The 'Private-Collective' Innovation Model under Permissive Licensing: A Case Study of OpenNebula Open Source Software

A thesis submitted for the degree of Doctor of Philosophy

By

Hazar Y. Hmoud

March 2018

Abstract

This thesis aims for understanding how the 'private-collective' model of innovation works in permissive open source software.

This model encourages the private investments in the collective software as well as the sharing of those investments with the collective community. By following this model in permissive open source software, this thesis suggested that private actors would experience a collective action problem referred to as a 'business dilemma'. This dilemma is the difficult situation experienced by private actors who would be reaping rewards by sharing their private investments but also losing their competitive advantage because of free riders.

Theoretically, private actors would be discouraged from sharing their private investments with the collective due to the business dilemma. However, in some real cases, we do not observe this constraint; private actors are not trapped by the business dilemma. Instead, they end up innovating and contributing to permissive open source software.

As a result, this thesis would investigate and answer the research questions: 'How can the private actors invest and share in permissive open source software without experiencing a business dilemma?' and 'Why private actors choose to invest and share rather than to free ride in permissive open source software?'

Ostrom's theory of collective action is used as a lens for investigating the patterns of the private contributions and answering the research questions consequently. This theory suggests that people, even with the absence of formal regulations, can talk and share their local knowledge and experiences in order to collectively arrange the pattern of their actions and to extract themselves from collective action problems accordingly.

OpenNebula open source software was the case study analysed. Findings are based upon an in-depth qualitative analysis of a substantial dataset involving 7,017 emails, 3,482 development requests, 4 technical OpenNebula official documentations and thousands of source code commits.

Findings revealed that private actors are voluntary entering in an 'active communications' with other participants. Findings proposed that an 'active communications' act as a prerequisite for the active private contributions done by private actors.

Those private contributions are locked within the collective software in the form of 'collective complementarities' through a 'transformation process'. Several evidences proposed that this process worked to align the private interests of private actors with the collective interests of the software.

Moreover, findings revealed that a set of 'rules' are emerged by the private actors and other participants in order to structure the 'active communications' and the 'transformation process'. Several evidences proposed that these 'rules' worked to support the alignment between the private and the collective interests.

Accordingly, it is proposed that the alignment between the private and the collective interests (which is done through the 'active communications', 'transformation process' and is supported through a set of 'rules) encourages private actors to share and to link their private software with the collective. Without sharing, their private software would not be part of the 'collective' complementarities' that are used in the different industries.

This thesis makes novel contributions to the literature of open source. In particular, it extends the 'private-collective' model of innovation by introducing the 'collective complementarities' as a theoretical concept for locking the private benefit alongside with the collective one. It extends the open source literature by providing a better understanding of the collective software as a type of 'commons' that is exposed to private appropriation. In this way, it provides a set of arrangements that can bring the best of open source software that are sponsored by private actors.

This thesis also encourages firms to: (1) share an ongoing control over the development of the software with the vibrant community members in order to develop software that can be used across industries and (2) harness information technology practices in their organizations to better serve the private and the collective interests.

Dedication

To mum and dad …

Without you, this thesis would never have commenced. Without you, I could never imagine having reached this far. Your ongoing support and unconditional love lifted me up through hard times. You never lose your faith in me.

If a day passes by without me saying “Thank you”, a moment shall never pass by without you knowing that I am grateful.

I love you.

To my little angels, Salma and Sura…

We went along together through this journey. And this piece of work is definitely not only mine, it is ours. Having you next to me through this journey keeps me sane indeed.

Girls, it is the time for our favourite song, “We are the little graduate dressed in black, wearing a gown and a funny looking hat”.

Hurray! We did it.

I love you.

Acknowledgement

I am thankful for my PhD supervisors, Prof. George Kuk and Dr. Corentin Curchod, for their efforts and time given in order to critically advance my research and to develop my research skills. I have learned so much from you. This research would never have been accomplished without you.

Thank you Dr. Rodion Skovoroda for co-supervising my first year. My sincere thanks to professors and doctors at the Business School for their academic advice and feedback through taught modules and seminars. Thanks for all of my colleagues who read my work and provided me with their feedback and comments. I would also like to thank Ms Andrea Tomlinson for her excellent help and support and Ms Teresa Bee for her advice and helpful counselling.

My gratitude goes to the University of Jordan for the Management Information Systems Department scholarship through which my research was funded. My special recognition and thanks for Prof. Rifat Shannak without whom I would never have had this opportunity. Thank you Dr. Raed Mas'adeh for your ongoing support. I am grateful for to Dr. Hani Jaber for his technical support and encouragement.

Unconditional love and thanks to my brothers: Ammar, Anmar, Mohammed, and Saif, and the rest of my inner circle of family and friends.

I am thankful for my office mates and doctoral society members at the University of Nottingham Business School without whom lonely times and boring weekends would have been too much.

Publication

HMOUD, H., KUK, G. & CURCHOD, C. From Restrictive to Permissive Licensing: The Governance of Appropriation in Open Source Software Project. British Academy of Management, 2016 Newcastle, United Kingdom.

Table of Contents

Abstract 2

Dedication 4

Acknowledgement 5

Publication 6

Table of Contents 7

List of Tables 11

List of Figures 13

Chapter 1 Introduction 15

1.1 Overview 15

1.2 Research Rationale 15

1.3 Research Motivation and Novelty 17

1.4 Research Question and Objectives 19

1.5 Thesis Structure 22

Chapter 2 A Business Dilemma in Open Source Software 25

2.1 Overview 25

2.2 Free Software 26

2.3 Open Source Software 29

2.3.1 Participants: Individuals and Private Actors 31

2.3.2 Licensing in Open Source Software 35

2.4 The 'Private-Collective' Model of Innovation 40

2.4.1 Private Contributions are Crucial 40

2.4.2 Revealing Private Contributions is Crucial 41
2.5 The Business Dilemma 43
Chapter 3 Governance of the Commons 48
3.1 Overview 48
3.2 Collective Action Problem 48
3.3 Theory of Collective Action 50
3.4 The suitability of a Theory 52
Chapter 4 Research Methodology 55
4.1 Overview 55
4.3 Thesis positioning 58
4.4 Case Study Design 58
4.4.1 The selection of OpenNebula case study 60
Chapter 5 Data Collection and Analysis 65
5.1 Overview 65
5.2 Data Collection 65
5.2.1 The First Type of Data: The Mailing List .. 70
5.2.2 The Second Type of Data: The Requests .. 76
5.2.3 The Third Type of Data: The Commits 77
5.2.4 The Fourth Type of Data: The Documentation 78
5.3 Reflexivity 80
5.4 Quality of the research 85
5.5 Research Ethics 88
Chapter 6 The Private Contributions in OpenNebula 90
6.1 Overview 90

6.2 OpenNebula as open source software 91
6.2.1 Start-up Stage 92
6.2.2 Growth Stage 95
6.3 The Private Actors in OpenNebula 97
6.3.1 The registration process 98
6.3.2 Participants mapping 100
6.4 Contributions in OpenNebula 107
6.4.1 Communications on the mailing list 107
6.4.2 Requests additions in the development portal 114
6.4.3 Contributions towards Collective Complementarities 118
6.5 Conclusion 129
Chapter 7 The Collective Software 131
7.1 Overview 131
7.2 The Nature of requests 131
7.3 The Transformation Process 134
7.3.1 Validation of Contributions 136
7.3.2 Selection of Contributions 141
7.3.3 Development and Testing of Contributions 145
7.4 Impact of Transformation Process on Contributions 149
Chapter 8 Rules in OpenNebula 153
8.1 Overview 153
8.2 The Rules 153
8.2.1 "Fast tracking" rule 156

8.2.2 "Follow up" rule 157
8.2.3 "Networking" rule 161
8.2.4 "Core-private reciprocity" rule 162
8.3 Execution Practices..... 163
8.3.1 Execution of the "Fast Tracking" Rule. 167
8.3.2 Execution of the "Follow Up" Rule 169
8.3.3 Execution of the "Networking" Rule.... 171
8.3.4 Execution of the "Core-Private Reciprocity" Rule 172
8.4 Impact of Rules on Contributions 173
Chapter 9 Discussion 180
9.1 Research Review..... 180
9.2 Empirical Findings..... 182
9.3 Research Contribution 184
9.3.1 Theoretical Contributions..... 184
9.3.2 Practical Contribution 191
9.3 Limitations of the Research 195
9.4 Direction for Future Research..... 197
References 199
Appendix A: The Analysis of Contributions in OpenNebula 213
Appendix B: The Analysis of Software Complementarities..... 239
B.1 Categories identified in the development page 239
B.2 The technical components in OpenNebula . 244

List of Tables

Table 2.1 Cost vs Benefits for individuals and private actors in open source software 33
Table 4.1 Examples of firms in different industries using OpenNebula in their business 63
Table 5.1 Triangulation Types 68
Table 5.2 A sample of participants analysed from the mailing list 74
Table 5.3 Description of the third set of data collected in this research 79
Table 5.4 Quality measures in this research 85
Table 6.1 Classifying participants of OpenNebula into individuals and private actors 104
Table 6.2 Contributions made by participants in OpenNebula 114
Table 6.3 Contributions in the development portal of OpenNebula 117
Table 6.4 Core and Complements modules in OpenNebula 120
Table 6.5 The different technology options for virtualization driver (complements module) in OpenNebula 122
Table 7.1 Summary of percentage of core and complements requests added into OpenNebula releases 144
Table 7.2 Impact of the transformation process 151
Table 8.1 Rules in OpenNebula 155
Table 8.2 Firms who participated in the 'Fund a feature' programme 156
Table 8.3 Examples of "networking" rule in OpenNebula 161
Table 8.4 Examples of the "core-private reciprocity" rule in OpenNebula 163

Table 8.5 Execution practices of OpenNebula rules..... 165
Table 8.6 Summarizing results for rules and their execution practices in OpenNebula 166
Table 8.7 Examples of formal execution of the “fast tracking" rule..... 169
Table 9.1 Differences between open source software and other appropriated 'commons' 187
Table 9.2 Examples of the advantages and disadvantages of autonomous and sponsored open source software..... 190
Table 9.3 The active role of participants with arrangements (formal and informal rules) 194

Table A. 1 The analysis of the different categories about contributions performed by participants 213

Table B. 1 Technical description of the categories as identified from the analysis of the development page..... 240
Table B. 2 Types of Drivers in OpenNebula..... 248

List of Figures

Figure 1.1 Thesis structure aligned with thesis objectives 24
Figure 2.1 Graphical illustration of open source software under restrictive license 36
Figure 2.2 Graphical illustration of open source software under permissive license 38
Figure 2.3 Open source software as an innovation 39
Figure 2.4 The business dilemma in open source software 45
Figure 2.5 Adoption of open source software in different companies 47
Figure 5.1 Number of requests added over time 77
Figure 5.2 Reflexivity in this research 81
Figure 5.3 Reflexivity in this research 83
Figure 6.1 The different stages of OpenNebula 94
Figure 6.2 The development of OpenNebula Software in the growth stage 96
Figure 6.3 Screen for registering new participant to the mailing list in OpenNebula 99
Figure 6.4 Screen for registering new participant to the development portal in OpenNebula 99
Figure 6.5 The different participants in OpenNebula ... 101
Figure 6.6 Rivals in the technology industry 105
Figure 6.7 Rivals in the education industry 106
Figure 6.8 An example of a newly added request to the development page of OpenNebula 115
Figure 6.9 Number of requests added into the development portal of OpenNebula 116
Figure 6.10 Technical design for software complements in OpenNebula 119

Figure 6.11 The difference between OpenNebula and the traditional context 125
Figure 6.12 Number of software releases in OpenNebula 128
Figure 7.1 Number of 'software bugs' and 'software features' requests added to different OpenNebula releases 133
Figure 7.2 Transformation process in OpenNebula 135
Figure 7.3 Percentage of valid, duplicated and invalid requests in OpenNebula 139
Figure 7.4 Number of valid 'software bugs' and 'software features' requests added to OpenNebula releases 142
Figure 7.5 An example of the scope rule used in OpenNebula 145
Figure 7.6 Commits added to OpenNebula core and complements software 147
Figure 7.7 The addition of different complements through time in OpenNebula 148
Figure 8.1 A set of rules that structure the patterns in contributions of private actors in OpenNebula 154
Figure 8.2 Impact of rules in OpenNebula 176
Figure 8.3 The technical components of the jointly developed OpenNebula software 178

Figure A. 1 'Modify Available Documentation' Code 235
Figure A. 2 Report Software Bugs/ Features 236
Figure A. 3 Solve Software Bugs 237
Figure A. 4 Develop Software Features 238

Figure B. 1 Classification of the software components according to core and complements categories 251

Chapter 1 Introduction

1.1 Overview

In section 1.2, this introduction will present the rationale behind this thesis. Section 1.3 describes motivation and reasoning behind this thesis. The research question and research objectives are presented in section 1.4. Finally, the structure of this thesis is detailed in section 1.5.

1.2 Research Rationale

This thesis aims for understanding how von Hippel and von Krogh (2003) 'private-collective' model of innovation is working in open source software that is declared under permissive licensing.

To start with, it is important to highlight that open source software was a prominent example for the 'private-collective' model of innovation. In this model, it is explained that private investments are crucial to the development of the collective software.

In addition, it is explained that the best course of action that can be done by the private investors is to share rather than conceal their private investments with the collective. This is justified because of the multiple rewards that the private investors would be reaping from such sharing.

This innovation model seems to work well for open source software declared under restrictive licenses (General Public License (GPL), for example). As will be explained in sub-section 2.3.2, restrictive licenses require the sharing of all

further modifications done to the original software. Thus, the software would be non-rivalry developed through time.

However, in this thesis, it is suggested that this innovation model is creating a business dilemma in open source software declared under permissive license (such as BSD or Apache) as follows.

In permissive licensing, sharing of the source code and further modifications of that code is not compulsory. Rivals are allowed to copy and imitate the shared private investments without contributing back to the collective software. As a result, the competitive advantage for the private investors would be decreased as imitation is easy.

Therefore, private actors would be experiencing a business dilemma; they want to share their private investments in order to reap rewards but they also do not want to share their private investments in order to protect their competitive advantage from rivals.

Theoretically, this business dilemma would discourage private actors from sharing their investments. Practically, there are some cases where private actors found to increasingly contribute to open source software through the time despite the dilemma. Contradiction between theory prediction and practice is deemed worth tacking.

Therefore, it is proposed that Ostrom's evolutionary theory of collective action would be a suitable theory for this thesis. This is because this theory suggests that participants can extract themselves from dilemmas even with the absence

of a formal authority. Thus, this theory would help in describing and explaining what happens with private actors and how they are encouraged to invest without creating the business dilemma in open source software that is declared under permissive licensing (where obligations against imitation are not introduced).

Based on the previous discussions, informed by Ostrom (1990) evolutionary theory of collective action, this thesis aims for understanding how von Hippel and von Krogh (2003) 'private-collective' model of innovation is working in open source software declared under permissive licensing by answering the research questions explained in section 1.4.

1.3 Research Motivation and Novelty

Open source software (OSS) is a computer program for which the source code is publicly available to be shared, used, modified, and distributed under an open source software license: either restrictive or permissive license (Raymond, 1998; O'Mahony, 2003; deLaat, 2007).

This thesis aims for understanding how von Hippel and von Krogh (2003) 'private-collective' model of innovation is working in open source software declared under permissive licensing.

This thesis is motivated by the different requests in the literature of open source where researchers emphasize that the role of firms and their private investments in open source software is not salient in the literature (Stol and Babar, 2009),

and a high concentration of research is needed (Markus, 2007a; Crowston et al., 2012).

The novelty of this thesis is identified from different dimensions.

First, this thesis is different in its settings.

While researchers focused on studying open source software under restrictive licensing, this thesis chooses to analyse open source software under permissive licensing.

Second, this thesis is different in its aspiration.

While other researches focused on analysing dilemmas that are experienced by hobbyists and individual actors (for example, O'Mahony, 2003, Baldwin and Clark, 2006), this thesis focuses on analysing a business dilemma that is experienced by private actors.

Private actors are crucial participants because they found to move open source software strategically and technically in the market (Casadesus-Masanell and Llanes, 2011). Their contributions are encouraged through von Hippel and von Krogh (2003) 'private-collective model of innovation'. Therefore, unpacking a collective action problem that they experience in open source software is deemed crucial. While high intensity of research was neglecting open source software in the context of organisations and private actors (Stol and Babar, 2009, Hauge et al., 2010), the novelty of this thesis is more salient.

Third, new domains of unravelled knowledge would be revealed when understanding the contradiction between theory and practice.

Theoretically, the suggested business dilemma would discourage private actors from investing and sharing their proprietary software to the collective. However, in practice, there are some cases where private actors are investing and sharing their private investments rather than concealing because of free riders. For example, statistics show that 78 per cent of companies invest in open source software in order to run their business, and 47 per cent of these companies disclose their private source code as open source software under permissive license especially Apache v2.0 (BlackDuckSoftware, 2015). Another example, OpenNebula, the case study analysed in this thesis, is an open source software with a large amount of commercial investment in different industrial sectors i.e. more than 180 firms from 13 industries are commercially investing in OpenNebula and .CITE

<EndNote><Cite><Author>Finlay</Author><Year>2002</Year></Cite></EndNote>Therefore, analysing the private collective mode of innovation in this case is novel as it will provide a logical explanation for the misalignment between theoretical suggestions and practice. And new domains unravel knowledge of would be revealed.

