technical components. A patent covers the whole *concept* of the technical innovation and the idea behind all implementations of a particular invention. This is one of the biggest differences between patents and copyrights, as copyright protection focuses on the protection of an individual *expression* of an idea, whilst patents protect the underlying function of it.

Patents grant their owners the exclusive right to prevent others from producing or using their invention. Because of this, owners can set monopolistic prices for their work to obtain compensation for the development costs behind the idea. Patents offer a strong way of IPR protection. Once granted, they even protect the idea, if someone else comes up with it independently after the patent has been filed. The other inventor would not be allowed to produce or to use the concept without the patent holder's permission. Everyone who would like to use, distribute, or produce a patented functionality would need to ask the patent owner for permission or otherwise risk prosecution. Owners can license the use of their patents to others and get compensated for this.

Patents need to be registered at the patent office in the inventor's jurisdiction. The registration is a long and costly process in which patent attorneys also need to be involved. For an application to be accepted, the invention has to meet the criteria of novelty, usefulness, and non-obviousness. To make an application, a patent document has to be submitted that outlines the purpose and technical implementation of the invention on a technical and detailed level. Diagrams detailing the construction and flow charts also need to be included, in order to provide an overview of the manner in which an object functions. Following, the granting of a patent, all these details of the invention are disclosed to the public.

The term in which patent owners have exclusive rights is limited. According to the TRIPS Agreement, the protection should be available for a minimum of 20 years. Most members of the TRIPS union have adopted this suggestion. For example, patented inventions will go into

21 Currently, 96 states are members of the Madrid Union. For an overview of membership, please consult the list of members as provided by the WIPO (http://www.wipo.int/export/sites/www/treaties/en/documents/pdf/madrid_marks.pdf).

the *public domain* after 20 years in the United States and Germany. On works released to the public domain, all IPR have expired and inventions are free to be used by the community at large. At this point, it becomes possible for practically anybody to use the disclosed but previously protected inventions.

The *prior use* exception grants the right to maintain usage of an invention that falls in the domain of a valid patent if the user was using the same idea first. In addition to prior use an exemption for research purposes is possible.

In terms of territorial rights, patents offer protection only in the jurisdiction in which they were filed and permission got granted. If inventors would like to protect their concepts internationally, they need to apply for patents in all the countries in which they would seek protection. For example, USA patents are, by default, not valid in Europe. In addition, patent law can vary from jurisdiction to jurisdiction, but the outlined requirements and regulations have been extensively standardized internationally. There is one important difference regarding requirements for the acknowledgement of patents, that could be named in a comparison between the United States and the European Union. According to EU patent law, an invention has to solve a *technical problem*, whilst in the United States there is not this explicit focus on a technical application (Välimäki, 2005).

In addition to the TRIPS agreement, the Patent Cooperation Treaty (PCT) is an important foundation for establishing a basis for international patent law. The PCT dates back to 1970 and is governed by WIPO. PCT provides a simplified way for the international application of patents by allowing to apply in all 148 member states simultaneously.[22] The PCT application can be filed at one's regional patent office. The PCT application will be assessed by international authorities and published by WIPO. After this assessment the national patent offices of the member states are notified and, in parallel, the single national authorities decide wether to issue the patent for their jurisdiction.

Patents are not the main IPR choice for protecting software. As explained further below software is mainly associated with copyrights. But it is possible to have software patented, which may seem desirable for developers if they wish to protect the whole idea and not only the individual expression of a program or piece of code. In the United States in particular, the patenting of software is becoming more and more practiced and the chances of being granted a software patent are not inferior to other kind of patents. In other jurisdictions, software may not always be seen as patentable — especially in the European Union, on the basis of the European Patent Convention (EPC) of 1973.[23] Article 53, paragraph 2 of the treaty excludes *programs for computers* from patentability, as computer programs are not understood as providing a solution to a technical problem as such. As Välimäki (2005) recapitulates in his book, it is nevertheless possible to acquire patents on software in the European Union as long as the program is the only means to create a technical effect and, as such, is a solution to a technical problem. Also, thousands of software patents have been accepted by the European Patent Office. The status of software patents in the European Union is therefore somewhat

22 Since July 2013 the International Patent Cooperation Union, which is formed by the contracting states of the PCT has 148 members including the countries of the European Union and the United States (http://www.wipo.int/pct/en/pct_contracting_states.html).

23 With the EPC treaty in 1973 the European Paten Organization became instituted. The last revision took place in 2000. The treaty has 38 members including all the states of the European Union but also other states of the European area such as, for example, Turkey (http://www.epo.org/about-us/organisation/member-states.html).

ambiguous as the main legal signal is not to accept them, yet, on the other hand, software patents never have been actually banned in Europe.

There is a lively global discussion about the economic impact of patents and whether they foster innovation. This discussion is especially active for the software industry, which is argued to be special on account of its short innovation cycles and its development processes, these being sequential and cumulative because developers usually incorporate existing technical solutions into their programs. Thus, the more ideas that can be picked up, the quicker the industry can spawn new innovations. Due to the number of patents and their broad — sometimes almost ambiguous — technical descriptions, it can be hard to obtain an overview of which concept is patented and what idea or small technical solution is free to be used. This state, firstly, creates the risk that a patent will be infringed unknowingly by a developer. This risk applies even for inventions made independently of, and unaffected by, the patent, since a valid patent covers the whole idea even if others come to the same concept on their own (Laurent, 2004). Secondly, the sequentially and cumulatively growing software industry is highly interconnected with regards to its innovations. Attaching the right to exclude others to the industry's novel ideas or technical solutions can create an untraceable thicket of patents that eventually may rather prevent innovation than foster it (Stallman, 2001).

3.1.3 Copyrights

Copyrights are granted for creative works. It is the form of IPR granted to creators for novel and individual creative expression. Placed under the protection of copyright law are all forms of artistic works such as art, literature, music, and writing. It also, as it is explained further down, protects software.

In order to be copyrightable, creative expressions need to be brought to life in a tangible medium. This would mean that one's thoughts would not be copyrightable — unless spoken and recorded on tape, for example. If an artist sketches a flower on a napkin this representation of the flower would be protected by copyright. As this example also describes, copyright applies as soon as a creative work is written, painted, or recorded somewhere. No one has to specially register creative works in order to have them protected since it is granted by default.

Copyright grants creators the exclusive right to copy, publish or — generally speaking — monetize their creative works in any form. If somebody else wants to commercially use the creator's expression, the creator has to be asked for permission. As only the fixed representation is protected, copyright can not prevent other expressions of the same idea. In the example with the sketch of a flower, not the general idea of a flower is protected, but only the particular drawing of it, as it was produced on the napkin. Thus copyright is considered to offer a weaker form of protection than patents, which protect whole general concepts (Lindberg, 2008).

The owner of the copyright can pass it on to others.[24] Equally, the owner may just grant certain rights to others, while otherwise retaining the copyright. This action is known as licensing. Both, transferring and licensing can be done by the owner of the copyright for free

24 Technically, the complete assignment of copyright via a contract would not be possible under European copyright law. For example, paragraph 29 of the German copyright act specifics that copyright is not transferable, except when the author passes away. Instead of copyright assignment, in Germany often, the exclusive right of use is licensed. In the United States complete assignment of copyright is standard practice. Please keep this distinction in mind whenever assignment of copyright is referred to in this text.

or against payment of a fee. Payments made within the context of licenses are known as *licensing fees*, *royalty payments,* or simply *royalties.*

The right to create *derivative works* from a fixed representation is also one of the exclusive rights granted to the copyright owner. As Laurent (2004) outlines, the concept of derivative works is very important for FLOSS licensing. A derivative work is always based on a preexisting work. This means that, when compared to the original, the derivative work is no work of its own. Only the copyright owner of the original has the right to create such a derivative work. For artistic works it is hard to provide an example of derivative works, but for software it is easier. Imagine that a first software work is put as a code module into a second program and the second program can only run on the basis of that code part. This would classify the second program as a derivative work that would thus only be able to be produced by the owner of the copyright of the first program, or a person or company to which the owner has granted the right to do so.

There is an exception to the concept of derivative works known as *transformed derivative works.* In this case, works based on an original are altered to the extent that a new work comes to life , one independent of the copyright demands of the owner of the original work. Transformed derivative works therefore allow the use of copyrighted works of others without copyright infringements.

Within the context of using works of others without copyright infringement, the doctrine of *fair use* must be mentioned. Fair use is a flexible standard of copyright law that allows certain distribution of copyrighted material, as long as this does not interfere with the owner's exclusive rights to commercially exploit the work. Also, the quantity of the work used, the effect on the work's value or its market, and the purpose — for example non-profit educational intent of usage — can be examined in order to evaluate whether fair use applies. According to Lindberg (2008) , commentary — like the educational use of the original or usage for book reviews — are a common application of fair use.

Another question is wether it is acceptable to redistribute legally sourced copyrighted works. The *first-sale doctrine* allows for this type of forwarding. For example, a purchased CD-ROM may be rented to friends or sold at a garage sale, but it would not be acceptable to distribute a modified copy or other duplications thereof.

As so far implied, the author of a creative work becomes automatically the initial owner of its copyright. But there can be a circumstance in which the creator of the work may not be the owner of it, even initially. If the work was created by an employee as part of prescribed workplace duties, it has to be treated as a *work for hire*. Owners of works for hire are not themselves the originally authors thereof, but the entity they are employed by. This distinction is especially important for the software industry as software is largely developed inside companies. Also with regard to open source, it is important to know that lawyers tend to recommend that one checks to evaluate wether code written in a hired programmer's spare time — for example in order to contribute to an open source project — is affected by this principle (Lindberg, 2008).

As trademarks, patents, and also copyrights are territorial rights, protection is not perfectly harmonized internationally. A milestone towards global homogenization was the *Berne Convention* for the protection of literary and artistic works in 1886. With the Berne Convention, an international union for the protection of the rights of authors was created. The intergovernmental treaty is now administered by the WIPO and had its last amendment in

1979.[25] Currently, the union has 168 members, including the United States and the countries of the European Union. The treaty is the basis for international copyright law as states tended, prior to its enactment, not to acknowledge the worthiness of protection for works created in other nations. That had severe implications for creators, who had to be fear that their works could be distributed freely outside of their jurisdiction. With the Berne Convention treaty, members agreed to grant the same protection to works of citizens of other states as to the works of their own country. Also, the treaty provides a framework regarding copyright law — including the principles of copyright presented in this chapter — that every member acknowledges. The principles agreed on serve as a basis that characteristics can be extended by the member states wishing to go beyond these minimum requirements of the treaty.

An example of this, and the resultant differences between single member jurisdictions, can be seen with regard to the time limit of copyright protection. According to the Berne Convention, a minimum term of protection covering the creator's lifetime plus additional 50 years after the death of the creator has to be ensured. Many jurisdictions — including Germany and the United States — have extended this to a 70-year term after the author's death. In the case of works made for hire in the United States the term of protection amounts to 95 years from the date of publication, or 120 years from the date of creation. In the European Union, the term for such works is determined with reference to the life of the last surviving author, providing 70 years of additional protection to it.

When the copyright protection period for a creative work expires, the work automatically enters the *public domain*. As no copyright on material in the public domain exists, everyone is allowed to copy, distribute, create derivative works and commercially exploit such works. As copyright is granted by default, every work is initially protected. Creators wishing to release their work into the public domain thus have to do so intentionally by declaration. FLOSS licenses do not explicitly not transfer a work into the public domain as the author remains the owner of the copyright when using a FLOSS license.

Also, software in the form of its source code is protected by copyright. Software is copyrightable as writing code is understood to be an authorial, creative act like writing literature. Even though code is written to perform a function — and, as such, has an appearance of a rather non-artistic but technical component — the author's idea is unique and protectable, as there are different ways in which code can, in order to implement the function, be written. Thus software is handled as a piece of artistic art and copyright protection applies. In line with the explanation of the example with the sketch of the flower, only the particular source code, and its form of expression by the programmer, is protected by copyright — not the idea embodied in the software, nor the way in which the functionality is performed within the program (Välimäki, 2005).

In principle, using a copyrighted work is not restricted by law. It is therefore acceptable to enjoy a beautiful painting or to listen to music. But as it is an infringement to make an unauthorized copy of a copyrighted work, copyright would in general prohibit the use of software. The reason is that, in order to use a software program, it first needs to be loaded into a computer's memory. This process would create a per se disallowed copy. Thus, to keep software usable, a special exception regarding software is granted by law. It is okay to create copies which are required to run a program on a single computer as long the source for that copy, for example a CD-ROM, has been legally obtained by the user. Also, it is permissible to

25 The 1979 version of the revised Berne Convention can be found on the WIPO homepage (http://www.wipo.int/treaties/en/text.jsp?file_id=283698).

create a single copy of software programs to be able to archive them and to have a backup available.

3.2 Software Licenses

Software licenses represent legal permissions to perform certain actions issued to the users of software. There are two rather different concepts of licensing in software that need to be distinguished.

The first concept is very important for proprietary software. In this context, software licenses grant the right to usage and forwarding of single copies of software. The licensing is done via *licensing agreements* which impose certain limitations and obligations on the licensee, and compliance is a condition for usage.

The second concept of software licensing can be used to allow users actions with software that would otherwise remain as exclusive under copyright, like the rights to make copies, modifications and to distribute those works. Here again, certain limitations and obligations can serve as basis for the license. This concept is used in FLOSS licenses. But also for proprietary software such rights can be licensed by the owner of copyright, mostly in return for the payment of license fees.

In general licenses represent the allowance to do or to use something (Rosen, 2005). For example, a time-limited usage right of a premise can be licensed to an interested party, if the owner does not actually want to sell the property.

In equivalent manner to tangible property, the usage of software has to be allowed by the IP holder. It is upon this that the first possibility of licensing — as described above — focuses. When software is acquired, the user typically does not gain the ownership of the software but only the license to use it. Generally, the license is assigned to the licensee as part of a purchase contract which specifies what the user has to do in order to be allowed to use the product. Typically, the purchase contract requires the licensee to pay a fixed price or recurring payments over the useful life of a subscription. Such a trade can be made in a shop when purchasing a CD-ROM or via the Internet when downloading software. Such trades based on purchase contracts make it possible for users to legally obtain a copy of the software that they are now able to use under the terms of the license. Purchases must not necessarily be made with the owner of the IP directly. As explained in the copyright section, a distribution-chain is possible because of the first-sale doctrine. Software is therefore often distributed via resellers. Typically, licensing is done via license agreements. License agreements are a certain form of contract in which the user accepts the terms of usage as imposed by the copyright owner.