1.4 Research Question and Objectives

Permissive open source is an open source license that simultaneously reinforces the business dilemma experiences

by private actors. Theoretically, the business dilemma would discourage private actors to invest and share their investments in open source software. However, in some real cases, we do not observe this constrain. Instead, private actors end to innovate and contribute within their open source community.

Thus, it is required to investigate and answer the following research questions:

- 'How can the private actors invest and share in permissive open source software without experiencing a business dilemma?'
- 'Why private actors choose to invest and share rather than to free ride in permissive open source software?'

Ostrom (1990) evolutionary theory of collective action is suggested as a theoretical lens for answering the research questions. This theory suggests that people are rationale. They can talk and use their local knowledge and experiences in order to arrange their collective actions even without formal regulations. They would arrange their actions in order to extract themselves from collective action problems.

The research purposes to achieve the following objectives:

Objective 1: To review the literature of open source software and to contextualise this thesis within the proper literature of open source software.

Objective 2: To construct a research methodology, from an epistemological perspective, in order to answer the proposed research questions.

Objective 3: To supply a detailed description of the case study; recognise participants who are involved with developing open source software, identify their contributions and the outcome resulted from their contributions.

This step is crucial in order to identify the private investments from private actors and their propensity to be contributed and shared rather than 'hijacked' by others.

Objective 4: To analyse how the private investments contributed to the development of the collective software.

This objective is necessary in order to understand patterns of the private contributions towards the collective software. Accordingly, this would achieve a better understanding of how private actors are encouraged to invest and share their investments.

Objective 5: To analyse how patterns of the private contributions are followed by the private actors in open source software.

This objective is necessary in order to understand the structure that organizes patterns of the private contributions in open source software. Accordingly, this would achieve a better understanding of why private actors are encouraged to invest and share their investments.

Objective 6: To discuss the findings in terms of theoretical and practical contributions. In addition, provide insights for future research directions.

1.5 Thesis Structure

This thesis is structured around nine different chapters, as follows:

Chapter one (this chapter) provides an overview of the thesis.

It presents the reasoning behind the research and alludes to governance literature in order to contextualize the thesis in relation to existing literature. This chapter also describes the research question, objectives, and structure.

Chapter two focuses on explaining the business dilemma experienced by private actors in open source software under permissive licensing.

Theoretically, the business dilemma would discourage private actors to invest and share their investments in open source software. However, in reality, we do not observe this constraint. Instead, private actors end to innovate and contribute within their open source community.

Chapter three shows that the business dilemma in open source software is solved through choosing appropriate governance. This chapter also explains the suitability of the theoretical lens of (Ostrom, 1990) theory of collective action. I propose that governance of open source software should follow the conventional theory of collective action (Ostrom, 1990).

According to this theory, participants cannot be trapped in collective action problems. They are able to extract themselves from these problems.

Chapter four explains the philosophical standpoint of this thesis as interpretive, qualitative research. This chapter presents OpenNebula as the case study to be analysed in this thesis.

Chapter five discussed the types of data collected and how this was analysed. In addition, it elaborates on the issues of reflexivity and quality measures such as credibility, plausibility, and transferability in this research. A short reflection on research ethics will also be presented.

Chapter six is the first empirical chapter. It explains that private contributions by private actors are supporting the development of OpenNebula software in the form of collective complementarities. In addition, this chapter proposes that the active communications constitute a prerequisite to the active contributions of private actors to the software.

Chapter seven is the second empirical chapter of this thesis. It explains how private contributions contribute to the development of the collective software. It theorizes how a transformation process is used to transfer contributions into collective software. In addition, this chapter proposes that the transformation process, through focusing the attention of OpenNebula participants, encourages private actors to reveal their private contributions to the collective.

Chapter eight, the third empirical chapter, explains rules that govern the private contributions and the development of OpenNebula software. It theorizes the different ways for executing the different rules that exist between participants in OpenNebula. In addition, this chapter proposes that rules are encouraging private actors to contribute their private contribution because rules are supporting private actors in inducing, verifying, legitimizing and adjusting their private contributions in open source software. Accordingly, Private actors, through these rules, seem to work as a collective rather than worrying about rivalry.

Chapter nine discusses the findings.

Its purpose is to analyse the theoretical background and discuss key discoveries, as well as put forward practical contributions and any implications of the findings. This chapter also identifies the limitations of the research and suggests directions for future research.

The structure of the thesis can be coherently juxtaposed to the four research objectives presented above, as shown in figure 1.1.

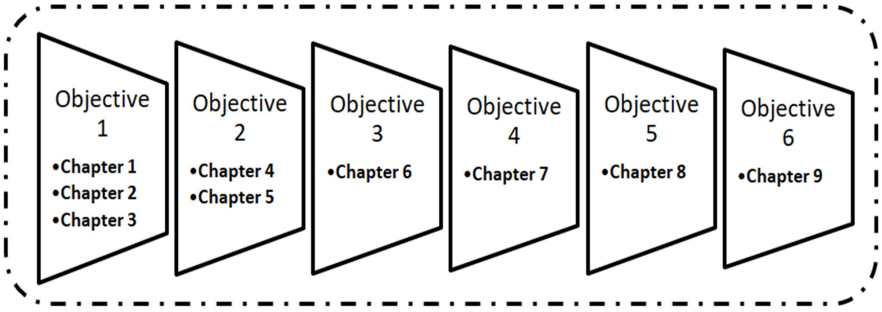

Figure 1.1 Thesis structure aligned with thesis objectives

Chapter 2 A Business Dilemma in Open Source Software

2.1 Overview

This chapter represents the literature review for this thesis. Literature was reviewed using suitable search keywords in 'Web of Knowledge' and 'Google Scholar' databases. Understanding the open source literature and its developments through time are used in this chapter to explain the business dilemma experienced by private actors in permissive open source software.

Free software and open source software have been used as interchangeable terminologies. However, Sections 2.2 and 2.3 explain the differences between free software and open source software and emphasise that open source software would be the focus of this thesis.

Section 2.4 explains open source software as a prominent example of the 'private-collective' model of innovation. According to this model, section 2.5 emphasises that private actors are crucial participants in open source software and solving their business dilemma is deemed essential.

2.2 Free Software

Free software is an initiative suggested by Stallman (1985a), this initiative respects the freedom of software developers by allowing them to share, modify, copy, and re-distribute the source code[1].

Throughout history, source code programming was largely conducted in both corporate laboratories and within academia. Source code was shared between software developers so that it could be put into practice and further developed.

In the 1950s, the US academy institution Massachusetts Institute of Technology (MIT) was the first incubator for the tradition of sharing source code between software developers. Richard Stallman experienced this when he joined the artificial intelligence (AI) lab in this institution. The environment was fruitful in the sense that software developers were willing to share the source code of software with any other software developer (in the same or different university as well as corporate laboratory) that requested it and, in turn, were not hesitant to ask for any piece of source code that they wished to understand, change, and reuse.

Software developers agreed to share without any regulations or rules in place with regards to exchanging source code. Software developers at that stage were called hackers; they were the 'heroes of the computer revolution' and their interactions were called Hacker Ethics (Zhu, 2011).

[1] The 'source code' part of the software is a collection of instructions written by software developers or programmers.

In 1976, Bill Gates (in the capacity of General Partner of Microsoft) issued an open letter accusing the minimal incentives given to software developers for their programming efforts due to this sharing spirit.

Gates angrily proposed further payment for developing Microsoft's interpretation of the Altair microcomputer after he discovered that time spent developing Altair was worth less than $2 an hour. He explained that this minimal incentive was caused by sharing spirit, as software developers shared source code with others who would steal the work.

Accordingly, Gates (1976) differentiated between the two types of code for any software:

(a) Source code is the set of instructions written by software developers. These instructions are solely understood and written by software developers but cannot be interpreted by computers. The source code is considered the intellectual property of the software developer who develops it.

(b) Binary code is the workable version of the source code. The source code is passed through a special program called a 'compiler', a program that can convert the source code into a collection of 1s and 0s (binary code) so computers can understand and execute the code. Binary code is useful in order to assist end users using the software, without them being required to understand it.

This differentiation between source and binary code caused a market shift; it encouraged the distribution of binary code (but not source code) because it benefitted both the end user and commercial firms. On the one hand, end users would save a lot of storage space when not compiling source code for software. On the other hand, commercial firms would maintain the right to protect a valuable intellectual asset, in terms of source code, from being imitated by competitors (Zhu, 2011).

As a result, software now falls under the Copyright Act in both the US and the UK; this was introduced in 1980 and 1985, respectively.

Richard Stallman (a software developer in the Artificial Intelligence lab in MIT) was extremely hostile towards proprietary rights given to software. He considered proprietary software as a social problem because it was against the liberty of programming and he suggested to 'stop using it and move to free software' (Stallman, 1985b).

Accordingly, Stallman started working on creating the GNU software, designed GNU General Public License (GPL), and founded the Free Software Foundation (FSF) in 1985. GNU software is a collection of software for managing different hardware and software components in a computer. It was developed by Richard Stallman and is comprised of free software that is declared under the GPL license.

The GNU GPL license is the first license designed in order to ensure the freedom of software (a detailed discussion about this license will explained in sub-section 2.3.2).

Under a GNU GPL license, it is compulsory that source code and further modifications of that code are shared. This license was designed in order to support the main social belief of the Free Software Foundation: sharing is acceptable and not sharing is unacceptable. Sharing is acceptable because it supports the liberty of software developers in sharing and developing the software. The GPL license is common and is currently being used by other types of software such as open source software.

Of course, free software was not welcomed by commercial firms who protected the source code of their software packages with private licenses. They considered source code as part of firms' intellectual assets that needed to be protected by the terms of software copyright.

Since source code can be reverse engineered by a company's competitors, revealing it online can be hazardous. Competitors can rewrite the source code and mimic a company's products.

2.3 Open Source Software

In 1998, Eric Raymond, one of the followers of the free software initiative, decided to break away from free software toward an increasingly open type of software: open source software.

Basically, Raymond valued the idea of sharing the software between developers but also argued that sharing software is not ultimately attractive for commercial firms. Firms are not willing to share their software because the software is

part of their intellectual property and competitive advantage. Therefore, Raymond initially emphasised that free software, while supporting the liberty of software developers, is not supporting businesses and commercial firms.

Therefore, Raymond declared open source software as an initiative that (a) supports the 'business use' of (b) a freely distributed source code. He said that software that is developed internally inside firms will be having a lower quality when compared to a software developed through sharing (Raymond, 1999a).

In addition, he emphasised that sharing source code can support failed proprietary software packages. He articulates that what are thought to be failed systems such as 'Linux' software can be brought back to life through sharing source code.

Accordingly, Open source software has been identified as a type of innovation in which the development and the usage of the software is delivered for, and developed by, users (von Hippel, 2001). Users are sharing the software on the internet so that everyone who has an interest in its development can participate.

2.3.1 Participants: Individuals and Private Actors

Participants in open source software are volunteers around the world who use software technologies to collaborate and develop source code. they can be individual participants and/or private actors such as firms (Lerner and Tirole, 2001).

The economic model suggested comparisons between costs and benefits in order to understand participant's motivations to participate in open source software. When the benefits of participation are greater than the cost, participants will contribute to open source software they are interested in (Lerner and Tirole, 2002). Some of the benefits and costs identified in the literature for both types of participants are summarised in table 2.2.

Individuals participate in order to fulfil their intrinsic and extrinsic motivations (von Krogh et al., 2012). For example, Hars and Ou (2001) found that individual participants participated in OSS projects in order to encourage altruism and to be identified by the projects' communities.

However, private actors participate in open source software in order to gain a business benefit (von Hippel and von Krogh, 2003) and fulfil their economic motivation (Bonaccorsi and Rossi Lamastra, 2003). Economic motivation for private actors is to either increase revenue or reduce costs (von Hippel and von Krogh, 2003).

According to Dahlander and Magnusson (2008), private actors (including firms) escalate their profits by selling complementary services and products and packaging their

open source software. Thus, firms appropriate open source software in order to increase their return (Dahlander, 2005). Appropriation in this sense means ‘capturing return from an innovation’ (ibid).

RedHat, a powerful open source firm, for example, increase their profits by selling services such as training and consulting. These services are not efficiently provided by the open source communities and are called ‘Living Symbiotically’ (Lerner et al., 2006). On the other hand, Microsoft is a firm that delivers a wide range of applications and services under the proprietary software license; it was one of the opponents of the open source social movement (von Krogh and Spaeth, 2007).

Table 2.1 Cost vs Benefits for individuals and private actors in open source software

Cost for Individuals	**Literature Example**
Opportunity cost	Lerner and Tirole (2002) define this cost as the lost time incurred by participants when they choose to develop source code for software.
Learning cost	Researchers such as Lerner and Tirole (2002) and Waring and Maddocks (2005) define learning cost as the time and energy expended when studying and comprehending source code.
Cost for Private Actors	**Literature Example**
Cost of Diffusion	von Hippel (2001) suggests that commercial actors such as firms may lose the cost of diffusing the source code with the public.
Loss of Proprietary Right	Different researchers such as von Hippel (2001) and von Hippel and von Krogh (2003) explain that commercial actors such as firms have a cost of publicly revealing and sharing the source code with competitors.
Benefits for Individuals	**Literature Example**
Gift culture	Different researchers such as Raymond (1999a), Zeitlyn (2003), and Ghosh (2005) define the gift culture as appraising the solidarity for giving and sharing behaviour between software developers.
Learning	According to Lattemann and Stieglitz (2005) and Shah (2006), hobbyists can benefit from open source software by understanding and developing source code.
Reciprocity	Lakhani and Wolf (2005) suggest that open source software encourages cooperation between participants of different technical skills. This is enables the development of fruitful open source software.
Enjoyment	Researchers including (Hemetsberger, 2002) and (Hertel et al., 2003) believe that hobbyists are satisfied because they are entertained by the process.

Reputation	By participating, hobbyists are able to build a positive reputation and earn recognition amongst their peers (e.g., Hars and Ou, 2001).
Career signalling	Ghosh (2005) and Roberts et al. (2006) found that participating in open source leads to better employment opportunities for participants.
Benefits for Private Actors	**Literature Example**
Cost reduction	Firms can develop a source code that is cheaper compared to the same source code that is developed under proprietary licenses (e.g., Hecker, 1999).
Dual licensing	Comino and Manenti (2011) identified dual licensing as providing two copies of the same source code: one copy under open source license to be used by individuals, the other under proprietary license to be used commercially by firms. 'Dual licensing' is crucial for gaining valuable improvements from the open source community; the firm is unable to produce this internally using the intellectual capacity of their internal employees.
Sale of complementary services or products	Gruber and Henkel (2006) provide cases in which firms benefit from open source software by providing technical or hardware related training to implement open source software.

2.3.2 Licensing in Open Source Software

Participants in open source software share their contributions with everyone. However, they are not willing to 'forfeit' these contributions (O'Mahony, 2003). Thus, open source software was declared under open source licenses. Every participant in open source software needs to comply with the requirements of the license associated with the software. There are different types of open source licenses, each of which has its own terms and conditions.

These licenses and how they are represented within open source literature will now be discussed.

In the early stages of open source, the GNU General Public License (GPL) was used for open source software. This license requires the sharing of the original software source code, and any further modifications and amendments, on the internet. Moreover, in this license, changing the terms and conditions is not acceptable.

The GNU GPL license is an open source license that is concerned with the freedom of the software; it ensures that everything related to the source code is given back and shared.

Over time, additional open source licenses were declared based on the feedback and requirements of the community, commercial firms, and academic parties.

With the introduction of additional licenses for open source software, researchers began comparing these licenses. Researchers agreed that the majority of open source licenses

are one of two types: ‘restrictive’ or ‘permissive’ (For example Colazo et al., 2005, Lerner and Tirole, 2005b, Sen et al., 2011).

Certain licenses are defined as ‘restrictive’ as they protect the software’s freedom; the source code and any modifications are revealed in full. GPL is an example of a restrictive open source license.

‘Permissive’ licenses, however, privatise the source code and its modifications, unless other participants have been involved in its development. An example of a permissive open source license is the BSD and Apache open source license.

Restrictive Licenses

In restrictive licenses, participants can copy the original source code, modify it, amend it, and combine it with another source code with the same open source license. By law, the work must be redistributed on the internet; the new source code cannot be used privately (see figure 2.1).

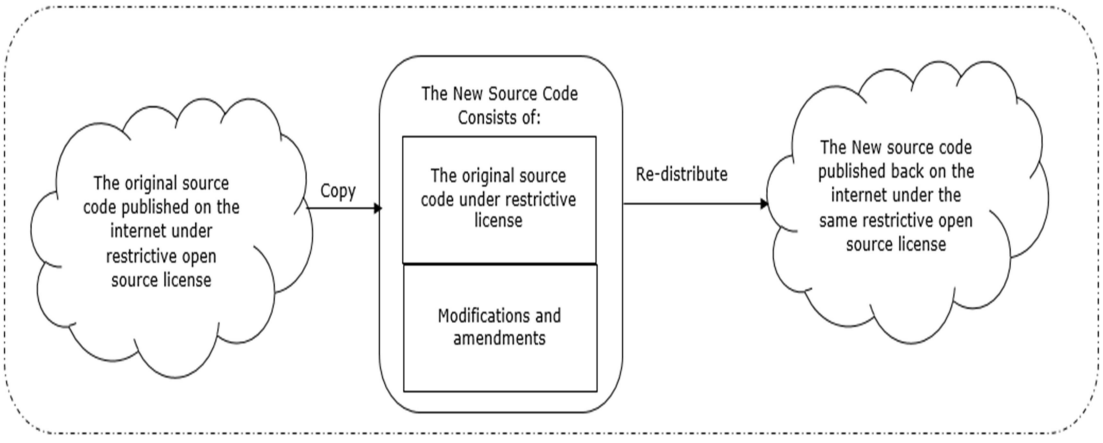

Figure 2.1 Graphical illustration of open source software under restrictive license

A Restrictive open source license is a type of open source license that requires sharing of the original source code and any modifications and amendments. Moreover, this license prohibits the re-licensing of the software to any license other than the original license declared.

Statistics show that restrictive open source licenses were initially used with open source software. According to the Black Duck Open Source Resource Centre, 70 per cent of open source projects were declared under a GPL license in June 2008 (Aslett, 2011).

Permissive Licenses

A GPL license focuses on the reciprocity of source code between software developers. However, its main problem is that it is hard in some cases to do further development for the open source software (Amo, 2007). This is true because, according to the requirements of restrictive licensing, any modifications to be done to the original source code must be declared under the GPL license. Therefore, in some cases, a source code under a different license cannot be combined with the original source code. For example, source code developed by commercial firms that is declared under a proprietary license cannot be combined with source code declared under a GPL license.

Any further development regarding the open source software will then cease.

Additional licenses that overcome these issues have fortunately been developed, such as BSD and Apache. These licenses are referred to as permissive licenses.

Permissive open source licenses allow the combination of different source codes, of different licenses, within open source software. As explained in figure 2.2, permissive licenses neither require modifications of the source code to be shared, nor the protection of the software from re-licensing to any other open source or proprietary license.

The only requirement is that participants and contributors who have assisted in developing the source code are acknowledged. The new source code can be used for private purposes.

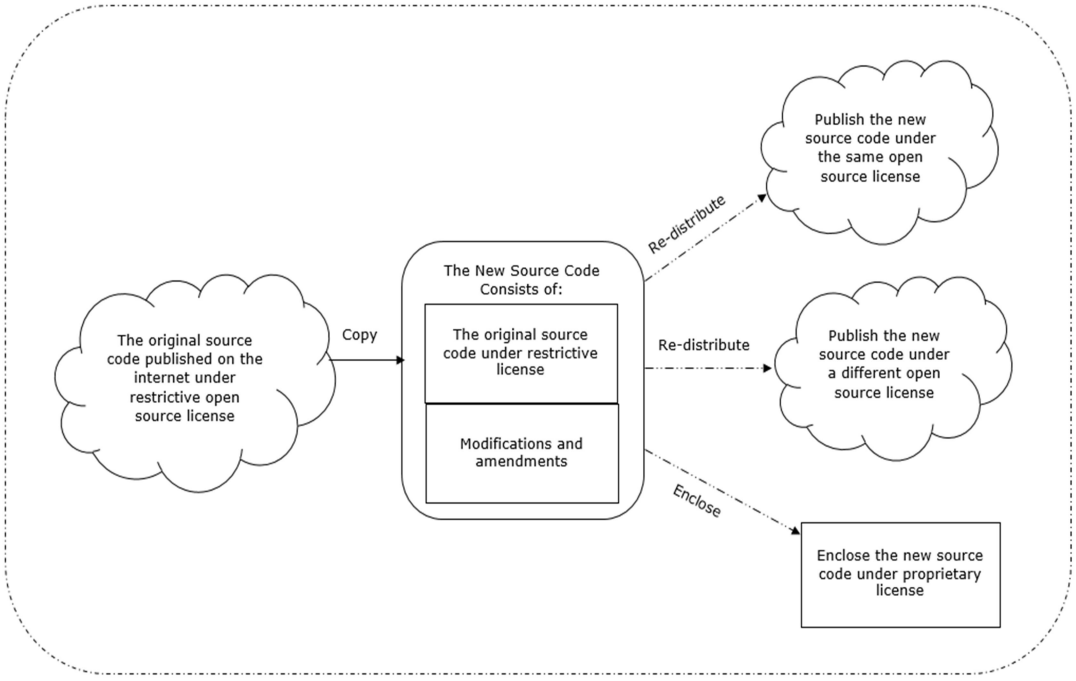

Figure 2.2 Graphical illustration of open source software under permissive license

A Permissive open source license is a type of open source license that does not require the sharing of the original source code and any modifications and amendments. Moreover, this license accepts the re-licensing of the software to any other open source or proprietary license other than the original license declared.

Permissive licenses may prompt businesses in different industries to utilise the software. Commercial firms in different industries are commercially investing and exploiting open source software.

As summarised in figure 2.3, open source software as an innovation consists of freely revealed software that is declared under an open source license and is developed by interested individuals and private actors.

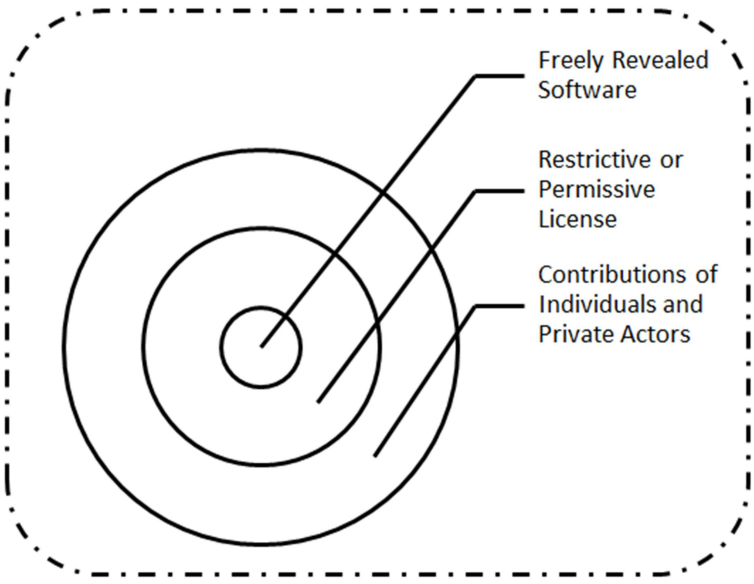

Figure 2.3 Open source software as an innovation

2.4 The 'Private-Collective' Model of Innovation

As discussed earlier, open source software is a type of innovation (von Hippel, 2001). And Open source software is the prominent example for the 'private-collective' model of innovation suggested by von Hippel and von Krogh (2003). von Hippel and von Krogh (2003) show that open source software is an innovation where private actors are considered crucial participants because they can invest their private resources to develop collective software. In addition, they can still reap private rewards when they freely reveal (share) their private investments with the collective.

2.4.1 Private Contributions are Crucial

It is worth mentioning that open source software is software in a source code format, and this source code is an unfinished product. This means that downloading the software is not enough to immediately start working on it. One needs to rewrite part of the source code in order to configure it properly to work within a computer network. After that, one needs to make sure that the source code is free of source code errors and compile the source code. Such a long process requires sophisticated users such as software developers and programmers. Not all users have the proper skills to understand and deal with different programming languages, software errors, and compilation procedures.

Accordingly, private actors such as firms who have professional developers and IT personnel that would copy the

source code, modify it, and combine it with other source codes in order to aid in preparing the software for end users and for business benefits (Sen et al., 2008). Of course, they participate in order to increases the competitive advantage for commercial actors by lowering costs, enhancing revenue through the creation of complementary assets (Alexy, 2008), entering and sustaining the market, or/and increasing internal cognitive human capital through interactions with open source software community members (Von Hippel, 1994).

Moreover, private actors rely on external sources, such as open source software, in order to improve private investment in the market (Von Hippel, 2007). Therefore, they are commercially investing in open source software as part of their open innovation. Daniel et al. (2018) provided different strategies that are crucial for firms in order to integrate external developer communities and internal knowledge resources.

In addition, private actors are aware of the economic and technical benefits of participating in open source software (Bonaccorsi and Rossi, 2003; von Hippel and von Krogh, 2003). Therefore, they are crucial participants who can develop open source software strategically and technically in the market (Casadesus-Masanell and Llanes, 2011).

2.4.2 Revealing Private Contributions is Crucial

von Hippel and von Krogh (2003) articulate that revealing the private investments of the software is better than just hijacking the software. Dahlander and Magnusson (2008) show

that firms can benefit from participating in open source software (as part of their open innovation) only when they follow three critical steps: access, align, and assimilate. Firms need to access open source communities, align the software with their private business through commercially investing in the software, and contribute to the community in order to spread their work throughout the open source community.

IBM, for example, are renowned for investing in open source software and revealing their source code so that their commercial products are sustained (Lerner and Tirole, 2005a).

Moreover, freely revealing the private investments is crucial for private actors. Private actors are innovators of the software and revealing the innovation would increase the positive externality for the software; the software will be a dominant design for customers (Von Hippel and Von Krogh, 2006). In addition, revealing the software allows for customer feedback which guarantees that the software is continually developed and enhanced.

Nagle (2018) found that firms would increase their productive value through sharing with and learning from the collective community rather than free riding them.

Furthermore, freely revealing the private investments is crucial for the development of open source software. The software becomes more attractive and can be supported over time. Commercial investment and contribution will sustain this innovation in the market (Dahlander and Magnusson, 2005). Without revealing contributions, the future stream of software

will be unavailable for the community and the software will be useless. West and O'Mahony (2005a) suggest that affecting the availability of the future stream of software represents ‘tragedy of the commons’ (Hardin, 1977) in open source software. Thus, commercial actors who invest in open source software need to reveal part or all of their commercial investments to the public for their own private benefit (von Hippel and von Krogh, 2003) and for the continual development of the software (West and O'Mahony, 2005a).

2.5 The Business Dilemma

According to von Hippel and von Krogh (2003) ‘private-collective’ model of innovation, private actors are investing their private resources in order to develop software. Hauge et al. (2010) elaborated that private investments by private actors can by through:

(1) Developing existing open source software. For example, many firms have paid for their employees to participate in the development of open source software in order to develop their programming skills (Hertel et al., 2003; Lakhani and Wolf, 2005).

(2) Integrating proprietary software with open source software; many firms sell their proprietary complements that are integrated with open source software (Gruber and Henkel, 2006). For example, OpenStack is an open source software under a permissive licensing agreement; Apache v2.0. Different commercial actors have been commercially investing the software. Although the software of OpenStack is shared in the

internet (non-excludable), but it can be used according to the product provided by these commercial actors only. Llorente (2014) discussed that OpenStack is a vendor-lock software and one cannot gain its value unless implementing the products provided by the vendor.

(3) Disclosing proprietary software as open source software. Some firms internally developed a software and then freely reveal the software to the collective as open source software. OpenNebula is an example of software that is developed internally by two developers and after a while, they decided to disclose the whole software to the collective as open source software under Apache v2.0 permissive license.

According to von Hippel and von Krogh (2003) 'private-collective' model of innovation, sharing of the private investment with the collective is the best course of action that can be done by private actors.

However, their shared investments may involve sharing of private knowledge that can be appropriated by anyone because open source software is non-excludable. This would mean that the 'appropriability regime' (knowledge in terms of know-how) would be available and, accordingly, imitating innovation would be uncomplicated (Teece, 1986). As a result, benefits would be shared by both the innovator of the software and imitator (would become rivals).

Accordingly, this thesis suggests that private actors would be facing a 'business dilemma' (as described in figure 2.4) when sharing their private contributions with the

collective. Private actors would be trapped in a situation where contributing or not is a hard decision. If they decide to contribute their private knowledge to the collective, they would be reaping benefits but also lowering their competitive advantage. And if they decide not to contribute their private knowledge with the collective, they would be protecting their competitive advantage but also losing rewardable benefits from open source software.

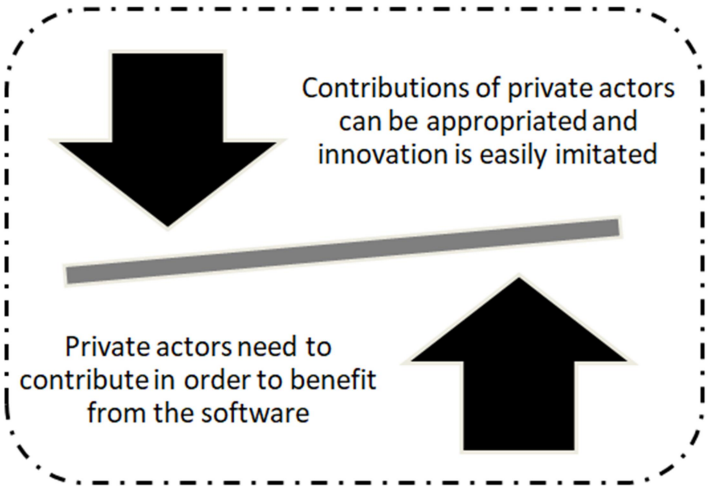

Figure 2.4 The business dilemma in open source software

Theoretically, the business dilemma would discourage private actors from contributing their private knowledge with the collective as rivalry would lower their competitive advantage. However, in reality, there are some cases where we do not observe this constraint. Instead, private actors end to innovate and contribute within their open source community.

For example, Santos et al. (2013) show that private actors are actually commercially investing and contributing to the software. Statistics provided by Skok (2013) viewed that over 2,000 open source software projects related to the

healthcare industry were adopted by different firms in 2013. These projects include medical, hospital and clinical, dental, nursing, patients and laboratory software. In addition, the adoption of open source software by firms is extending across different industries such as government, financial, media, automotive and retail (Skok, 2013) .

In addition, BlackDuckSoftware (2015) shows that open source software is growing within companies (see figure 2.5). 78% of companies are running their business on open source software. In addition, the analysis shows that 39% of these companies are planning to start their own external open source project, 47% are planning to release some of their internal tools as open source projects, and 53% are expecting to decrease barriers for their employees to participate in open source projects.

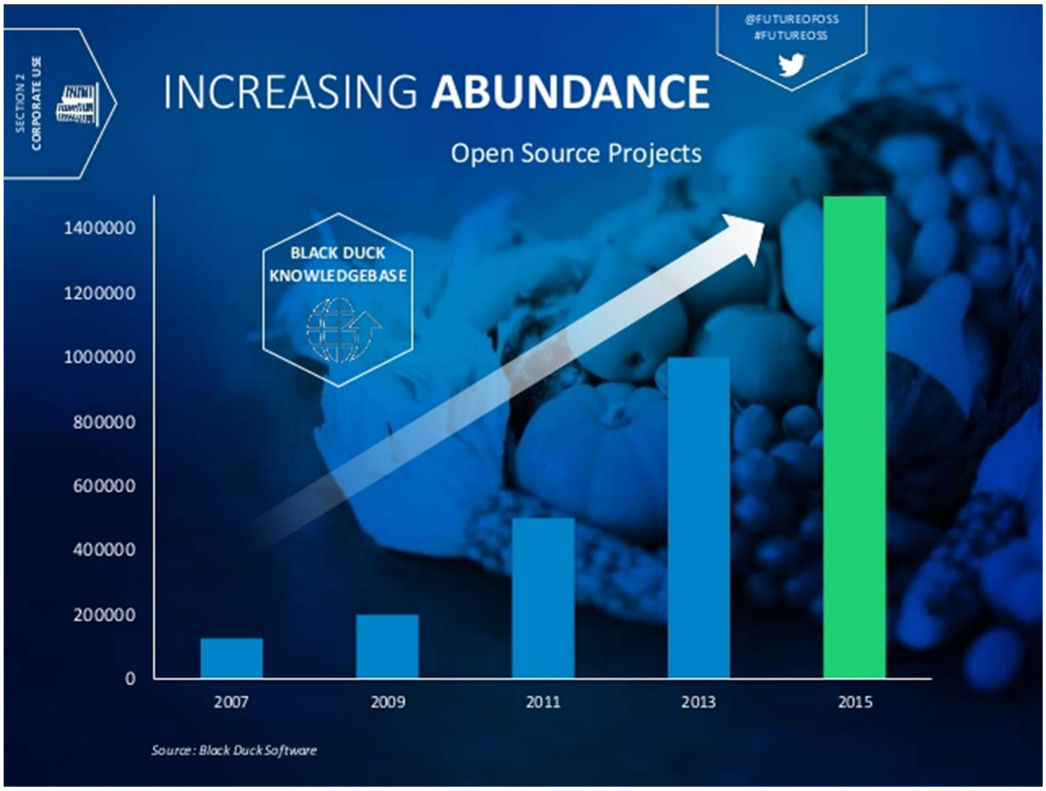

Figure 2.5 Adoption of open source software in different companies

Therefore, this thesis aims for understanding the 'private-collective' model of innovation in open source software that is declared under permissive licensing.

This suggested to be done through the research question: How can the private actors invest in open source software that is declared under permissive licensing without experiencing a business dilemma?

Chapter 3 Governance of the Commons

3.1 Overview

This chapter starts with describing open source software as a commons that suffer from collective action dilemma. After that, in sections 3.2 and 3.3, it is proposed that Ostrom's evolutionary theory of collective action can be used as a lens for investigating the patterns of the private contributions. Finally, the suitability of the theory is explained in section 3.4.

3.2 Collective Action Problem

Open source software is a 'commons' where 'Commons' is a term usually utilised to refer to non-excludable goods that are prohibitively expensive to exclude certain people (Ostrom, 1990). Non-exclusion means that the goods are able to be utilised without payment. Users may pay for non-exclusive goods, but excluding non-paying users is costly and difficult.