Licenses and license agreements — irrespective of wether these grant mere usage rights or rights which are otherwise exclusive to copyright owners — typically contain a paragraph making it clear that the whole license will *terminate* if the licensee does not comply with any one single term of the document. Upon termination, all rights given expire and the licensee may no longer retain the right to use the software. Also, distributing copies or modifications will infringe the rights of the copyright owner, if such were ever granted.

A general principle of licensing is, that licenses need to be directly issued by the right owner. As software licensing focuses on whole programs as created by developers, copyrights have to be seen as the legal foundation for licensing (Välimäki, 2005). But *software patents* can play a role that must also be considered when allowing usage or copying, modification, and distribution of software. The usage of patents can also be licensed, a process which usually involves the payment of expensive license fees. If a program as a whole is licensed within the scope of copyright law, also potential patents, which may protect certain used code modules of the program, will have to be licensed before the user may run the program. At this point, the high degree of interconnectedness — occurring as soon as code is reused — within software licensing, gets noticeable. Developers who have incorporated code parts developed by others, need to be legally able to license this software, that contains a combination of reused modules and newly added code. Thus, they must obtaining licenses for doing this from every single copyright owner and patent holder whose programs have been used. Otherwise, the developers will be infringing IPR and may face serious punishment by law. As reuse of code is very common, the resulting network of licensing is huge and it can be hard to trace which code fragment is patent-protected and by whom (Stallman, 2001).

Licenses and license agreements include limitations of liability and warranties. Also they often have a passage that seeks to limit liability if the software is found to infringe third-party IPR.

As already indicated, there are two main types of licenses: *proprietary software licenses* and *FLOSS licenses.*

3.2.1 Proprietary Software Licenses

As mentioned at the beginning of the licenses chapter, proprietary software licenses represent one concept of software licensing. As proprietary software is dominant in the software mass market that is mainly targeting individuals as end users, proprietary software licenses represent the most common form of software licensing. Proprietary software licenses aim to allow a specified usage of the software the licenses are applied on. The actions the user is allowed to perform and the number of software copies the user is allowed to use, tend to be narrowly described in the license. Proprietary licenses also explicitly restate the limitations emerging for the users because the licensor is the copyright owner. It is typically reasserted in the text, that the user is prohibited to copy, modify or to distribute the software. Also, such licenses mostly do not allow access to the source code and exclude the user from either learning from it, or from making changes to the program. Because of this feature also the term *closed source software* is often used for proprietary software.

As already stated, the usual form of licensing for proprietary software is by the means of license agreements. Especially for mass-marketed software, like the office applications from Microsoft, it is typical that users first have to agree with the terms of an end-user license agreement (EULA) before they are allowed to use the obtained copy of a program.

The license agreement texts will make sure that the whole license will become invalid if the user does not comply with any of the usage terms. Anyone who has no valid license, or who has infringed the terms, is not allowed to use the software and must fear prosecution if acting against this prohibition.

Different licensing models are possible and these also impact the payment models adopted in the purchase contracts used for distribution of the software. For example, for mass-marketed software it is normally the case that users have to pay a certain amount per copy. Software specifically designed for corporate use on the other hand — like enterprise resource planning (ERP) applications — can often have a revenue stream charging per issued license. Different models are possible, such as licenses granted per user or per device, or even per processor core.[26]

26 Microsoft, for example, uses the number of processor cores as basis for licensing their SQL Server

The terms of most license agreements will not permit the making of copies of the licensed software. If users — distributors, for example, as part of a business model — would like to distribute self-made copies of a software product, they would need to ask the copyright owner of the software and obtain a license from him, that explicitly grants rights otherwise prohibited by copyright. This was introduced as a second software licensing possibility at the beginning of the licensing chapter. The most likely option for proprietary software would be for the copyright owner to demand royalties. License fees would be based on the number of copies the distributor produces. Here, software licensing follows the distribution model of another mass-market medium, book publishing. The author earns money with every copy the publisher produces. As outlined in the next section, this can be another major difference when compared with FLOSS licenses.

3.2.2 Free/Libre Open Source Licenses

FLOSS licenses do not pursue the aim of restricting the usage rights of software users to a limited set of explicitly granted exceptions. The concept of FLOSS licenses goes one step further than simply dealing with rights concerning the *use* of software. These type of licenses are firmly rooted in the framework provided by copyright. At the level of copyright law, it is not prohibited to apply this to software, as long it does not interfere with the exclusive rights secured by the copyright owner. Licenses are only needed if users are to be granted rights that would otherwise be exclusive to the copyright owner. Such a case exists if software developers want to distribute — together with their software — the right to make copies and modifications, and to grant the right to redistribute these derivative works. In this case, developers explicitly need to license the right of redistribution. Thus, FLOSS licenses grant rights to third-party software users that they would not otherwise be given.

Two institutions ensure that all licenses labeled as free software or open source software licenses are allowing for certain key principles. One of these organizations, the Open Source Initiative, has been already introduced in the previous chapter. OSI maintains the Open Source Definition and accredits licenses which comply with its 10 principles. Also, the Free Software Foundation — an organization founded by Richard Stallman, the author of the GPL and initiator of reciprocal software licensing — demands that certain freedoms are granted to licensees of software in order that the software qualifies as free software. The four freedoms of the Free Software Definition resemble the requirements of the Open Source Definition to a high degree (https://www.gnu.org/philosophy/free-sw.html). Both works assure that FLOSS software satisfies two essential conditions:

First, they ensure that users are free to apply the software, and can study, modify, and redistribute its source code. The terms of the license attached to the software must allow for that.

Second, the software's source code must be made available, for example via download. This applies as soon as FLOSS software is distributed. Access to the source code is essential to perform modifications or to be able to study software distributed with FLOSS licenses.

Unlike proprietary licenses, FLOSS licenses do not depend on contracts wrapping the licenses in the form of licensing agreements that need to be accepted prior to usage of the software. Even though FLOSS licenses also may enforce terms and limitations on the user, and may have the form of a contract, they usually stand for themselves. Because FLOSS licenses impose requirements only when software is distributed, it is not necessary to confirm the

application (“Microsoft Volume Licensing, n.d.).

license prior to usage of software. Conversely, proprietary licenses need the license agreements to limit usage to narrow terms before the user runs the software. By contrast, the copyright-focused FLOSS licenses do not interfere with any use of the software that complies with IPR requirements. Terms are imposed only when the user wants to distribute copies or derivative works. Here, IP law provides an especially solid basis for the working method of FLOSS licenses, thus ensuring the licenses' validity and obligations. Laurent (2004) describes this phenomenon as the “self-enforcing nature of open source licenses”(p.151). As software, by default, is copyrighted, only the license will enable users to distribute copies without infringements. Thus, users failing to acknowledge the open source license, or acting against one of its terms, would basically have no means of distributing copies. If users want to exercise rights provided by a license, they are forced to accept all limitations and obligations that come with it. The fact that copyright infringements tend to be strictly penalized at courts, and substantial financial penalties can be imposed, serves to support compliance with FLOSS licenses.