For example, fireworks are non-exclusive goods because anyone can watch them even if they did not pay. Thus, open source software is non-excludable because the software is publicly available and can be copied without payment.

Examples of 'commons' are natural resources such as water fisheries and forests, and knowledge such as open source software.

'Commons' suffer from collective action problems; a collective action problem is a problem that is caused by

individual actions in a ‘commons’ (Ostrom, 2010). Several collective action problems have been identified such as the ‘tragedy of the commons’ problem (Hardin, 1977)and the ‘free riding’ problem (For example, Isaac and Walker, 1988, Raymond, 1999b, Franck and Jungwirth, 2003, Lee and Cole, 2003, von Hippel and von Krogh, 2003, Baldwin and Clark, 2006).

For example, the “tragedy of the commons” problem (Hardin, 1977) is the problem related to the depletion of the natural resource because of the overconsumption behaviours done by people over time.

The chapter focuses on the problem of free riding because of its relation to the business dilemma explained in chapter two; the free riding of the shared private investments would cause the business dilemma experienced by private actors in permissive open source software.

Generally speaking, the problem of ‘free riding’ refers to the decreased incentives of participants to contribute to a ‘commons’ resource because such contributions will be shared with everyone, even non-contributors (Olson, 1967). This problem is related to a non-exclusivity characteristic of a ‘commons’ resource.

This thesis suggests another collective action problem that is related to a non-exclusivity characteristic in open source software which is a business dilemma experienced by private actors in open source software. As explained in section 2.5, sharing of private knowledge would affect the competitive

advantage for private actors because imitation of their contributions would be easy. Accordingly, they would be experiencing a business dilemma.

As will be discussed in the following section, it is proposed that Ostrom (1990) evolutionary theory of collective action is a suitable theory that can describe and explain what is happening with the private actors when they keep their contributions to open source software despite the business dilemma.

3.3 Theory of Collective Action

There are two schools of theories of collective action that are discussed in the literature in order to solve collection action problems: the conventional and evolutionary theories.

Hardin (1977) conventional theory of collective actions is a theory used to suggest governance solutions for collective action dilemma. This theory explains that participants (while using the 'commons') will perform actions that increase their short-term interests only and accordingly the tragedy of the commons would emerge. This theory argued that the only remedy for this problem is through a centralised private or governmental authority that will regulate access and use of the commons by different people. This centralised authority could be a private or governmental party (For example, Demsetz, 1967, Lovejoy, 2006).

However, Nobel Prize winner Elinor Ostrom suggests that Hardin's solution is straightforward but impractical, especially in complex settings and environments. She suggested an

evolutionary theory of collective action (Ostrom, 1990). This theory suggests that:

First, the central authority cannot be always the acceptable solution to govern the commons (Ostrom, 2009).

For example, private ownership for the commons would force the appropriate policies, strategies and rules to protect the long-term benefits through sustainability assurance. This kind of ownership, however, is not the solution in every circumstance, as certain commons cannot have boundaries enforced, such as ocean fisheries or fields that vary in productivity levels throughout the seasons.

Another example, governmental ownership may be another solution to protect the resources where the whole area of the resources is shared by a community group that will exclude others from using the resource arbitrarily. Whereas the government enforces policies, strategies and rules of cooperation to guarantee sustainability, government will not always act ideally for the sake of the public, since they may not be fully aware of the ecological system surrounding the resources. Hence, they may not approve changes in the public interest, or they may not have the sufficient motivation to create optimal changes.

Second, users of the 'commons' are rationale. They understand that focusing solely on their private benefits and interests is harmful because it would lead to a long term damage for the 'commons' and their private benefits accordingly.

Third, users of the 'commons' are capable and able to act without a forced authority or guidance. They can use their knowledge and experience in extracting themselves from collective action problems that they experience (Ostrom, 2007). This can be done through finding their own ways for aligning their private benefits and interests with the collective benefits and interests.

3.4 The suitability of a Theory

This chapter argued that Ostrom (1990) evolutionary theory of collective action would be of benefit for understanding and explaining why and how private actors are investing and sharing their investments in permissive open source software without experiencing the business dilemma.

It is believed that this theory is suitable for this thesis for many reasons. First, the theoretical perspective of this theory privileges the human agency and the human interactions over other materiality metrics in solving dilemmas. Thus, this theory would be better for answering the research questions (as will be discussed shortly).

There are different theoretical frameworks that are employed in the IS literature. Some theoretical frameworks have a theoretical perspective that privilege materiality of technology over the human agency (Markus and Silver, 2008). Such theoretical perspective emphasized that technology plays a crucial role in shaping outcomes and causing phenomena in open source software.

For example, Yamauchi et al. (2000) employed this theoretical perspective to show the important role of technology in supporting the work of participants in open source software. In addition, Scacchi (2002) employed this theoretical perspective and explained that technology supports elicitation, analysis, specification, validation and management of requirements in open source software. Accordingly, technology supports the proper development of open source software.

However, Orlikowski and Baroudi (1991) argued that various theoretical frameworks that are employed in the IS literature found to privilege the human agency over materiality metrics while exploring a certain phenomenon. Such theoretical perspective emphasized that human interactions and their shared norms play a crucial role in shaping outcomes and in explaining phenomena.

For example, Nakakoji et al. (2002) employed such theoretical perspective in order to study the evolution of four open source software and their communities. As a result, they could understand collaboration between participants and how collaboration contributed to the success of the software.

Moreover, Shah (2006) employed the same theoretical perspective and explored participation in open source software. Findings show that motivations were important for participants to initiate their participations in open source software but their shared norms were also important to sustain such motivations.

In this thesis, it is believed that a theoretical framework of a perspective that privileges human agency over materiality shall be employed in order to answer the research questions. This is justified because it is important for this thesis to identify the private actors and their patterns of contributions (rather than the technology they use) in order to understand how they extract themselves from the business dilemma. Accordingly, Ostrom (1990) evolutionary theory of collective action is suitable to be employed.

Second, this thesis will analyse permissive open source software where formal regulations against free riding are absent. Therefore, this theory is suitable because it focuses on a context where formal authority lacks or fails.

Chapter 4 Research Methodology

4.1 Overview

This chapter discusses the conceptualization of this research. In section 4.2, various philosophical standpoints will be examined: critical standpoints, pragmatism, positivism, and interpretivism. In section 4.3, this research is positioned alongside these philosophical standpoints based on the research question identified previously. In section 4.4, it is divulged that a single case study will be used. The logic surrounding the use of a specific case study is defined.

4.2 Philosophical Standpoints in Research

Guba and Lincoln (1994) suggested that understanding the research paradigm is key when attempting to answer a research question. 'Paradigm' here refers to the researcher's philosophical views which may steer the research process.

Denzin and Lincoln (2011) explained that these beliefs and assumptions are a collection of 'epistemological, ontological, and methodological' principles held by the researcher.

1. Ontology: methods of constructing reality. 'How things really are? and 'how things really work?' (Denzin and Lincoln, 2008 p.201)

2. Epistemology: methods of determining scientific knowledge. 'Epistemology is the study of the criteria by which

we can know what does, and does not, constitute warranted, or scientific, knowledge.' (Johnson and Cassell, 2001 p.2).

3. Methodology: methods of conducting research scientifically. 'A way to systematically solve the research problem' (Kothari, 2004 p.8).

Generally speaking, four main philosophical standpoints have been identified (see table 5.1): positivism, interpretivism, critical, and pragmatic (Chua, 1986, Orlikowski and Baroudi, 1991, Goldkuhl, 2012, Blackstone, 2012). Initially, researchers identified positivism and interpretivism as the two main philosophical standpoints in social research (Crotty, 1998). Researchers gain knowledge based on their interactions and ongoing exchanges with society; this exchange is either objective or subjective. Following the identification of positivism and interpretivism, pragmatic and critical standpoints were then discerned.

1. Positivism: This philosophical standpoint employs quantitative methods to deductively test hypotheses, whereas interpretivism employs qualitative methods to inductively generate theories. Some researchers (for example Crossan, 2003, Johnson and Onwuegbuzie, 2004) argue that the positivist standpoint begins with hypotheses created from common literature; the data collection process is uncomplicated and speedy. Moreover, data collected covers a vast number of observations from a larger sample. Owing to this, results are dependable and able to be generalised.

2. Interpretivism: Researchers that employ this philosophical standpoint argue that humans are not objects in laboratories; they have emotions and attitudes that affect, and are affected by, society. Thus, results from the positivist standpoint cannot be explained or implemented in a specific social context (Benton and Craib, 2010). An interpretivist standpoint is needed because it delves deeply into rich data and is flexible in its understanding, discovering novel findings related to beliefs, attitudes, opinions, and feelings. A major criticism of the interpretivist standpoint is the possibility of bias from the researcher. Therefore, the reflexivity concept is essential in overcoming this bias and will be discussed in this chapter (Hibbert et al., 2010).

3. Critical: This philosophical standpoint is utilised to eliminate causes of unwarranted domination (Klein, 2004). The researcher investigates the history of a phenomenon by describing knowledge in order to understand how beliefs and values shape, and are shaped by, investigation (Avison and Pries-Heje, 2005).

4. Pragmatism: This philosophical standpoint is employed by researchers who seek to promote change. Different types of pragmatism have been identified. For example, Goldkuhl (2008) suggested three types: functional, referential, and methodological. Regardless of the type of pragmatism, researchers make use of data generated from assessment and interventions of a certain phenomenon; they

check this data and its functionality in relation to action and change (Goldkuhl, 2012).

4.3 Thesis positioning

This research is guided by an interpretive, qualitative, philosophical standpoint. Interpretivism in qualitative research is crucial for analysing and identifying the subjective meaning of a phenomena being studied (Silverman, 2013). Such subjectivity is crucial for understanding the context of the phenomenon and the way it affects, and is affected by, processes (Walsham, 1993). This research uses a qualitative method with different data sources in order to identify how business dilemmas that private actors tend to experience in open source software can be solved?

The interpretive, qualitative standpoint acknowledges that a deeper understanding regarding the research area of enquiry is more important than the generalization of findings (Siggelkow, 2007). Therefore, a case study became the qualitative method used in this research. The OpenNebula project is the case study presented in this research. Section 4.4 describes the process of selecting OpenNebula as the core case study and the variation between OpenNebula and case studies frequently analysed within the literature.

4.4 Case Study Design

Per the interpretivism philosophical standpoint, qualitative method and research design are required in this thesis in order to gain access for rich data. Rich data will be analysed in order to identify the different participants, to

determine their motivations and behaviours, to precisely recognize their private contributions, to investigate and identify how and why private actors are sharing their private investments rather than concealing them.

A case study is one of the qualitative methods employed to aid researchers in understanding their research in-depth and gain knowledge of the phenomena they are interested in. The knowledge construction nature of case studies assists researchers in viewing the complex picture that comprises people's lives and behaviours. Eisenhardt (1989) suggested that researchers could utilise case studies to systemically predict and develop theories across various disciplines.

A single case study of open source software is presented in this research. The rationale behind utilising a single case study is that a noteworthy case study is required in order to fulfil the aim of this thesis. This case must: (1) be declared under a permissive open source license, (2) be active for a significant duration and possess rich data archives that can be employed during analysis, (3) consist of software that is utilised within the market by various industries, and (4) possess different types of commercial actors from various industries, who participate in the software. Thus, a single critical case, also called also an 'Information-Oriented' case study (Flyvbjerg, 2006) is more suitable.

4.4.1 The selection of OpenNebula case study

Guided by Lerner and Tirole (2005b) list of open source licenses, data was collected from various online resources to identify available permissive open source software. These resources included: technical presentations, survey results, and project data recorded online. A list of permissive open source software was identified.

Following this, the complexity of the software on the list was examined based on the availability of: (a) software e-newsletters, (b) code repositories, (c) mailing lists, and (d) technical documentation. Consequently, certain software was excluded due to short-term activity, as well as instability regarding source code releases.

The resulting software options were analysed based on their e-newsletters. The method of utilising e-newsletters in order to analyse a gathered selection draws on Scott (1990) method of depending on documentation that is 'authentic, credible, representative, and meaningful'.

This improved my understanding the wider issues regarding the enquiry under investigation. Newsletters are typically employed to provide general views on any organization/event/project/community, etc.

The analysis of e-newsletters was crucial because the method included examining: (1) the existence of individual participants and business appropriators as community members for the software, and (2) the technical feasibility of

the software in terms of its implementation in the market by different firms.

Based on the analysis of e-newsletters, OpenNebula was selected as the software to be examined within this research. The e-newsletters for OpenNebula consist of monthly e-newsletters which commenced in June 2011. These newsletters were recorded until September 2014, amounting to a total of 44 newsletters. These e-newsletters were judged to be authentic and credible because they satisfied the authenticity conditions suggested by Platt (1981). For example: the documents were published on OpenNebula's official website, the newsletters were original copies produced for the project and published by an official employee, they were approved by the project leader, and they suited a standard format which was approved by the OpenNebula team.

OpenNebula is the 'exploratory case' (Yin, 2014) that can be used in this thesis in order to explore how the 'private-collective' model of innovation is working under the permissive context for open source software.

OpenNebula can be described as follows. First, OpenNebula is cloud computing software: software that develops technologies in order to virtualize the infrastructure of a computer network such as computers, servers, routers, etc. Cloud computing software is a relatively recent technology that has advanced rapidly (Marston et al., 2011). This is crucial because the cloud computing industry is highly

attractive to supercomputing firms who want to virtualize their data centres (Milojicic et al., 2011).

In the early stages of OpenNebula, the two founders of OpenNebula software participated in “the European Union’s Seventh Framework Programme”. This programme helped the founders in understanding the business needs in the European market for cloud computing software like OpenNebula. As a result, private actors are important participants in OpenNebula. OpenNebula is now utilised by various firms across the market, in a wide range of industries, as table 4.1 shows.

Table 4.1 Examples of firms in different industries using OpenNebula in their business

Industry sector	Firm examples
Telecommunications and Internet	Akami, BlackBerry, China Mobile
Government	National Central Library of Florence, bDigital, Deutsch E-Post, RedIRIS, GRNET, Instituto geoGrafico Nacional, CSIC, Gobex
Financial, Banking, and Risk Analysis	Monte Dei Paschi Di Siena, produban, LexisNexis, AXCESS Financial
Media and Gaming	BBC, Unity3d, R.U.R., Crytek, iSpot
Hosting Providers	OnVPS, NBSP, Orion VM, CITEC, LibreIT, Quobis, Virtion, OnGrid, Altus, DMEx, LMD, HostColor, Handy Networks, BIT, GoodHosting, Avalon, noosvps, bpsNode, PTisp, Ungleich.ch, TAS France, TeleData, CipherSpace
SaaS (Software as a Service) and ecommerce	Scytl, LEADMESH, optimalPath, RJMetrics, Carismatel, Sigma, niar.me, GLOBALRAP, Runtastic, Moz, Rentalia, vibes, Yuterra
Aerospace	ESAC ESA, ESRIN ESA, NASA, ScanEx
Supercomputing	NCHC, CESGA, CRS4, PDC, CSUC, Tokyo Institute of Technology, CSC, HPCI, Cerit-SC, LRZ, PIC
Research	FermiLab, NIKHEF, LAL CNRS, DESY, INFN, IPB Halle, CSIRO, fccn, National Institute of Advanced Industrial Science and Technology, KISTI, KIT, ASTI, Fatec Lins, MIMOS, SZTAKI, Ciemat, SurfSARA
Academic	Telecom SudParis, Harvard School of Engineering and Applied Science, Universidade Federal Do Ceara, Instituto Superiore Mario Boella, Academia Sinica, UNACHI
Information Technology	IBM, DELL, CentOS, KPMG, Engineering, Logica, CloudSky, Netways, ippon, Terradue, Unisys, MAV Tecnologia, Liberologico, Etnetera, EDS Systems, inovex, bosstek
Cloud Products	ClassCat, Hexagrid, CloudWeavers, Impetus, ZeroNines

Second, OpenNebula started as a research software project, initiated in 2003 by an associate and assistant professors at Complutense University of Madrid. After years of developing the software, they declared the research software project as open source under an Apache v2.0 open source licence. Apache v 2.0 is a permissive license for open source software (Lerner and Tirole, 2005b). Having a permissive license reinforces the business dilemma experienced by private actors in OpenNebula.

Third, OpenNebula is permissive open source software that was declared open source is still active at the present time. Thus, the project has vast archives that can be employed for analysis. Yin (2014) argued that a case study with rich data is crucial for effectively conducting necessary in-depth analysis.

Chapter 5 Data Collection and Analysis

5.1 Overview

This chapter explains data collected and analysed. Several methods of data collection are employed. Section 5.2 describes the data and the rationale behind selection. Emphasizing the importance of carrying out robust research, sections 5.3 and 5.4 are written to discuss reflexivity and other quality measures implemented in this research. Moreover, section 5.5 presents the ethical procedure used in this research.

5.2 Data Collection

The data collected for this analysis consists of four types of online data: emails from OpenNebula's mailing list, requests added into the development portal of OpenNebula, source code commits[2] from the GitHub portal, and technical documents provided on OpenNebula's official website. It is important that the logic and reasoning behind these categories of online data is explained.

Utilising more than two types of data within the same research process is known as 'Triangulation' (Mitcbell, 1986, Thurmond, 2001). According to Thurmond (2001), there are different types of triangulation: methodological triangulation,

[2] A source code commit is a technical term used to describe the latest changes applied to the source code repository for the software.

investigator triangulation, theoretical triangulation, and data sources triangulation.

Table 5.1 outlines the difference between the types of triangulation employed in research (Jick, 1979, Boyd, 1993, Nau, 1995, Mingers, 2001, Denzin and Lincoln, 2011, Myers, 2013).

Denzin and Lincoln (2008) suggested that depending on one method or data type is not always enough to answer research questions, as each method or source of data possesses pros and cons. Utilising varied sources of data and different methods solves any problems that arise when using just one source.

Moreover, Nolan and Behi (1995) discussed that the convergence between the different measures employed through triangulation can increase research confidence regarding the phenomenon that is under analysis.

In addition, Lincoln and Guba (1985) emphasized that studies in qualitative research apply triangulation in order to seriously consider research and results.

This research proposes that triangulation be employed to respond to the research question. This is because this research follows the 'interpretivism' philosophical standpoint which depends on the researcher understanding and interpreting data collected and analysed. However, it is also essential within this research to gain justified reliability regarding qualitative findings, an inclusive view of the phenomenon being studied, and a low level of potential bias.

One possible way to achieve this is by applying triangulation to the research (Denzin and Lincoln, 2011).

In addition, data source triangulation is employed in this research; different sources of data need to be collected and analysed in order to answer the research question. Four different types of online data are collected and analysed: mailing list emails, requests added into the development portal, source code commits added to GitHub, and available documentation of OpenNebula.

The first source of data collected was mailing list emails. These emails contained qualitative data that was analysed using thematic coding and theoretical memos (as will be discussed in chapters 6 and 7).

After that, the second and third types of data were collected: requests added into the development portal of OpenNebula and source code commits from the GitHub portal of OpenNebula. These data sources served to improve the understanding of individual cases within qualitative results. For example, one of the qualitative findings revealed that private actors are contributing their private investments of the source code into OpenNebula. Utilising quantitative data in the development portal reveals that contributions can be measured by 'requests' added into the development portal.

Table 5.1 Triangulation Types

	Methodologic	**Investigator**	**Theoretical**	**Data Sources**
Definition	Two or more methods employed. Two types of methodologic triangulation: Across-method and within-method	Two or more investigators or data analysts conducting the research.	Multiple theories and hypotheses employed within the research.	Different sources of data utilised. These sources differ in time, people, or space.
Benefits	Increase validity. Identify relationship.	Acquire a varied skillset and viewpoint. This leads to enhanced research and decreased bias.	Move beyond specific theoretical perspectives and explanations.	Achieve an improved and comprehensive understanding of the varied perspectives within the research. Decreases bias.

Moreover, both types of data are employed in order to gain new information that was difficult to capture using the emails in the mailing list alone. For example, a comprehensive understanding of the technical layout of OpenNebula software was captured using quantitative records. Quantitative records provide a numeric categorization of the different modules of OpenNebula software and their development over time.

However, emails in the mailing list and requests added into the development portal contain technical terms related to the functionality of OpenNebula as a case study in this research. These technical terms are somewhat difficult to comprehend, thus a fourth source of data was gathered: documentation available on OpenNebula's website. Employing this data reduced my bias as a computer engineer.

For example, as an engineer I initially classified the different components of OpenNebula, based on their technical function, into 'Infrastructure as a Service (IaaS)' and 'Platform as a Service (PaaS)'. However, using the documentation, I was able to expand my comprehension; I understood the technical implementations of these components and, accordingly, I was able to classify these components into 'core' and 'complements'. In addition, I understood the level of implementation required for both types of component. Thus, I was able to label particular levels of implementation as 'software complementarities' and reveal that 'software complementarities' is the outcome resulting

from different actions performed by participants in OpenNebula.

5.2.1 The First Type of Data: The Mailing List

A mailing list is communication between participants in the open source software, in which they share and exchange information. A mailing list is a type of text communication employed in online research (Mann and Stewart, 2000). More specifically, it is a common technique used in the literature of open source software to understand the historical behaviours of participants (for example, Kuk, 2006).

In addition, emails in mailing lists contain texts that are rich in their implications. Texts are valuable sources of information regarding actors and their communications, contributions, and interactions, including their organizations. Mailing lists can provide not only text but an understanding of the socially constructed organizations that they formulate (Atkinson, 2004).

Texts inside documents covers the lifespan of an organization, and these texts contain massive chunks of data relating to objects and actors (Berger et al., 2007). Texts are suitable methods, especially if the data required about actors and their contributions/interactions is needed based on the history of their communications over a significant time period (Barley and Tolbert, 1997, Suddaby and Greenwood, 2009).

During the focused reading I conducted at the early stages of this research for the e-newsletters of OpenNebula (in order to be able to identify a suitable case study for this

research; refer to section 4.4.1), I was able to identify two different online sources in which online texts are shared between participants in OpenNebula. These online sources were IRC (Internet Relay Chat) sessions and mailing lists.

IRC sessions are live chats between participants and key members of OpenNebula which are not archived. Thus, these sessions cannot be employed to analyse data exchanged between participants over time, as no data can be collected and analysed.

Mailing lists are a source of communication for participants, enabling them to exchange information. There are three mailing lists within OpenNebula:

1. 'Community support and users' discussions' mailing list: This mailing list was utilised by participants in order to discuss different technical problems encountered, suggest future development activities for the software, and share ideas and consultancy. This mailing list was the most active list in the project; it started on March 2008, running to the present. This mailing list was employed in this research.

2. 'Development discussions' mailing list: This mailing list was employed by participants in order to discuss development activities for different OpenNebula releases. This list was not heavily used because participants relied on available documentation published on OpenNebula's official website. Consequently, this list was not used within this research.

3. 'Community discuss and collaboration' mailing list: This mailing list was employed by participants in order to announce

news, outreach events, conferences, and technology days that the core members of OpenNebula participated in. Consequently, this list was not used in this research study as it represented an announcement space.

Accordingly, the 'community support and users' discussions' mailing list is the mailing list used initially in this research. Anyone can view this mailing list. However, one needs to sign-up in order to participate in the list by sending and receiving emails. Signing up the mailing list requires registering a unique username, password, and email address.

The emails, usernames, and email addresses of participants who utilised the mailing list from May 2008 until September 2014 were extracted. The resulting data was vast; the data contained 18,890 emails between 1,337 participants.

Due to the impracticality of dissecting these emails qualitatively, participants were categories into five groups based on email affiliation.

1. Participants with 'OpenNebula' email affiliation: 8 participants with email addresses ending in opennebula.org.
2. Participants with 'education' email affiliation: 68 participants with email addresses ending in .edu.
3. Participants with 'government' email affiliation: 16 participants with email addresses ending in .gov.
4. Participants with 'corporation' email affiliation: 699 participants with email addresses ending in .org.

5. Participants with ‘individual’ email affiliation: 546 participants with email addresses ending in .hotmail, .gmail, or.yahoo.

These groups are essential because they cover all types of participants in OpenNebula. The number of emails sent by each of the 1,337 participants was counted. Then, five participants from each group were analysed. These five participants per group were selected based on the number of emails sent in the mailing list and the duration of their participation. Participants who sent a high volume of emails, and participated for long periods of time, were selected, as they were the most active.

The first sample for analysis consisted of 25 participants (5 participants from each group) and 7,017 emails (see table 5.2). 25 is an acceptable sample size that can be employed when using qualitative research methods (Charmaz, 2006). Emails are extracted and added to NVivo 10 software in order to be used for the analysis.

It is worth mentioning that the anonymity of participants’ names and related information was required in this research. Thus, a special naming convention was used. For example, the most active participant from individual email affiliation was labelled as ‘AnonyAI1’. Where ‘Anony’ means ‘anonymous’, ‘AI’ means ‘affiliation is individual’. ‘1’ refers to the first participants within a category to send emails via the mailing list.

Table 5.2 A sample of participants analysed from the mailing list

	Participant's Name	Participation Duration	Number of Emails Collected
Individual Affiliation	AnonyAI1	May 2009 – April 2012	177
	AnonyAI2	March 2012 – Sept 2014	130
	AnonyAI3	Jan 2011 – June 2014	130
	AnonyAI4	Sept 2011 – Sept 2014	123
	AnonyAI5	Oct 2012 – May 2014	63
Government Affiliation	AnonyAG1	Oct 2010 – Sept 2014	138
	AnonyAG2	July 2012 – May 2014	53
	AnonyAG3	June 2010 – Aug 2011	25
	AnonyAG4	Oct 2011 – Jan 2012	13
	AnonyAG5	June 2012 – June 2013	12
Education Affiliation	AnonyAE1	Jan 2013 – June 2013	69
	AnonyAE2	April 2013 – Sept 2014	37
	AnonyAE3	March 2011 – Aug 2012	42

	AnonyAE4	Nov 2011 – Feb 2012	36
	AnonyAE5	Nov 2009 – Sept 2010	25
Corporation Affiliation	AnonyAC1	Sept 2010 – Sept 2014	230
	AnonyAC2	Sept 2013 – Sept 2014	165
	AnonyAC3	July 2013 – Oct 2013	150
	AnonyAC4	Oct 2010 – Sept 2013	143
	AnonyAC5	Nov 2011 – Sept 2014	124
OpenNebula Affiliation	AnonyAI1	March 2008 – Sept 2014	1,315
	AnonyAI2	March 2008 – Sept 2014	1,128
	AnonyAI3	March 2008 – Sept 2014	1,011
	AnonyAI4	March 2008 – Sept 2014	930
	AnonyAI5	March 2008 – Sept 2014	748
Total		**25 participants**	**7,017 emails**

5.2.2 The Second Type of Data: The Requests

The analysis of emails from the mailing list (data collected in section 5.2.1) revealed the different participants who were involved in OpenNebula as well the different contributions that they had made to OpenNebula software (as will be discussed in chapter 6). This analysis implied that the result of participant contribution is closely linked to developing the source code of Open Nebula.

Participants discuss the development of the source code in the mailing list and officially request the development of the source code by adding official requests to the development portal. The development portal of OpenNebula is an official web page that contains all requests added by participants to develop OpenNebula software. Participants can add a request to report a software bug or create a new software feature.

To ascertain which discussions within mailing lists were reflected in the source code, it was essential to analyse requests within the development portal. Analysis of the source code supports understanding and identification regarding the technical layout of the software.

The number of requests added to the development portal is continually increasing over time (see figure 5.1). In addition, within the sample collected from March 2009 until November 2015, 85% of requests added to the development portal were closed, meaning that requests from participants were fulfilled and the source code was developed.

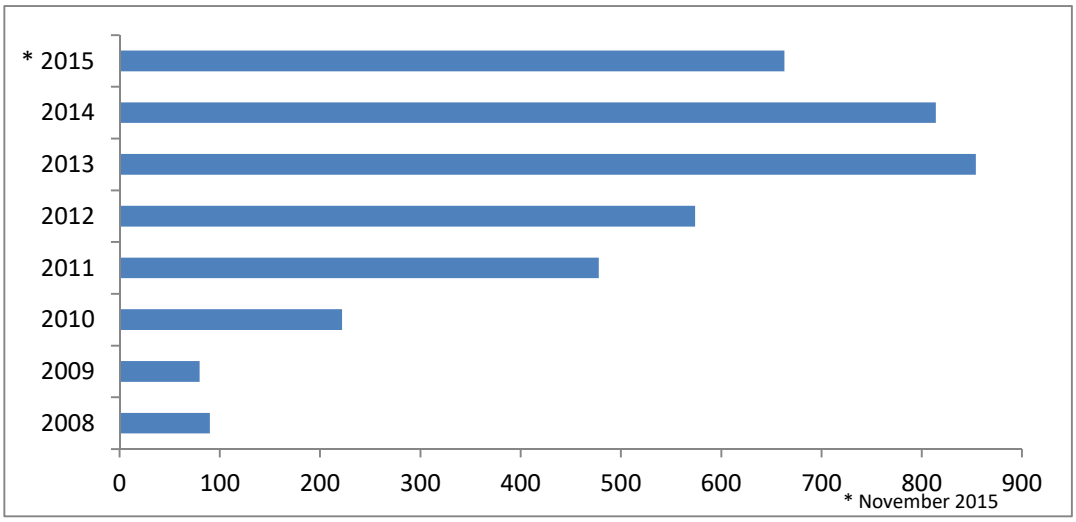

Figure 5.1 Number of requests added over time

The second set of data collected comprised of requests from OpenNebula's development portal; the theoretical sample method was employed to achieve this. On the development page, each request possessed a unique tracking number, tracker, author, assignee, status, start date, category, priority, target version, and update date. A total of 3,482 requests were extracted from the development page for the period of March 2009 to November 2015.

5.2.3 The Third Type of Data: The Commits

Gathered data from OpenNebula's development portal reveals that requests and suggestions ensure that OpenNebula is further developed. Technical details regarding OpenNebula software can be tracked and understood by using the data collected from the development portal. However, quantitative records relating to the number of changes implemented over time are stored in the GitHub portal (as

indicated in the development portal). The number of changes implemented is technically referred to as he number of source code commits.

The GitHub portal was used to gather data regarding source code commits; this data was then employed within this research. The GitHub portal is an official webpage on which the source code of OpenNebula is stored over time. The number of source code commits was extracted into an Excel sheet. These commits were counted on a monthly basis from March 2009 until the end of November 2015.

As will be discussed in chapter 7, this data was needed in order to understand and measure the development of OpenNebula software over time. In addition, this data was used to measure the development of the different components of OpenNebula software including core and complements components. The data was also essential for ascertaining how participants contribute to the development portal via requests, and how they expand the development of source code commits via GitHub.

5.2.4 The Fourth Type of Data: The Documentation

Data collected from the development portal was qualitatively analysed. However, the data contained certain technical terms that were difficult to understand. Therefore, a third set of data was collected including four technical documents that were available on the official web page for the project (see table 5.3).

Table 5.3 Description of the third set of data collected in this research

Documentation Title	Documentation Date	Description
Release 1.0	July 2008	Technical documentation for upgrade release number 1. This documentation gives technical details about the first official release of the project. These details include: technology utilised, supported platforms, development languages, etc. Moreover, this documentation contains the basic function for the project as a virtual infrastructure engine.
Release 2.0	October 2010	Technical documentation for upgrade release number 2. This provides information on the second official release, including community engagement, maturity and functionality.
Release 3.0	October 2011	Technical documentation for upgrade release number 3. This technical documentation provides details regarding the third official release. These details include new components added to manage the internet cloud for the project. This documentation makes it clear that the project possesses core components, and other peripheral components are added.
Release 4.0	May 2013	Technical documentation for upgrade release number 4. This technical documentation provides details regarding the fourth official release for the project. These details include core components within the project, end-user components, administration interfaces, etc.

Reports offer technical information, so documentation is essential for improving software understanding. This documentation contained a glossary of terminology and definitions related to the components of the software. Furthermore, this documentation contained screenshots and simplified diagrams that aided in developing an improved understanding of the computer network employed in OpenNebula.

Analysing these three data sets generated an improved level of understanding regarding OpenNebula as a case study and the governance used in OpenNebula.

5.3 Reflexivity

Interpretive, qualitative research is characterised by a high degree of subjectivity and engagement by the researcher. A researcher is required to possess a level of theoretical sensitivity in order to grasp the significance of data sets (Strauss and Corbin, 1990). However, this may lead to bias in qualitative research. Therefore, qualitative research requires a high degree of reflexivity in order to overcome bias. Reflexivity is defined as the sequential process of questioning methods of conducting research, also called 'reflection', as well as being willing to alter actions, also called 'recursion' (Hibbert et al., 2010). Reflexivity is deemed crucial in this research, given the ontological position I have taken as an interpretive researcher, whereby the significance of data is analysed and translated (Walsham, 1995).

In this research, I have not tried to remove myself, as a computer engineer, from the technical understanding required for the project. Neither have I tried to remove myself as a researcher from understanding and conducting the writing of a thesis. However, I have tried to be reflexive throughout the three stages of research suggested by Finlay (2002): the pre-research assumptions stage, the data collection stage, and the data analysis stage (see Figure 5.2).

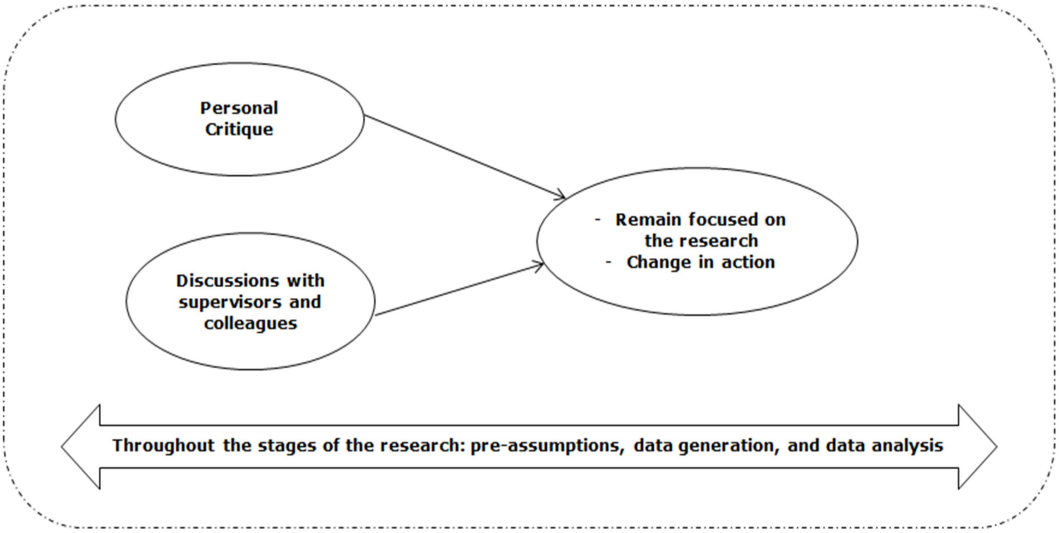

Figure 5.2 Reflexivity in this research

It was important to critique this research, including aspects that had been neglected or emphasised (Cunliffe and Jun, 2005). In addition, engagement in discussions with my supervisor and colleagues was essential. This led to the stabilization of assumptions and concepts within the research, as well as altering aspects of the research in order to achieve reflection and recursion.