FLOSS licenses do not apply practically for the production of copies, that are meant to only b e used privately or within the boundaries of a corporation. For example, a company customizing a Linux distribution for its special needs would generally not need to adhere to the terms of FLOSS licenses. Only if this derivative work were to be published, or otherwise distributed, would the creator's copyright of the original work be infringed. The company would thus need to adhere to the terms of the available license in order to avoid prosecution.

FLOSS licenses can be seen as the legal means that enables the whole open source concept to function smoothly. They make sure that software in the form of its source code can be changed, be enhanced, and freely shared. As well as granting these permissions, FLOSS licenses also help to encourage involvement in open projects and in developer communities. The attribution clause which is standard for FLOSS licenses can, for example, help to attract developers. Building a reputation is a major motivation for FLOSS developers (Välimäki, 2005). Therefore it can be very helpful to contribute to a code base that is publicly available. The FLOSS licensing terms can have strong signal effects that foster contributions to FLOSS projects. Also, they can ensure that the communities and projects remain healthy.

Lindberg (2008) explains the FLOSS situation by means of an analogy with the famous *prisoner's dilemma* used in game theory in order to explain the mechanics of FLOSS licenses.[27] In his understanding, open source software development has to be evaluated from the viewpoint of the community. To achieve quick and economic development, to produce robust software, and to expand into different markets, *cooperation* between the single developers is a necessary condition. Due to this cooperative nature of development, open source is prone to *free riders*. All market participants can benefit from the work done by the other developers who choose to make the code publicly available. This would not be an issue per se as long as all those who use code from the communities did not make amendments to proprietary software that exclude all others. If participants, like corporations which are in the proprietary business, use code obtained from the community in this manner, they can be

27 Game theory is a mathematical discipline used in economics to study how agents will act in situations in which their single decisions will influence the outcome for all involved. One famous model to examine how agents will react in such conditions is the *prisoner's dilemma*. It shows that even though the best outcome for two agents may be achieved if both were to cooperate, the incentives to defect may be stronger. Thus, it is unlikely that the result will be optimal. Both parties will have less in the end than they might have had through cooperating. An outline of the prisoner's dilemma and how exactly it can be adopted to FLOSS software licenses can be found in Lindberg (2008) from page 169.

classified as free riders. Unfortunately, the incentive for everyone to use the code without giving anything back is very strong, as in this way the individual may minimize costs. A status in which all developers cooperate is thus unstable. But cooperation is the basis for generating software from which everyone can benefit. All users would be better off if everyone were to share back their code.

FLOSS licenses provide a legal framework that stabilizes cooperation and can limit the free rider problem of open source development. Lindberg (2008) identifies that they “serve two functions in a game-theoretic context” (p. 171). First, they have a very strong signal effect. If the source code of one open source project is placed under a FLOSS license, potential contributors know the terms of the project and what they are, or are not, permitted to do with code they may share. Via the license, the intent of cooperation is signaled. Second, open source licenses grant rights that are otherwise exclusive to the copyright holder. As such, the initial developers can decide the terms successors have to comply with if they want to work with the source code. The requirements imposed on licensees can be used as a tool to assure that FLOSS software will remain open and that its derivatives will come with the same freedoms. Because of this effect, open source licenses provide the means to stabilize a cooperative atmosphere between the developers at an optimal level.

In chapter two, it has been already explained that there are two major groups of FLOSS licenses. On the one hand, permissive licenses do not impose strong terms towards licensees and may thus not be able to stabilize cooperation in terms of game theory. On the other hand reciprocal licenses demand that users will need to apply to the same conditions to their copies or derivative works. Anyone exercising the right to distribute a derivative work, but not adhering to the condition to make work freely available as well — the way in which free riders might act — violates copyright. The serious legal penalties the copyright holder may impose in such cases significantly increase the costs of the exclusive usage of the code. The mechanism of copyleft creates a strong incentive to give reprocessed code back into the publicly accessible pool of the community. Projects that use reciprocal licenses for out-licensing thus give a strong signal that the project's achievements will remain open over the longer run. From a developer's perspective, the use of licenses with copyleft may especially encourage contribution to FLOSS projects as the reciprocal licenses signals that developers will also be able to access future innovations.

Control over the copyright of software can thus be seen as crucial, enabling both cooperation and thus FLOSS to flourish. If developers were to give their copyright into the public domain instead of choosing an open source license, they would have no enforceable control mechanism to keep the code publicly accessible. This shows that FLOSS licenses, and most particularly the copyleft mechanism, are based on copyright law.

There is strong agreement inside the FLOSS community that it is better not to draft new FLOSS-like licenses but rather to select an existing one (Rosen, 2005). This principle is different to releasing software into the public domain or for proprietary software licenses. Releasing software into the public domain is done via informal declarations that have to be attached to the published software. Proprietary software license agreements, for their part, require unique customization, in turn, mostly require a draft for each usage case. On the other hand, FLOSS licenses have, in the first instance, to comply with the Open Source Definition and FSF freedoms requirements in the first place and do not so much have to respect individual varieties of the single cases. Rosen argues that using popular and well-known FLOSS licenses will have benefits for both licensors and licensees (Rosen, 2005). Licensees will better know what terms they are obliged to adhere to, and licensors will be able to profit from more precise assessment by the users. In addition, the mainstream FLOSS licenses may prove more reliable when compared to self-drafted versions if the licensor should need to enforce compliance in a lawsuit. It is therefore strongly recommended that only FLOSS licenses approved by either the FSF or OSI be used.

The selection of a license is based on many factors. As already mentioned in the case part almost no latitude is left concerning in-licensing decisions. If only a single desired component comes only with a license attached that is evaluated to have undesirable effects, another component with similar features will have to be found — unless the copyright holder of the component is willing to re-license it to grant better matching terms. Also, the compatibility with the licenses of the already incorporated components has to be borne in mind.

When out-licensing software components, the copyright holder has, as a first step, to evaluate the latitude left for arriving at a licensing decision. Depending on the licenses used for in-licensing, scope can be very limited. When choosing a license, the licensor first has to respect the conditions as required by in-licensed terms. If the work to be licensed is based on the reuse of a GPL licensed code for example, there is no choice left because of the copyleft mechanism. Software that is a derivative from GPL code, or is a collective work including a part that is GPL code, must be licensed under a GPL.

Presumed the case that, in the event of using any third-party software, only modules under permissive licenses have been incorporated into the product, the key decision of choosing a license is selecting the degree of restriction (Laurent, 2004). If copyright holders prefer not to sign over any of the rights that are — by default — exclusive to them, the best choice would be a proprietary license. Otherwise, the choice is between permissive FLOSS licenses, or reciprocal FLOSS licenses with strong or weak copyleft. Which license serves the purpose depends largely on the aims of the copyright holder who publishes the software. The licensor could primarily be aiming to earn money or, perhaps to make the software publicly available with as few entry barriers as possible. But, as shown by the teaching case presented, licensors should not fail to meet the requirements of their business model and the expectations of the community, especially within a FLOSS environment. Also, copyright holders should bear in mind the signal effect of their licensing choice. As previously outlined, projects signaling — by selecting a strong reciprocal FLOSS license like the GPL — that further innovation will also remain open may be able to attract a larger and more active developer community for their endeavor.

Exhibit CB-1 provides an overview of the attributes and goals licensors need to keep in mind when choosing licenses for their projects or code components.