In the pre-research assumption stage, I was aware of my personal motivations for conducting this research. I

decided to conduct this research as part of my career advancement process as a lecturer at the University of Jordan. In order to advance an academic career, it is necessary to hold a PhD, therefore, I studied open source literature and developed pre-understanding regarding my research subject and interests. These motivations allowed me to focus on research.

During the formation of this research, a rich understanding of the research topic and an improved formulation of the research question was attained. For example, the initial research focus was aimed towards understanding governance in open source software. However, over time, it became clear that for the research question to be positioned correctly within open source literature, and to achieve originality, the question needed to focus on governance that solve collective action dilemma in permissive open source software. This would ensure describing and explaining what happens with private actors in permissive open source software.

During the data generation phase, data was collected from the OpenNebula mailing list. The vast amount of emails within the mailing list was staggering. My initial plan was to analyse all data within the mailing list, but it was difficult to analyse all emails within the proposed time frame. Therefore, after consultation with my supervisors, only a sample of these emails was applied (details in section 5.2).

However, within analysis it was crucial to generate additional data from multiple resources in order to achieve the level of understanding required for this research.

In the data analysis stage, reflexivity was achieved by following Srivastava and Hopwood (2009) reflexive framework for analysing qualitative data (see figure 5.3). Their framework encourages researchers to begin analysis by answering the question 'what is the data telling me?' This question is critical for developing theoretical understanding regarding data that may not have been present previously. Letting the data 'speak' is crucial as it improves reflexivity in research.

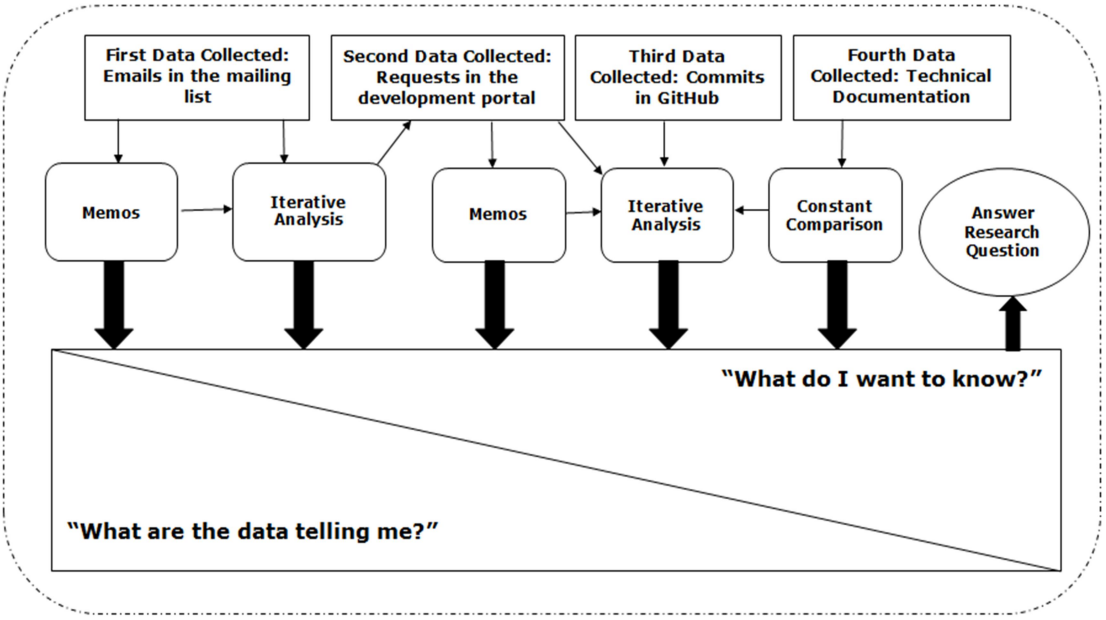

Figure 5.3 Reflexivity in this research

Theoretical memos were written at the beginning of the analysis phase; these memos described emails sent by participants. These theoretical memos were general memos

regarding the general communications of each participant. This aided in obtaining a clear picture of OpenNebula as software and the core focus of the participants. It also assisted with the inclusion of ideas that were not taken into consideration before beginning analysis.

Following this, the analysis of data was conducted in several iterations; each iterative phase had its own theoretical focus. Logs were kept regarding the analysed data for each participant, within each iterative phase. These logs were continually compared to avoid salient bias in the analysis.

Furthermore, additional technical documentation available on the official website of OpenNebula was studied in order to ensure technical understanding of OpenNebula.

Moreover, in the data analysis stage, reflexivity was also achieved through the salient shifts in my writing skills.

As a computer engineer, I could describe my own writing as a technically-intensive writing that focused on listing definite technical facts and concluding relationships between these facts.

However, I have learnt from my supervisors that the audience for this thesis differs somehow from purely technical audience that I used to write for. I was regularly encouraged to understand my new audience and I have gradually gained the needed writing skills through receiving feedback from my supervisors and colleagues as well as attending academic writing courses and one-to-one writing sessions.

Therefore, clarity and consistency are important aspects of this thesis; I introduce and clearly define the different technical and non-technical terms used in this thesis. In addition, I consistently used these terms through the thesis.

Moreover, I have learnt that thesis should be rigorous and simple. Thus, I have written plenty of drafts in which I have learnt how to write my ideas, develop arguments as well as logically present and justify the different contents of this thesis.

5.4 Quality of the research

The quality of this research was measured against its credibility, plausibility, and transferability (see table 5.4).

Table 5.4 Quality measures in this research

Quality Measure	Steps taken in this research
Credibility	Employing a representative sample of the data generated
	Utilising different sources for data generation
	Sharing ideas with supervisors and colleagues
Plausibility	Ensuring a detailed description of data
	Replicability
Transferability	Describing the context of the research thoroughly

Credibility is a concept coined by (Lincoln and Guba, 1985) in order to replace validity used to measure quality in quantitative research. Credibility is employed to ensure the believability of results and that the interpretation of

generated data remains as close as possible to the significance of the data (Silverman, 2006). Within this research, I followed several steps to ensure credibility:

First, I employ a representative sample of generated data. As there was such a vast amount of data extracted from the mailing list, it would be unreasonable and time-consuming to qualitatively analyse all of the data. Therefore, a representative sample from this mailing list was analysed. Accordingly, emails sent by the 25 most active participants in the mailing list were studied. The sample (that was rejected later) comprised of 25 participants, all of them originating from the official sponsor of the project.

Analysing all participants from one group and neglecting other participants was not credible. Therefore, participants in the mailing list were categorised according to their email affiliation. The most active participants from different email affiliations were selected (this sample was employed in analysis). This method allowed a greater number of participants to be analysed, and provided a broader research scope.

Second, I utilise different sources for data generation. The analysis of the mailing list enabled other sources of data to be used to enrich understanding of the project and answer the research question. This additional data was also gathered and studied. Collecting data from different resources ensured rich data generated for analysis.

Third, I Share ideas with supervisors and colleagues. This is crucial to achieve credibility as research is constructed by participants, the researcher, and readers of the research (Finlay, 2002).

As mentioned earlier in this section, the quality of this research was measured based on its plausibility. Plausibility represents the ability of the research thesis to convince the reader of the research interpretations (Thorpe and Holt, 2007). Plausibility in this research was achieved using the following steps:

First, I ensure a detailed description of data. This step was crucial as most of generated data in the mailing list contained rigid technical terms that were used in the project. These terms were not defined because they were well known by IT professionals within the project.

Second, I apply 'Replicability' (Easterby-Smith et al., 2008) of theoretical ideas via different iterations of analysis. Analysis went through several iterations. In every phase of iterative analysis, the analysis was repeated for all 25 participants. Therefore, to ensure the consistency and focus of the analysis for different participants within the same iteration, a template for each phase of iterative analysis was created, detailing the central argument. This did not eliminate the transparency required for data analysis.

The final measure employed to ensure the quality of the research was transferability. Transferability can be defined as the generalizability of the research; this is impossible in

qualitative research that uses a single case study as a method of interpretation. Within this research, a complex description was employed to provide context and describe cases that demonstrate similar ideas. For example, in this chapter, it is described that: (1) a permissive open source project was used; (2) the project was utilised by several private appropriators and firms; (3) the industry relies heavily on the virtualization of computer devices and this, in turn, ensures the development of software complementarities. Thus, this research can be applied to other open source projects that contain these basic assumptions.

5.5 Research Ethics

The ethical format for conducting this research was first submitted and approved by the Nottingham University Business School Ethics Committee. The issue of anonymity within the research project is examined in this section.

The two core sources of data utilised in this research were: project data published on the project website and the emails archived from the project mailing list. The public are able to post on the project website; however, just because they have posted information on a public forum does not mean they have consented to it being used for research purposes. In this case, obtaining consent from the vast number of participants becomes impractical and time-consuming.

Although conducting research without the informed consent of all participants is justified in this case, two main

conditions are crucial. First, anonymity should be guaranteed for all participants and direct quotation prohibited. The logic behind this is that if direct quotation is used, that quote could be searched for and discovered on the internet. Once the post is discovered, the username would be revealed, which may be used by the same person elsewhere on the web. Other clues regarding their identity may have been distributed in other areas of the site. Project manager was consulted regarding the issue of consent within the project, and the terms of this were agreed.

Chapter 6 The Private Contributions in OpenNebula

6.1 Overview

This chapter explains that private contributions by private actors are supporting the development of OpenNebula software in the form of collective complementarities. In addition, this chapter proposes that the active communications constitute a prerequisite to the active contributions of private actors to the software.

This chapter starts by introducing OpenNebula and focusing on its growth stage for data collection and analysis. The growth stage is remarkable because the community of participants is growing and the software is rapidly growing. A map for participants in OpenNebula is identified in section 6.3. The majority of participants in OpenNebula are private actors who belong to different companies in different industrial sectors. According to the literature, this shall reinforce the business dilemma in OpenNebula and private actors would withdraw their collective action.

However, section 6.4 shows that communications are playing a crucial role in encouraging contributions by participants in OpenNebula and that contributions support the development of the software in the form of collective complementarities.

6.2 OpenNebula as open source software

OpenNebula is the permissive open source software chosen for this research. As summarised in figure 6.1, OpenNebula started as a research software project, initiated in 2003 by an associate and an assistant professors at Complutense University of Madrid. After years of developing the software, they declared the research software project as open source under an Apache v2.0 open source licence.

Apache v2.0 is a permissive license for open source software (Lerner and Tirole, 2005b). As explained by Sen et al. (2011), permissive open source software does not require the sharing of the original source code and any modifications and amendments. Moreover, this license accepts the re-licensing of the software to any other open source or proprietary license other than the original license declared.

OpenNebula has gone through two stages in its development: the start-up stage (2005-2010) and the growth stage (2010 until now).

In the start-up stage, the founders of OpenNebula software participated in “the European Union’s Seventh Framework Programme”. This programme helped the founders in understanding the business needs in the European market for cloud computing software like OpenNebula. Their major achievement was the declaration of its first official stable release.

In the growth stage, 'OpenNebula Systems' was declared as the official company that sponsors OpenNebula in order to provide commercial services from OpenNebula software. The growth stage for OpenNebula can be considered as an interactive stage because channels for networking between participants were opened. These channels were the mailing list and the development portal. As a result, the community of participants was increased and the software was rapidly evolving into multiple stable official releases. It is believed that this stage influenced the analysis in this thesis because most data collected were extracted from the mailing list and the development portal of OpenNebula.

6.2.1 Start-up Stage

In 2008, OpenNebula as a software project received funding from "the European Union's Seventh Framework Programme (FP7/2007-2013)" under the following grant agreement: "RESERVOIR– Resources and Services Virtualization without Barriers, 2008-2011, EU grant agreement 215605".

Based on the agreement of the programme, the two founders of OpenNebula, with the other two software developers who worked with those founders, were introduced to different business cases in the European market. They took centralised decisions to develop OpenNebula software according to the market needs.

At this stage, the development portal of OpenNebula was declared as the official web page to store and develop OpenNebula software. In addition, the first official upgrade[3] open source release (Release 1.0) was published in July 2008 on the development portal of OpenNebula.

[3] An upgrade release is software that implements changes and features to OpenNebula's source code.

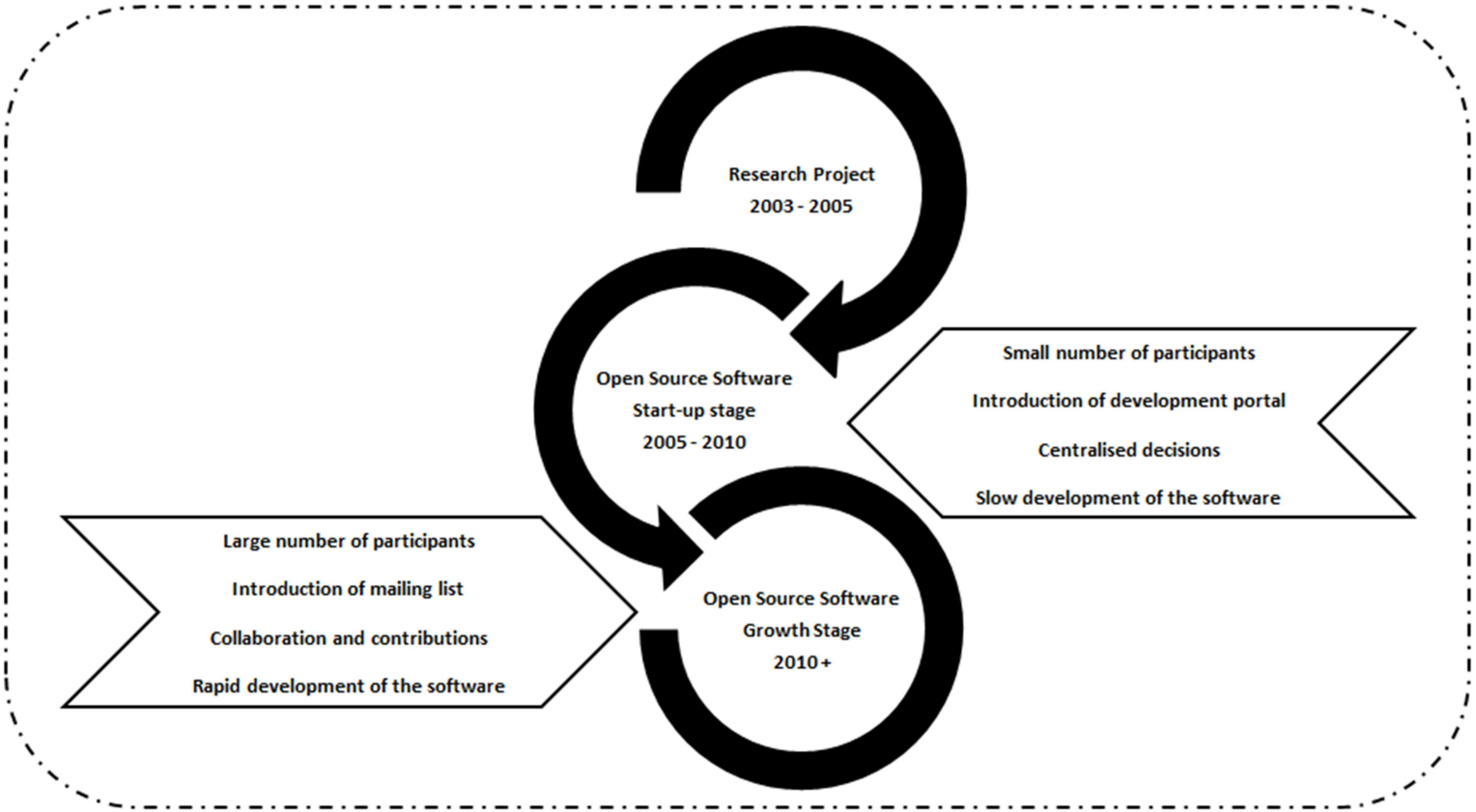

Figure 6.1 The different stages of OpenNebula

25 participants registered on OpenNebula's development portal and contributed to this official release by adding requests to resolve software bugs they found. As a result, two update[4] releases (release 1.2 and release 1.4) were publicly published on the development portal of OpenNebula in February and December 2009, respectively.

6.2.2 Growth Stage

Several important declarations were announced at this stage. Alongside with the development portal of OpenNebula, mailing list was declared as the official channel in which participants can communicate and share their ideas, concerns and questions. In 2010, 'OpenNebula Systems' (formerly called 'C12G labs') was officially declared as the official firm responsible for the commercial development of the OpenNebula software package. Thus, OpenNebula is being spinout open source software.