3.3 Exhibit CB-1: Factors to bear in mind when out-licensing

Type of License	**Proprietary Licenses**	**Reciprocal Licenses with strong Copyleft**	**Reciprocal Licenses with weak Copyleft**	**Permissive Licenses**
Examples	EULA, etc.	GPL, AGPL, etc.	LGPL, etc.	MIT, BSD, Apache License, etc.
Degree of Rights granted to Licensees	Very low	High	High	High
Level of imposed Requirements	Very high	Medium	Low	Very low
Motivation of Licensor	• Ensure software only available for users with explicit permission • Ensure software stays closed above compatibility	• Public availability • Ensure attribution • Ensure software stays FLOSS above compatibility	• Public availability • Ensure attribution • Compromise between ensuring software stays FLOSS and compatibility	• Public availability • Ensure attribution • High degree of compatibility
Selection of new Draft	Drafted on individual basis	Recommended to choose existing OSI or FSF approved license	Recommended to choose existing OSI or FSF approved license	Recommended to choose existing OSI or FSF approved license
Signal Effects on Community	Not enabling a developer community	Strongly encouraging	Weakly encouraging	Weakly encouraging

Table 4: Factors to bear in mind when out-licensing

4 Teaching Note

4.1 Case Title

Getting Licensing Right

4.2 Case Synopsis

On the morning of October 6, 2014, Mr Holger Dyroff, co-founder of OwnCloud Inc., was on his way to an important meeting. While waiting at a red traffic light, he was outlining for himself the strategy he would shortly be presenting to the other two co-founders of OwnCloud. The meeting was with Mr Frank Karlitschek, who had initiated the OwnCloud project and was in charge of the development, and Mr Markus Rex, OwnCloud's chief executive officer (CEO). The aim of the gathering was to come to a final licensing decision regarding the IOS- client, which allows users to access their OwnCloud's file sync and share server instances from all mobile Apple devices. Dyroff's responsibilities at OwnCloud included the management and governance of intellectual property and licensing. All the three co-founders were convinced that these fields were of tremendous relevance for the success of OwnCloud. Up until this point, the IOS mobile client had been exclusively under a proprietary license, even though OwnCloud was based on open source. In order to be able to distribute it via Apple's App Store it was necessary to have the IOS client under a proprietary license, because Apple does not accept free/libre open source licensed programs. Unfortunately, having the IOS mobile client not open sourced meant two things: First, the OwnCloud executives had to discover that the IOS client made significantly fewer technical improvements when compared to OwnCloud's Android client. The Android client was open-sourced and the community was thus able to work with it. Second, critical remarks inside the community about the fact that the crowd was deprived of the code of the IOS client were becoming louder and was gaining resonance. Waiting at the traffic light, Dyroff was weighing up in his mind different courses of action and what the associated risks and chances would be. Should he advocate to give the IOS client an open source license? The undesired effects resulting from having a component not open-sourced were likely to diminish, but would OwnCloud Inc. then be losing revenue?

4.3 Teaching Objectives

This case was written for a course in software product management. Participants are normally full-time students with a major in computer science or information systems, either advanced in their bachelor or at the beginning of their master course of studies. Neither prior knowledge of product management nor about FLOSS licensing can be assumed.

Regarding the analytical dimension, participants will be directed to the issue but no solution will be provided. Instead, students will need to reach a decision on behalf of the decision-maker introduced in the case. Students therefore have to generate alternatives, and to evaluate the case with the help of decision criteria they need to identify. Finally, a decision has to be formulated.

On the conceptual dimension, this case helps to demonstrate the fundamental importance of licensing decisions for the business success of companies. Furthermore, it provides an introduction to FLOSS licenses and supports an understanding how licenses are linked to code components as well as the software product as a whole. Finally, the case contains an introduction to the single-vendor commercial open source business model.

In addition to presenting the immediate issue, the case also includes an introduction to the most essential theoretical concepts and to the context in which the issue occurred. Equipped with this information, students should be able to derive the criteria required to successfully master the analytical challenges of the case.

4.4 Immediate Issue

The immediate issue faced by the decision-maker faces is that of arriving at a licensing decision concerning one of the key components of the software suite.

4.5 Basic Issues

1. The concept of software licensing and how it is derived from intellectual property law.
2. Differences between FLOSS licenses and proprietary licenses.
3. Relation between code components and software licenses.
4. The interplay of FLOSS and proprietary licenses in the form of the dual licensing strategy in order to allow for a single-vendor commercial open source business model.

4.6 Suggested additional Reading

- Papers:
    - To facilitate a rapid basic understanding of FLOSS, the paper by Bonaccorsi and Rossi, *Why Open Source Software Can Succeed* (2003). This work also underlines the role of software licenses in this respect.
    - *The Single-Vendor Commercial Open Source Business Model,* by Dirk Riehle, outlines the business model for which this case acts as an example (2012).
    - *Dual Licensing,* by Michael Olson provides a good explanation of the helpful licensing strategy as a means of commercializing open source software (Olson, 2005).
- Books:
    - Because of its focus on the real-life issues of software practitioners, the book *Intellectual Property and Open Source* by Van Lindberg may be of interested to future product managers (2008).
    - *Understanding Open Source and Free Software Licensing,* by Andrew Laurent provides a detailed overview of the FLOSS licensing complexities and also comments on the single most important FLOSS licenses (2004).
- Web resources:
    - The website of the Open Source Initiative is highly recommended as a resource for the Open Source Definition, the overview of OSI-approved licenses, and the very helpful frequently asked questions (FAQ) section (https://opensource.org).

  - Also, the websites maintained by the Free Software Foundation are strongly recommended. Here, one can find detailed answers to questions about FLOSS licenses and a useful FAQ helping with the answers relating to the GPL (http://www.gnu.org/).
  - The Institute for Legal Questions on Free and Open Source Software (ifrOSS) is a German organization providing detailed information about FLOSS licenses together with a helpful FAQ in German-language FAQ (ifross.org).
- And, not to read but to watch:
  - Participants who are new to the topic of FLOSS may be interested to search the internet for the documentary *Revolution OS*, by J.T.S. Moore (2001). It provides a rich and first-hand overview of the history and importance of FLOSS by interviewing the initiators of the movement. Also the role of FLOSS licenses gets described.

4.7 Suggested Student Assignment

If you were in the position of Holger Dyroff, what licensing strategy for the IOS client would you recommend in the upcoming meeting. Why?

4.8 Case Analysis and Teaching Plan

4.8.1 Comprising Teaching Suggestion

The outlined approach assumes individual preparation and, wherever possible, prior discussion in small groups. In order to reach a decision, the participants first have to gain an understanding of the concepts and theories that are important when making a recommendation. Therefore, following a high-level introduction to the case, to OwnCloud and the student assignment, the most important aspects of the single concepts should be recapitulated in class. In addition to a common understanding of OwnCloud and software licensing the decision criteria should be discussed. Both will serve as basis for deriving decision alternatives and arriving at a final decision in the case assignment.

4.8.2 Recap

- ➢ Establishment of a common understanding of OwnCloud:
  - → Proposed assignment question: What is OwnCloud?
    - ◆ Supportive questions: Is OwnCloud a typical mainstream software company like Microsoft or Google? What makes it special? How many products has OwnCloud have?
    - ◆ Teaching Suggestion: Keywords the students associate with OwnCloud could be gathered and organized in the form of a mind map. Only information regarding the company, the community, and the product should be noted down at this time. The topics FLOSS and licensing should be left for later.
    - ◆ Goal: Structuring of the set-up of OwnCloud for a common understanding and explanation of terms.

- Outline of an answer: Figure 8 shows a mind map likely to emerge in the process.

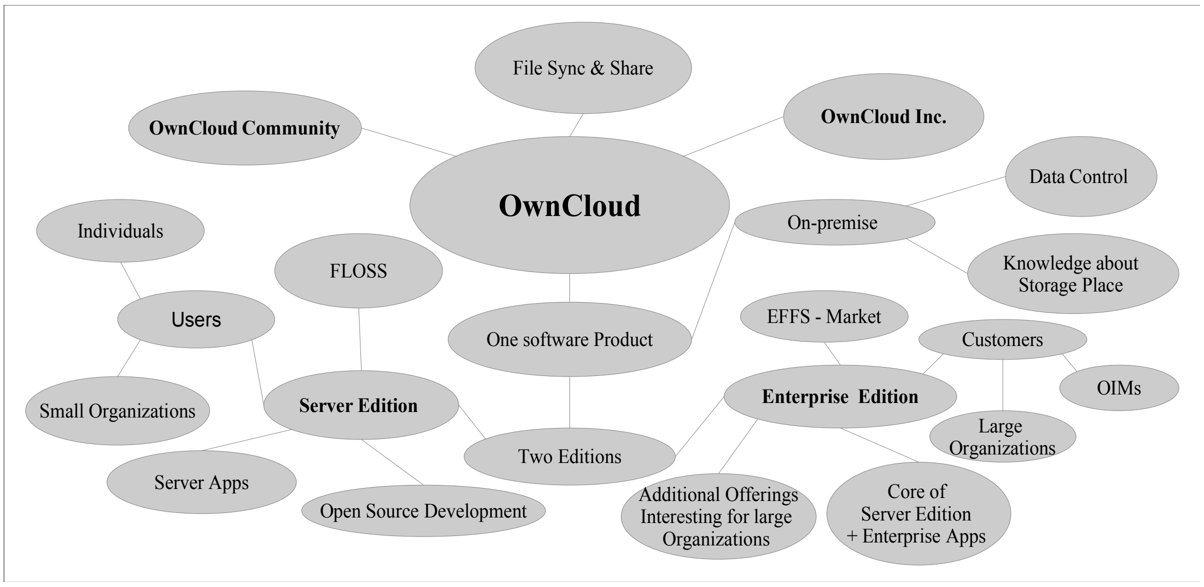

Figure 8: Possible Outcome of Brainstorming Session about OwnCloud

→ Proposed assignment question: What criteria defines the business model of OwnCloud? What kind of business model is it?

- Goal: Recap of the components of the OwnCloud business model that are most important to have in mind when choosing a licensing strategy.
- Outline of an answer: OwnCloud is a single-vendor commercial open source firm. Criteria gathered should be oriented around the business model canvas according to Osterwalder as outlined in the case. The value propositions around both different editions of the product, and the OwnCloud subscription model, are both considered to be especially important to name. At this point, it should also be clear that the Server Edition is mainly available for free. Aspects of FLOSS, or the dual licensing strategy, should not be discussed in detail at this point.

➢ Establishment of a common understanding of FLOSS:

→ Proposed assignment question: What distinguishes FLOSS from proprietary software?

- Outline of an answer: Here, it is important to name the fact that he source code has to be made available and that the software is free to be modified as well as to be shared — requirement of free redistribution.

→ Proposed assignment question: In which ways is FLOSS and the open source development model beneficial to OwnCloud?

- Goal: Establish an understanding of how FLOSS acts as a driver for the success of OwnCloud in order to grasp the motivation for mastering the associated licensing challenges.

- ◆ Outline of an answer: FLOSS helps at several levels. First, when the project was started the open source development model made it possible to engage a whole community willing to contribute to it. So the project quickly became a success without the need for high investment costs. Open source development, together with efficiency, also creates a high-quality, secure product. The huge user base of the successful open source project generates trust and significantly reduces marketing effort and expenses for OwnCloud Inc. Also, open source development facilitates the reuse of code from a massive pool of FLOSS software. Most important is the transparency of FLOSS that allows for evaluation of the source code. FLOSS is thus one key factor that enables OwnCloud to be trusted as an alternative to mainstream services for FSS and EFSS — most especially for all those who care about data control. FLOSS is the unique characteristic that sets OwnCloud apart from the competition.

➢ Establishment of a common understanding of FLOSS Software Licensing:

→ Proposed assignment question: Why are licenses required in software development?

- ◆ Outline of an answer: Under IPR, software is classified as literature and copyright law applies thus for its protection. In copyright law, all creative works are protected by default, i.e. no third party is allowed to redistribute them. Licenses are permissions granted by the copyright owner that enable others to legally copy and redistribute such creative works. Without licenses, the principles of FLOSS could not work.

→ Proposed assignment question: What are the difference between the goals of FLOSS and proprietary software licenses?

- ◆ Outline of an answer: Whereas proprietary software licenses ordinarily have the goal to limit usage rights, as well as to explicitly prohibit sharing of software in order to create and retain a basis for the earning of licensing revenue, FLOSS aims to freely share software with the community and, as such, to assign rights otherwise held — by default — exclusively by the copyright owner.

→ Proposed assignment question: What are the two main FLOSS licensing groups and how do they differ from each other? What is copyleft?

- ◆ Copyleft is a key attribute of so-called reciprocal licenses like the GPL that impose the condition on developers to use the same license for the outcome of any modification or combination of the software as the license that was originally attached. In this way, reciprocal licenses guarantee that software spawned on the basis of FLOSS remains FLOSS. Permissive FLOSS licenses, on the other hand, do not have the condition of copyleft and can even be incorporated into proprietary licensed products.

→ Proposed assignment question: What is the difference between the in-licensing and out-licensing of code components?

- ◆ Outline of an answer: Every code component needs licenses in order to be used by persons other than the copyright holder. In-licensing refers to the reuse of components in software projects. Before a component can be integrated the terms of the license attached to the component need to be evaluated, together with their impact on the software project. Out-licensing is the choice of the license the software project, or the single developed components of it, shall have. It is restricted by the freedom of choice left by the terms of the in-licensed components and the degree of intellectual property ownership of the component by the licensor.

- ➢ Licensing as a business-oriented strategy:
  - → Proposed assignment question: Why is the dual licensing strategy of such importance for OwnCloud?
    - ◆ Outline of an answer: Dual licensing allows multiple licenses to be placed on one component — provided the licensor holds the IPR required. This allows OwnCloud to license their product under FLOSS licenses on the hand and with proprietary licenses on the other. This again supports the single-vendor commercial open source business model, in which revenues are also made from subscription fees.
  - → Proposed assignment question: What is the main proprietary license used for the Enterprise Edition and why is it special?
    - ◆ Outline of an answer: The OwnCloud Commercial License has almost all the typical FLOSS attributes. Consumers given access to the source code and can make modifications to it, which means they can perform security audits and integrate the EFFS solution into their platform. For the last feature, it is also important that the Commercial License is not FLOSS because, in this way, the customers do not risk any intellectual property leakage due to possible copyleft conditions.
  - → Proposed assignment question: What is the in-licensing policy and the out-licensing policy of OwnCloud? Why?
    - ◆ Outline of an answer: Regarding in-licensing, permissive licenses only, or the LGPL at most, are used in order to avoid the risk of losing control over the choice of a dual licensing strategy for out-licensing — copyleft does not allow for proprietary licensing. The Server Edition is out-licensed only under strong copyleft FLOSS licenses. That makes it less attractive for proprietary organizations to choose this edition, as they cannot perform modifications or directly incorporate it into their software without fearing that they would have to open-source their code. The Enterprise Edition, on the other hand, is made proprietarily available in order to unlock special benefits for enterprise customers and to be able to charge fees for distribution.