A total of eight employees are working in 'OpenNebula systems' and they are referred to as 'Core members' in this thesis. There are other participants who are registered on the mailing list and the development portal of OpenNebula. Some of those participants are private actors, employees in companies in different industrial sectors: finance, banking, telecommunications, government, the media, academia and

[4] An update release resolves bugs and implements software features, service packs, and patches within the source code.

research, aerospace, e-infrastructure, Software as a Service (SaaS), and more.

OpenNebula, as open source software, witnessed a rapid growth in its software development (as seen in figure 6.2). OpenNebula had four additional stable releases: 2.0, 3.0, 4.0, and 5.0. Each release has boasted subsequent updates and maintenance releases that have improved the functionality required for that release. OpenNebula's four stable releases served to improve OpenNebula's functionality and created additional customized complement components.

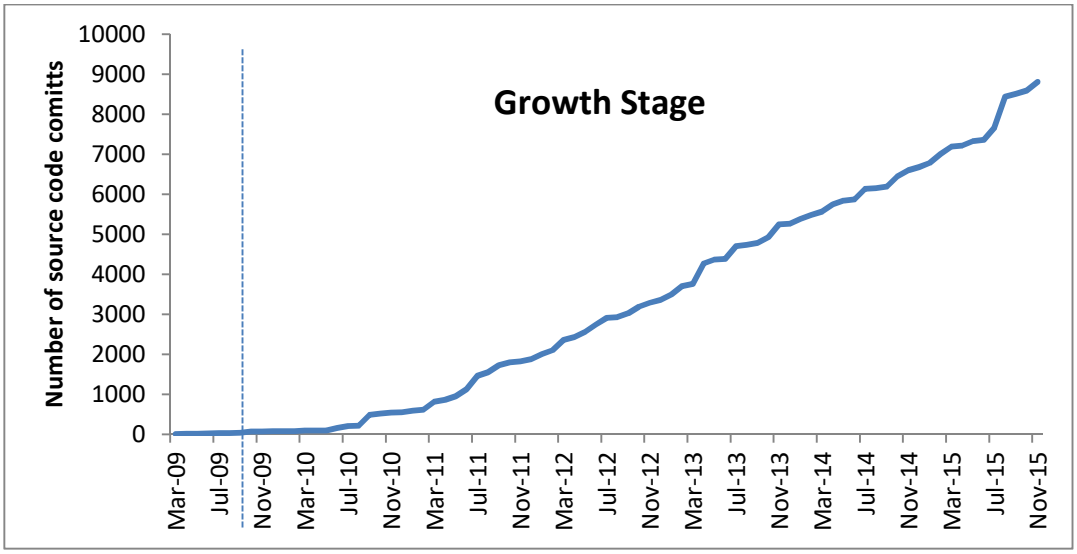

Figure 6.2 The development of OpenNebula Software in the growth stage

Feller et al. (2008) suggested that the majority of software analysed in literature bolsters commercial products relating to vendors or commercial firms. However, as stated in the official web page for OpenNebula, OpenNebula is vendor agnostic software; the software has integrated various complement components related to competitive firms. KVM,

XEN, VMware, OpenVZ, and ESXi are examples of different competitive virtual machines that are integrated within core OpenNebula. In addition, Ubuntu, Debian, OpenSUSE, and CentOS are examples of different competitive operating systems that are integrated with core OpenNebula.

Currently, OpenNebula is expanding within the business market. For example, 5,000 downloads per month were recorded in 2012; in addition, a mass scale production deployment was recorded in 2014. For example, more than 200,000 virtual machines were deployed across different businesses. OpenNebula is one of the most successful open source software in the cloud computing market.

Data gathered in this stage from the mailing list and the development portal are analysed. As a result, it is understood that all participants are communicating together and private actors are contributing to the development of the software over time. OpenNebula is a unique case study that would definitely serve in conducting and answering the research question.

6.3 The Private Actors in OpenNebula

In this section, a map for participants in OpenNebula is identified. In addition, it is found that the majority of participants in OpenNebula are private actors. Moreover, it is found that the community of private actors in OpenNebula is heterogeneous; they belong to different companies in different industrial sectors. Following the literature, such a

community of participants would face the business dilemma and accordingly we can anticipate that they would withdraw their participations. However, as will be discussed in section 6.4, those private actors are actually participating in OpenNebula.

6.3.1 The registration process

The source code for OpenNebula is available online on the official OpenNebula web page. Anyone interested in downloading, modifying and combining the software is allowed to do so without being required to contribute back to the public.

However, it is stated on the OpenNebula web page that participants cannot participate in OpenNebula unless they register on OpenNebula's web page and development portal (details about participation in OpenNebula will be discussed in the following section). Participants in OpenNebula are officially registered on both the OpenNebula web page and the OpenNebula development portal. The registration process involves submitting credentials for those participants.

The credentials required by OpenNebula's mailing list include an email address, a username, and password as mandatory (refer to figure 6.3).

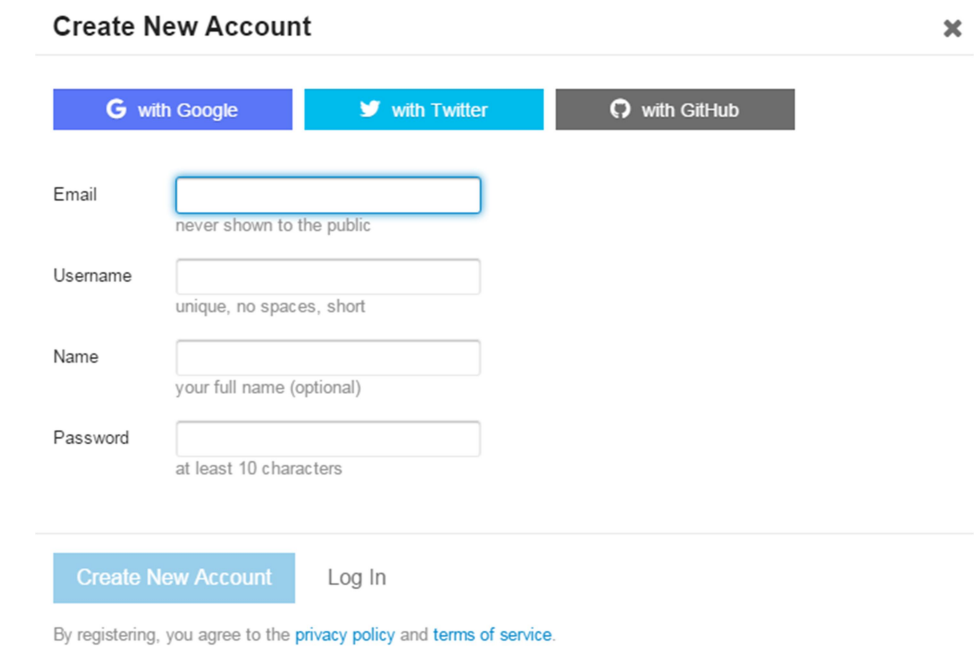

Figure 6.3 Screen for registering new participant to the mailing list in OpenNebula

Credentials that are required to register for the development portal of OpenNebula are: login, password, confirmation of password, first name, last name, email address, and language (see figure 6.4).

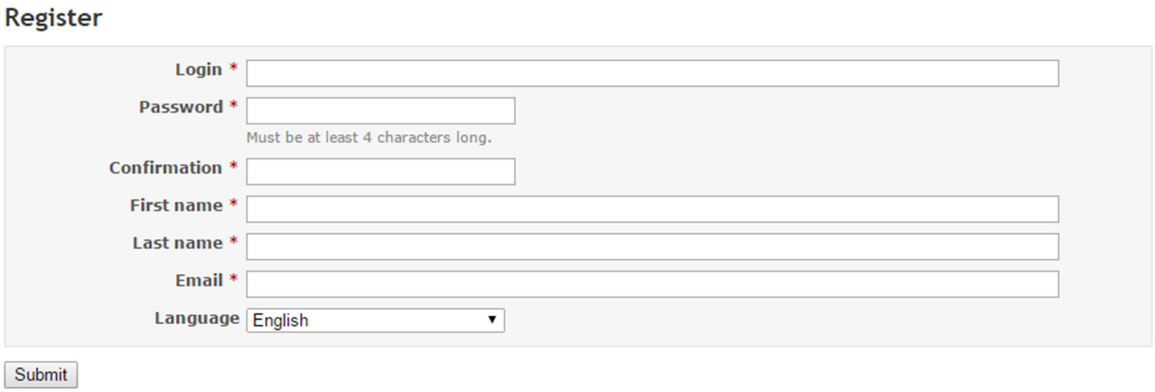

Figure 6.4 Screen for registering new participant to the development portal in OpenNebula

Until the end of February 2017, the number of registered participants on the mailing list and the development portal were 1,344, 916, and 49 respectively.

Email addresses were employed in this research, as explained in the methodology chapter, as a first step to classify the most active participants on the mailing list by referring to their email affiliations.

For example, a participant with an email address that ends with an '.edu' email extension is classified into the 'education email affiliation' group. As another example, a participant with an email address that ends with an '.opennebula.org' email extension is classified into the 'core members' group. Up tol November 2014 for example, there were 8, 546, 68, 16 and 706 registered participants belonging to OpenNebula, individual, educational, government and corporate email affiliations, respectively.

6.3.2 Participants mapping

Initially, 7,017 emails sent on the mailing list were gathered. These emails were sent by the top 25 participants on the mailing list. This consisted of five participants from 'core members', five participants from the 'corporation' affiliation, five participants from the 'individual' affiliation, five participants from the 'education' affiliation, and five participants from the 'government' affiliation (for more details, please refer to methodology chapter). These emails were analysed for the purpose of identifying these participants sending emails on the mailing list. This is beneficial because it allows for the identification of the different participants in OpenNebula as described in figure 6.5.

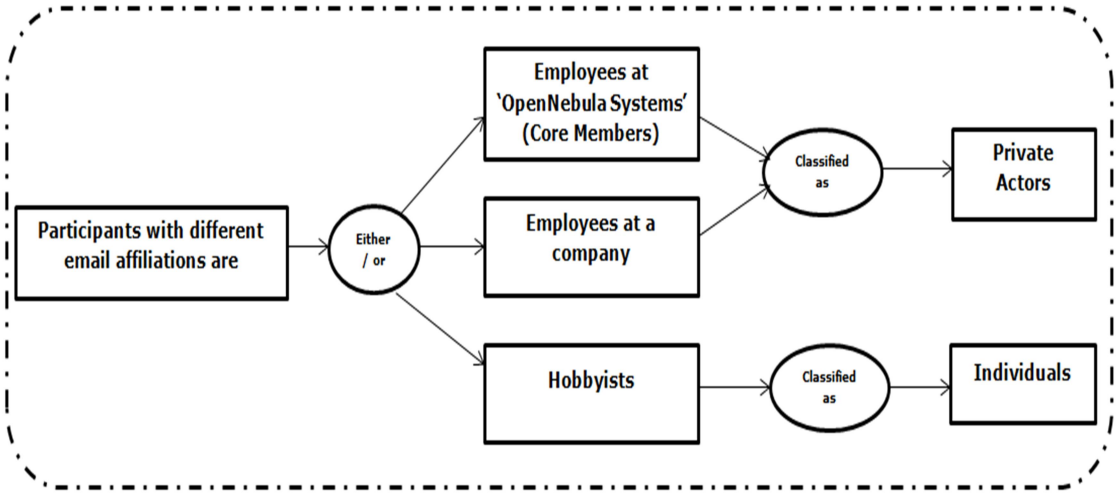

Figure 6.5 The different participants in OpenNebula

First, Core members are employees at 'OpenNebula Systems' company; this company is the official sponsor for OpenNebula. These participants respond to emails and resolve customer enquiries in order to fulfil the needs of the customer.

For example, AnonyAG3, an employee in company X, sent an enquiry to the mailing list. AnonyAO3 responded to this email by opening a live chat with AnonyAG3 and solved the problem.

5 out of 25 participants in the sample were identified as Core members and they were AnonyAO1, AnonyAO2, AnonyAO3, AnonyAO4, and AnonyAO5.

Second, there are participants who are employees at companies and they are implementing OpenNebula software into their IT business. However, these participants encountered technical problems when attempting to implement OpenNebula. Thus, they shared information with other

participants by sending emails through the mailing list. Information was contained in their computer network, log files, screenshots, and error messages.

For example, AnonyAG2 sent an email that contained information regarding the computer settings he was using to access OpenNebula from the internet, such as OpenNebula release 3.2, the 'X509' method for secure authentication, and the 'EC2' tool. He also requested technical assistance to resolve an error message that appeared while using OpenNebula in his work.

16 out 25 participants in the sample were identified as employees at companies: AnonyAI1, AnonyAI3, AnonyAI5, AnonyAG1, AnonyAG2, AnonyAG3, AnonyAG4, AnonyAG5, AnonyAE1, AnonyAE2, AnonyAE3, AnonyAC1, AnonyAC2, AnonyAC3, AnonyAC4, and AnonyAC5.

Third, hobbyists are participants keen to respond to enquiries rather than sending their own technical enquiries. However, these participants are not core members of OpenNebula. They want to share their knowledge with other participants and learn from them. 74 per cent of emails came from the category AnonyAI2; these emails revealed responsive behaviour when solving technical enquiries from different participants.

It was discovered that participants utilise their technical experience, both within and outside of OpenNebula, to distribute knowledge. Participants responded to technical

enquiries regarding the ideal implementation of OpenNebula. These participants understand that there is no specific solution for implementing OpenNebula; various options exist. Therefore, they depend on the exchange of information between participants and then provide their opinion and technical advice.

For example, a participant sent an email through the OpenNebula mailing list explaining the different virtual machines utilised in his firm. In addition, he explained the implementation problem he encountered when using virtual machines alongside OpenNebula software. AnonyAI2, a possible knowledge sharer, commenced discussions with the participant in order to understand the network settings that had been implemented. AnonyAI2 suggested solutions that might solve the implementation problem, but insisted that some solutions might cause flaws within the firm's computer network (based on his own experience with the same implementation problem).

4 out 25 participants in the sample were identified as Hobbyists: AnonyAI2, AnonyAI4, AnonyAE4, and AnonyAE5.

Understanding participants in OpenNebula as core members, employees at companies and hobbyists helps in classifying those participants, as suggested in the literature, into individuals and private actors (see table 6.1). Lerner and Tirole (2001) suggested that participants in open source software can be classified as individuals or private actors. Individuals are programmers who are participating in open source software for learning and pleasure while private actors are employees who

are participating in open source software in order to gain private benefit from the software.

Table 6.1 Classifying participants of OpenNebula into individuals and private actors

	Individuals	Private Actors
Core member		✓
Employee at a company		✓
Hobbyist	✓	

Classifying participants as individuals and private actors is beneficial in this chapter as it implies that there is a private interest in OpenNebula. This is crucial because , in general, private actors are key contributors to open source software; their contributions are deemed crucial (Casadesus-Masanell and Llanes, 2011) to guide the open source software both in monetary terms and strategically. According to the von Hippel and von Krogh (2003) 'private-collective model of innovation', private interest are key actors who are investing their private resources for the collective benefit.

In addition, such classification is beneficial in this chapter as it implies that the majority of participants in the chosen sample are private actors; 16 out of 25 participants are private actors. This finding guides the analysis to see how similar or different those private actors are.

Therefore, email addresses of all participants who are registered on the mailing list are collected. Referring to those email addresses, it is found that up to the end of 2015, the majority of registered participants have corporate email affiliation – 699 registered participants. In addition, those

participants are employees at competing companies across different industrial sectors. For example, as seen in figure 6.6, different registered participants are employees at 70 companies. Those companies operate in the technology industry but belong to 6 different sectors such as supercomputing, cloud products, information technology, etc.; companies that belong to the same sector can be competitors. For example, IBM and DELL are competing companies that provide the market with different technology devices.