4.8.3 The IOS Client Licensing Decision

- ➢ Key criteria of the decision:
  - → Proposed assignment question: What purpose is served by the mobile clients of the OwnCloud software suite?
    - ◆ Outline of an answer: The mobile clients are a convenient way for end-users to access the OwnCloud server. They also facilitate synchronization with the core

and the user's mobile devices. For OwnCloud they have a strategic relevance as a factor encouraging customers to choose the Enterprise Edition over the Server Edition. This is especially designed to target the customer segment of OEMs, which install OwnCloud on their servers in order to offer FSS as SaaS to their own customers. OEMs therefore seek to offer their customers clients branded with their own logo. The ability to own-brand the clients is a feature OwnCloud wishes to offer exclusively with the Enterprise Edition. OwnCloud makes an exclusive exception for the Android client that may also be branded when using the Server Edition.

→ Proposed assignment question: What special function does the IOS client have for OwnCloud?

- Outline of an answer: At the level of mobile clients, the IOS client has a special strategic function as an incentive for customers like OEMs to prefer the more expensive Enterprise Edition. Technically, it is possible to offer an Android client of the Server Edition with customer logo via the Google Play Store. So OEMs which use the gratis Server Edition can rebrand the client for Android. But this is not possible for the IOS client at the current setting as the IOS client is purely proprietarily licensed and users are not granted the right to modify it. Thus only the IOS client serves as a barrier meaning that not all Server Edition mobile clients can be branded by OwnCloud customers. They therefore have to choose a subscription package for the Enterprise Edition, which once again increases the business success of OwnCloud Inc.

→ Proposed assignment question: Which external criteria seem to be important for the IOS client?

- Outline of an answer: The fact that the Apple's App Store does not accept GPL licensed code may be of relevance to a possible solution. Even if the IOS client may be put under a FLOSS license — in accordance with the FLOSS licensing policies of OwnCloud — Server Edition users would not be able to distribute their modifications via the Apple App Store. This means that the IOS client retains its attribute as strategic barrier preventing customers from offering their clients to their employees or to end-users.

→ Proposed assignment question: Are there any negative effects of a closed sourced IOS client?

- Outline of an answer: The co-founders had to take account of two severely negative effects of keeping the IOS client under a proprietary license. First, the whole development process is closed and thus lacking in transparency for the community. As the community could contribute neither to the code nor to bug fixes, the IOS client made less development progress than the Android client. Second, some in the community frequently expressed their unhappiness with their exclusion from the source code of the client and questioned the whole FLOSS spirit of OwnCloud. The arguments were, on the one hand, painful for the FLOSS enthusiasts at OwnCloud to take on board, and on the other hand it was also not beneficial to their reputation.

➢ Outline of possible alternatives for the IOS licensing decision:

- → First, Dyroff could recommend to keep the IOS client under a proprietary license. A possible argument may be that the policy of Apple not to accept GPL licensed apps for redistribution at the App Store, may get revoked in the future. This would mean that, if deciding to open-source the IOS client, OwnCloud would have no barrier left to prevent customers dependent on branded sync clients from using the Server Edition. Further, the development of the IOS client would not necessarily become more efficient when open-sourcing its code. Android is the more important mobile operating system for the FLOSS community. In addition, the IOS client is programmed with a different technology than that of the Android client. Thus it may well be that no developer from the community would have a self-interest in contributing to the IOS client. The criticism from the community would prevail, yet despite being painful to the heart of a FLOSS enthusiast it may not have a big impact on the commercial success of OwnCloud — which always has a certain priority for a business opportunity, even for a FLOSS-driven one.
- → Instead, Dyroff could, as a second option, recommend changing the licensing of the IOS client from a proprietary to a FLOSS license. This may have positive effects on the development and most certainly satisfy the critics from inside the community. But switching the license to FLOSS could also have severe consequences for the business model of OwnCloud. In order to make the FLOSS license more attractive to potential Enterprise Customers, the co-founders could think about making an exception to their out-licensing policy and giving it a permissive FLOSS license. This would allow their customers to integrate it without restrictions. Choosing the GPL —due to the risks of open-sourcing their code — may, on the other hand, turn potential customers away. Offering the IOS client exclusively licensed under a GPL would result in OwnCloud not being possible no longer being able to offer their own client to users via the Apple App Store. So choosing a reciprocal license may not be recommended in this case. Unfortunately, by choosing a permissive license, the IOS code may be made open source, but there would be no guarantee that it would continue to be openly developed. Interested companies could in-license the code and keep derivatives of it behind closed doors once again.
- → Instead of focusing on one or the other licensing type, Dyroff could, as a third option, recommend to applying the dual licensing strategy to the IOS client, too. In this way, the IOS client could, on the one hand, remain under the Apple Store compatible priority license agreement. The proprietary license would be used for two things. Firstly, it would be the standard license for the IOS client offered within the Enterprise Edition, in similar fashion to the offering of the Android client. But, secondly, OwnCloud also could use the proprietary licensed IOS client to ensure that direct distribution of the IOS client via the Apple App Store prevails. All users downloading the client app from there would still have to agree to use a component under a proprietary license agreement. But this may not be so very disastrous after all, as OwnCloud would, on the other hand, at the same time offer the IOS client under a FLOSS license. In line with their strict out-licensing policy, the license chosen would have to be of the reciprocal family with a strong copyleft attribute — most likely the newest version of the GPL.

➢ Outline of a possible decision and the rationale behind it:

→ Comparing the alternatives by going back to the previously outlined decision criteria once again, it may be most appropriate to choose a dual out-licensing strategy, as drafted in the outline of the final option above. This strategy aligns best with the business model of OwnCloud Inc. and should carry no risk with regard to the overall business success of the company. It also gives the source code in most sustainable manner to the community and all interested persons, by choosing the GPL and its copyleft feature. This solution would also make the development process of the IOS client transparent and would include the offer to the community to also participate in its development. Also, it should speed up the development of the IOS client as new features are likely to be contributed by the community. Finally, the whole FSS software suite of OwnCloud, including all clients, would be open sourced and no longer the target of criticism from inside the community.

5 Case Difficulty Classification

This chapter aims to provide a classification of the case difficulty according to the *case difficulty cube* as introduced by Leenders, Maufette-Leenders, and Erskine in their book *writing cases* (2001). The case difficulty cube provides a model to evaluate how much time participants are expected to need to understand the material and to address the presented issue. The difficulty cube consists of three different dimensions whose levels of difficulty get evaluated on a scale from 1 to 3. A difficulty of degree one would signal that the dimension is rather simple to master whereas a difficulty of three would indicate a very high level of complexity.

	Case Difficulty Cube Dimensions		
	Analytical Difficulty	**Conceptual Difficulty**	**Presentation Difficulty**
Difficulty Degree	**2**	**2**	**2**

Table 5: Overview of expected Case Difficulty according to the Case Difficulty Cube Model of Leenders, Maufette-Leenders, and Erskine

The *analytical dimension* gives a rating of how complex it is to identify and to find an answer to the issue considered in the case. *Getting Licensing Right* is evaluated to represent an analytical difficulty of the second degree. The participants are directly pointed towards the issue for which they are to develop a solution, but the solution is not represented in the text. A degree of motivational challenge and operational freedom therefore remains for the case participants. In this case, the task is to present a licensing strategy for OwnCloud's IOS client.

The *conceptual dimension* deals with the number and complexity of theories participants need to have understood in order to be able to present a solution for the case. The following concepts have been identified as important for participants if they are able to solve the issue presented in the case:

- The product and business model of OwnCloud
- The concept of FLOSS
- The concept of software licensing and how it is derived from intellectual property law
- The interplay of FLOSS and proprietary licenses in the form of the dual licensing strategy in order to allow for a single-vendor commercial open source business model

A basic understanding of the following concepts is also considered to be helpful:

- Modular Programming
- Concept of Cloud Computing
- Open Source Development Model
- Motivation of FLOSS development and communities

Due to the many concepts the conceptual difficulty is considered to be second degree.

The *presentation dimension* evaluates to which degree key information from the text is presented and how precisely the information is delivered. The presentation difficulty of the case is rated to be second degree. Since the students attending the product management course cannot be expected to have substantial previous knowledge, the author made a special effort to include explanations. This was at the cost of a short case.

III References

Arlotta, C. (2014, July 8). Gartner Magic Quadrant for Enterprise File Synchronization and Sharing. Retrieved February 20, 2016, from http://talkincloud.com/cloud-computing-research/070814/gartner-magic-quadrant-enterprise-file-synchronization-and-sharing

Bonaccorsi, A., & Rossi, C. (2003). Why open source software can succeed [Web Blog Post]. *Research policy*, *32*(7), 1243-1258.

Contributions, B. (2010, May 26). More about the App Store GPL Enforcement. Retrieved January 16, 2016 from http://www.fsf.org/blogs/licensing/more-about-the-app-store-gpl-enforcement

Endsley, R. (2011, December 14). Former SUSE Exec Joins Open Source OwnCloud, Launches Commercial Entity. *CMS Wire*, Retrieved October 25, 2015, from http://www.cmswire.com/cms/document-management/former-suse-exec-joins-open-source-owncloud-launches-commercial-entity-013849.php

Evans, D. S., & Layne-Farrar, A. (2004). Software patents and open source: the battle over intellectual property rights. *Virginia Journal Of Law & Technology*, *9*, 10.

Fitzgerald, B. (2006). The transformation of open source software. *Mis Quarterly*, 587-598.

Graham, S., & Somaya, D. (2004, March). Complementary use of patents, copyrights and trademarks by software firms: evidence from litigation. In *DRUID Conference Paper. Copenhagen: Danish Research Unit for Industrial Dynamics*.

Helmreich, M., & Riehle, D. (2012). Geschäftsrisiken und Governance von Open Source in Softwareprodukten. *HMD Praxis Der Wirtschaftsinformatik, 49*(1), 17-25.

Idris, K. (2003). *Intellectual property: a power tool for economic growth* (Vol. 888). WIPO.

Jaeger, T. & Metzger, A. (2011). *Open Source Software: Rechtliche Rahmenbedingungen der Freien Software*. München: C.H. Beck Verlag.

Karlitschek, F. (2014, June 19). Why I Built OwnCloud and Made It Open Source.*Linux.com*, Retrieved November 10, 2015, from https://www.linux.com/news/enterprise/cloud-computing/777158-why-i-built-owncloud-on-open-source

Laurent, A. M. S. (2004). *Understanding open source and free software licensing*. Sebastopol, CA: O'Reilly Media.

Leenders, M. R., Mauffette-Leenders, L. A., & Erskine, J. A. (2001). *Writing cases*. Ivey Publishing, Richard Ivey School of Business.

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *The journal of industrial economics*, *50*(2), 197-234.

Lindberg, V. (2008). *Intellectual property and open source: a practical guide to protecting code*. Sebastopol, CA: O'Reilly Media.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.

Metz, C. (2015, August 20). Github's Top Coding Languages Show Open Source Has Won. *Wired*, Retrieved November 03, 2015, from http://www.wired.com/2015/08/github-data-shows-changing-software-landscape/

Microsoft Volume Licensing. (n.d.). Retrieved January 16, 2016 from https://www.microsoft.com/en-us/licensing/learn-more/brief-licensing-by-cores.aspx

Moore, J.T.S. (Producer & Director). (2001). Revolution OS [Motion Picture]. United States: Wonderview Productions

New IDC Worldwide File Synchronization and Sharing Forecast Shows Market Will Grow to $2.3 Billion by 2018. (2014, October 09). Retrieved October 25, 2015, from http://www.idc.com/getdoc.jsp?containerId=prUS25192614

Olson, M. (2005). Dual licensing. O*pen Sources 2.0*, 71-90. Sebastopol, CA: O'Reilly Media.

OwnCloud kann Zahl der Kunden verdoppeln und Umsatz verdreifachen [Web Blog Post]. (2015, September 22). Retrieved November 10, 2015, from https://owncloud.com/de/owncloud-kann-zahl-der-kunden-verdoppeln-und-umsatz-verdreifachen/

OwnCloud Logo [Image]. (n.d.). Retrieved February 19, 2016, from http://tsdr.uspto.gov/#caseNumber=85979290&caseType=SERIAL_NO&searchType=statusSearch

Raymond, E. S. (2001). *The Cathedral & the Bazaar: Musings on linux and open source by an accidental revolutionary*. Sebastopol, CA: O'Reilly Media.

Rex, M. (2016, January 26). OwnCloud grows over 100% in 2015 [Web Blog Post]. Retrieved January 26, 2016, from https://owncloud.com/owncloud-grows-100-2015/

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

Rosen, L. E. (2005). *Open source licensing: Software freedom and intellectual property law*. Upper Saddle River, NJ: Prentice Hall PTR.

Stallman, R. (n.d.). FLOSS and FOSS. Retrieved January 25, 2016 from http://www.gnu.org/philosophy/floss-and-foss.html

Stallman, R. (2001). The Danger of Software Patents [Speech]. Retrieved December 14, 2015 from http://www.gnu.org/philosophy/stallman-mec-india.html

Stallman, R. (2009). Viewpoint Why open source misses the point of free software. *Communications of the ACM*, *52*(6), 31-33.

The Open Source Definition. Retrieved November 28, 2015, from https://opensource.org/osd

Top 20 Open Source Licenses. (n.d.). Retrieved January 15, 2016, from https://www.blackducksoftware.com/resources/data/top-20-open-source-licenses

Välimäki, M. (2005). *The rise of open source licensing: a challenge to the use of intellectual property in the software industry*. Turre Publishing.

What is Copyleft. (n.d.). Retrieved January 15, 2016, from http://www.gnu.org/copyleft/copyleft.html

Williams, S. (2002). Free as in Freedom: Richard Stallmans Crusade for Free Software.