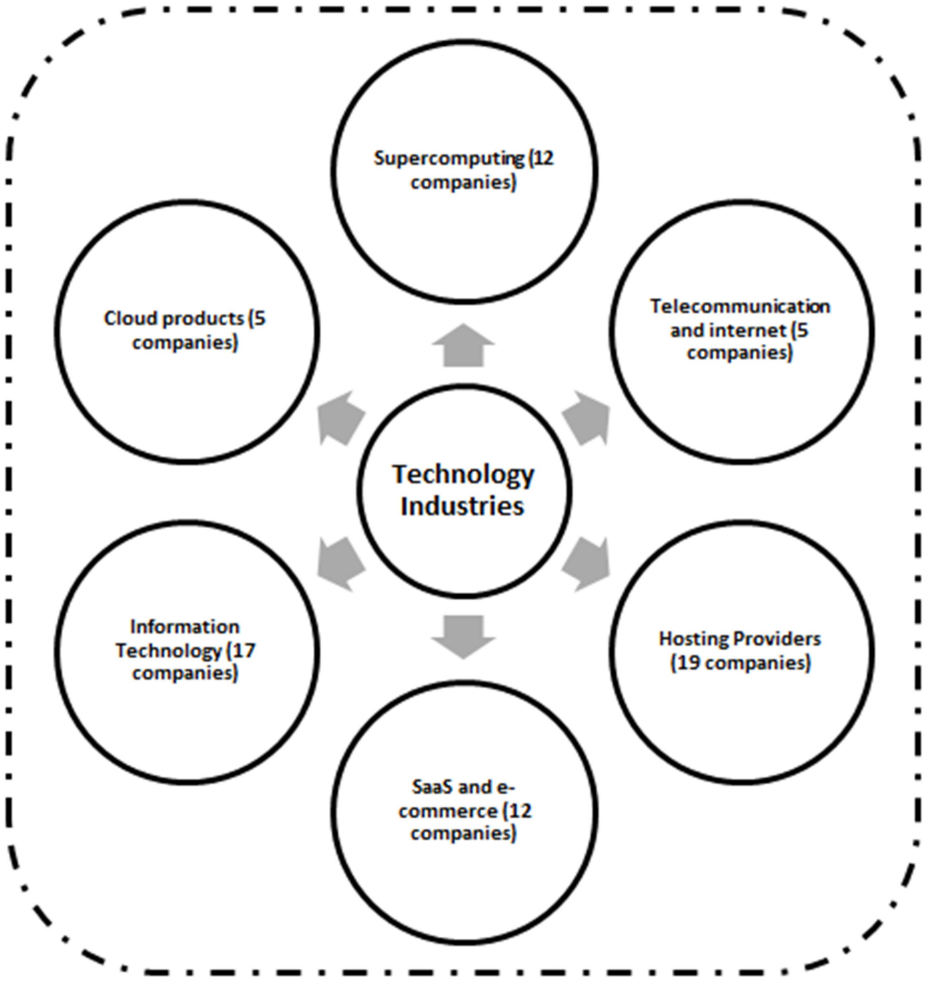

Figure 6.6 Rivals in the technology industry

In another example, as seen in figure 6.7, different registered participants are employees at 21 companies. Those companies operate in the education industry but belong to two different sectors: academic and research. Companies that belong to the same sector can be competitors.

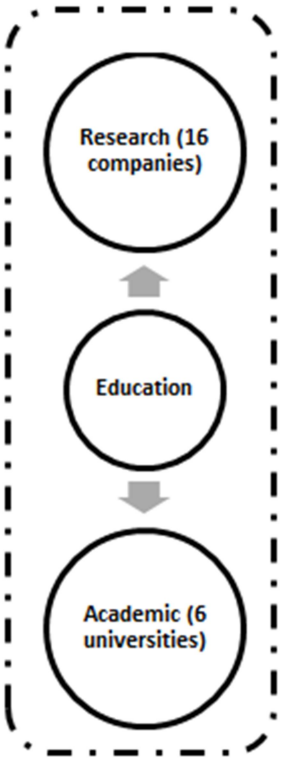

Figure 6.7 Rivals in the education industry

The von Hippel and von Krogh (2003) 'private-collective model of innovation' suggested that contributions of private actors are crucial to open source software but heterogeneity in OpenNebula includes rival companies who are competing with each other. Thus, it is interesting to understand whether such heterogeneity affects their contribution, especially in that heterogeneity includes rival companies so contributions may affect their competitive advantage if rivals decide to free ride

instead of collaborate and contribute. Answers to this question are explained in the following section.

6.4 Contributions in OpenNebula

Private actors and individuals, the heterogeneous community in OpenNebula, are registered in OpenNebula in order to be able to participate. It is found that they can participate in OpenNebula by (1) sending and replying to emails on the mailing list and/or (2) adding 'requests for developing software in the development portal.

6.4.1 Communications on the mailing list

The same sample of emails collected earlier was used again. However, the analysis this time involved coding all data on a line-by-line basis in order to understand:

(1) Among the heterogeneous community, who is sending and who is replying to emails?
(2) What is the benefit from participants' communicating through these emails?

In an attempt to answer the first question for this stage of the analysis, it was revealed that private actors (specifically employees at companies) mainly started by sending emails and the reply emails were not exclusive to any type other than others.

Those private actors started their emails by introducing themselves and their businesses. They usually explained that they implemented or wished to implement OpenNebula software within their IT business. They shared fine information

about the technical network for their IT businesses and the technical configuration for OpenNebula within their business. They also shared log file and error messages in their emails. Accordingly, they sent emails in order to ask questions related to the software and its implementations, to report software bugs, to ask for advice, to suggest and develop software features …etc.

Core members were found to reply to almost all emails sent by different participants in OpenNebula. They clarified ideas, explained source code, provided suitable documentation, debugged software bugs and provided technical advice. West and O'Mahony (2005a) explained that core members usually spent their time and efforts with other participants in order to attract those participants to participate in the software.

However, it was also found that not only core members replied to the different emails on the mailing list, but also hobbyists and other private actors were found to reply to emails and provided technical help for requesters. For example, AnonyAC1 is one of the most active participants in OpenNebula who sent many emails on the mailing list. Analyzing his communications through the mailing list reveals that he sent 230 emails for the duration September 2010 through September 2014. Through these emails, he raised 35 different technical topics related to technical problems that he faced while implementing OpenNebula software. 21 registered

participants and 4 core members replied and communicated with him through different emails. As a result of their communications, software bugs were solved and faulty documentation was amended.

All types of participants in OpenNebula communicated by sending and replying to emails on the mailing list (refer to the first question in this section). Ostrom and Walker (1991) suggested that if users of a public resource voluntarily decided to communicate with each other, then the level of cooperation between them would increase. As a result, communication was found to benefit those users in different ways, such as allowing them to contribute to each other and solve the dilemmas that they might be facing (Cardenas et al., 2004). Based on communications of the mailing list, the analysis revealed the different types of contributions by the different participants that were identified in OpenNebula as follows.

First, emails and memos were examined. Then, texts that were thought to represent contributions were highlighted and positioned within different 'categories'. For example, in one of his emails, AnonyAO2 explained the technical implementation of the 'Libvirt' system in OpenNebula, including the installation steps and the network connectivity required for integrating the system with OpenNebula. Thus, his email was highlighted and placed in the category 'explain possibilities for system implementations within OpenNebula'.

Some of the categories proposed, however, were deemed redundant and overlapped with one another. As a result, categories were re-examined and merged in order to create more accurate groups.

A list of categories was created and presented in an organised manner, as suggested by Thomas (2006). As seen in table 1 in appendix Empirical 1, each category possesses its own label, detailed description, and text gathered from emails and memos. A list of 75 categories was created based on the analysis of 7,017 emails sent from 25 different participants (as explained in the methodology chapter).

Second, similar ideas were grouped together in categories and labelled under a specific 'code' that represents a contribution.

As a result, 4 main contributions were identified (see details in Appendix A):

1. Modify available documentation: participants contribute by amending incorrect information that exists in the available documentation for OpenNebula.
2. Report software bugs / features: participants report software bug reports that exist in OpenNebula software / participants suggest software features to be added into the original OpenNebula software.

3. Solve software bugs: participants share source code that solves software bugs that already exist in OpenNebula software.
4. Develop software features: participants share source code that adds and integrates new software features into OpenNebula software.

These contributions are supported by the communications between different participants in OpenNebula.

For example, participant 'AnonyAI1' was a private actor (employee at a company) of OpenNebula. He sent over 180 emails from 2009 to 2012. He started his early emails by introducing himself, his business, and the way he implemented OpenNebula within his business. For example, he described the types of systems used in his business, the computer network in his business and the releases of OpenNebula he was implementing.

Through several emails, he raised questions that would help him and other participants to understand the logic of OpenNebula software and the commands used for OpenNebula screens and interfaces. For example, he sent several emails describing technical problems that he faced while implementing a special type of virtual machine offered by OpenNebula software called ESXi. Core members, on the other hand, asked him to report the different software bugs that he faced while implementing OpenNebula by adding

requests into the development portal (this represents the ‘report software bug’ contribution).

With the help of core members, employees at companies and hobbyists, he could overcome these problems. And AnonyAI1 was found to share complete versions of these solutions on the mailing list and share updates for related documentation accordingly (this represents both ‘solve software bug’ and ‘modify available documentation’ contributions). For example, there was a problem with the parameters used in ESXi virtual machine and that was behind the software bugs that he reported. So AnonyAI1 shared the correct parameters with others through his emails and highlighted that these new parameters shall be corrected in the documentation.

Based on these communications, AnonyAI1 was able to successfully integrate OpenNebula within his business. After a while, AnonyAI1 started to suggest the addition of new features into OpenNebula software in order to help him in advancing his business (this represents the ‘report software feature’ contribution). For example, he suggested the addition of authentication methods in OpenNebula. He explained that these methods were important to ensure the security of data while implementing OpenNebula in his business.

Many participants supported his suggestion and accordingly he added his suggestion as a request in the development portal. He started his cooperation with core

members to develop and add the new source code into OpenNebula software (this represents the ‘develop software feature’ contribution).

As seen from the previous example, AnonyAI1 contributed to OpenNebula software based on his communications on the mailing list in different ways. Of course, not all private actors contribute in the same pattern. For example, AnonyAI3 contributed by suggesting a software feature but not developing the feature. AnonyAI3 suggested, through his email on the mailing list, that the development of a new driver called ‘OpenVZ’ should be added to the source code of OpenNebula (this represents the ‘suggest software feature’ contribution). After discussions with other participants on the mailing list, AnonyAI3’s added a new request to the development page. As a result of this, the ‘OpenVZ’ source code was developed and implemented within OpenNebula’s own source code by core members.

AnonyAI1 and AnonyAI3 are only examples of participants who were encouraged to contribute because of their communications with other participants. As shown in table 6.2, each type of participant in OpenNebula was found to perform a bundle of contributions through communications on OpenNebula mailing list.

Table 6.2 Contributions made by participants in OpenNebula

	Individual	Private actor	
	Hobbyist	Employee at a company	Core member
Modify available documentation	✓	✓	✓
Report software bugs/features	✓	✓	✓
Solve a software bug	Partially	✓	✓
Develop a software feature		✓	✓

Studying the 7,017 emails from the mailing list, as described earlier, revealed that participants communicated and contributed in different ways on the mailing list. After that, they passed their contributions to OpenNebula's development portal, where the source code is stored and developed, by adding requests. Participants can add requests into the development portal only after they log onto OpenNebula's development portal.

6.4.2 Requests additions in the development portal

The addition of new requests in the development portal is achieved when participants log on to the development portal and add new 'requests' to the development page in order to modify and enhance the source code. The addition of a new request to the development page requires (1) writing the name of the request and (2) adding a description regarding this request (see example in figure 6.8).

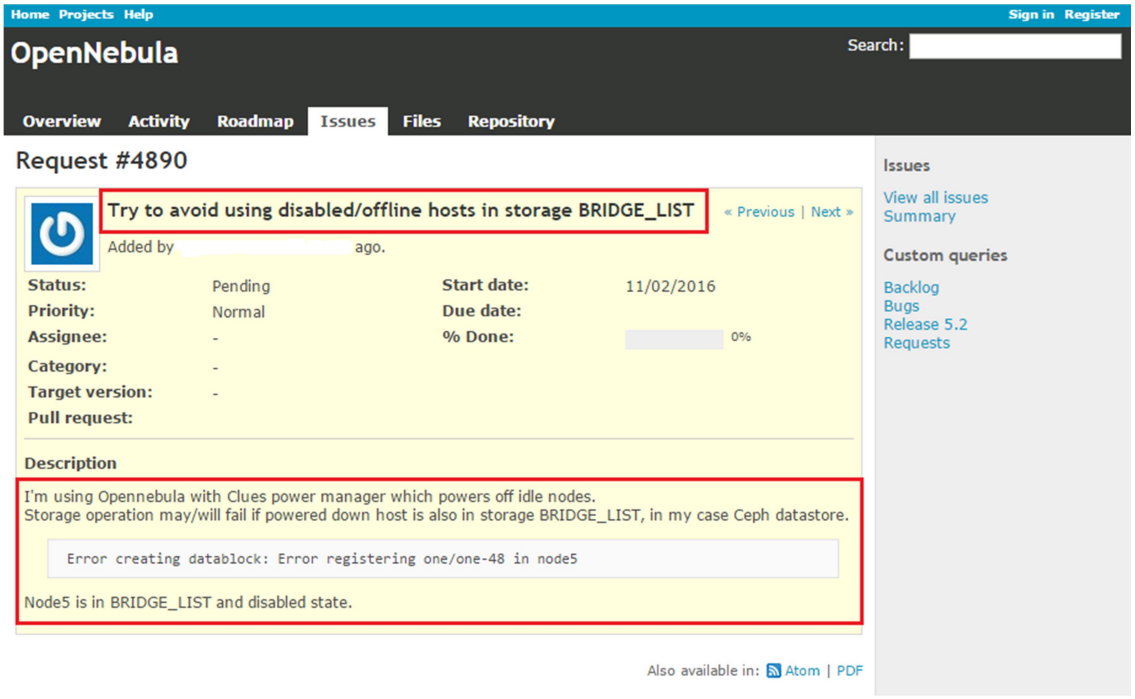

Figure 6.8 An example of a newly added request to the development page of OpenNebula

A total of 3,842 requests were added to the development page of OpenNebula between March 2009 and the end of November 2015 (these requests are part of the 7,017 emails sent on the mailing list, as described earlier). These requests were exported from the development page into an Excel sheet that was then added to NVivo 10 software and analysed.

It was found that nearly 75% of requests added into the development portal were added by private actors (the 21 participants in our chosen sample). And requests were added into the development portal over time (see figure 6.9). This indicates that there is a private interest in OpenNebula and that private actors are contributing to OpenNebula by adding requests into the development portal.

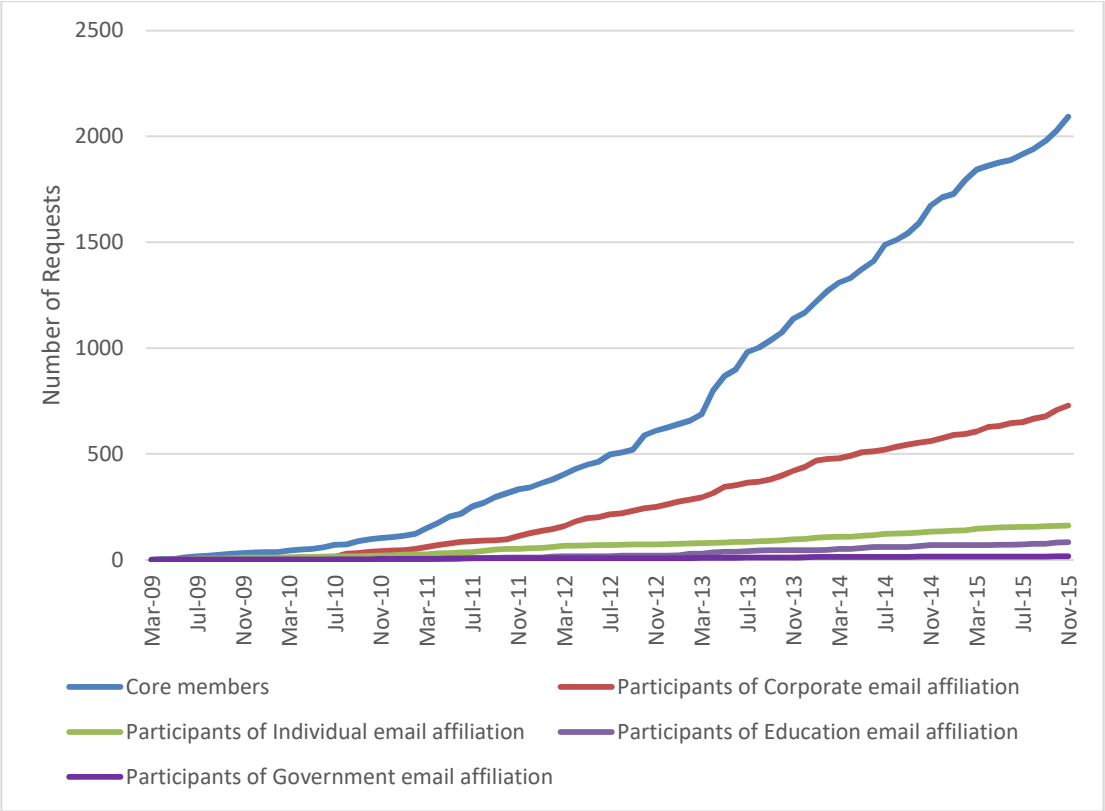

Figure 6.9 Number of requests added into the development portal of OpenNebula

From 2008 to the end of 2015, 3,842 requests were submitted to OpenNebula's development portal. Of these, 143 requests were duplicates and 381 were found to have no possible solutions (invalid requests). Thus, valid requests added to the development portal amounted to 3,318 requests. Table 6.3 shows that these requests are actually reflecting the different contributions they had on the mailing list as identified earlier in section 6.4.1.

Table 6.3 Contributions in the development portal of OpenNebula

Modify available documentation	**6% of requests added to the development portal are related to issues of modifying the documentation of OpenNebula**	**Majority are added by private actors (65% added by core members and 25% added by participants with corporate email affiliation)**
Report software bugs / features	54% of requests added to the development portal are related to issues of reporting software bugs	Majority are added by private actors (58% added by core members and 28% added by participants with corporate email affiliation)
	46% of requests added to the development portal are related to issues of reporting software features	Majority are added by private actors (75% added by core members and 19% added by participants with corporate email affiliation)
Solve software bugs	93% of added reported software bugs are solved by software patches[5]	Core members are mainly responsible for solving the requests. However, other 93 private actors (participants with corporate email affiliation) share source code to solve the software bugs
Develop software feature	73% of reported software features are developed by software patches	Core members are mainly responsible for solving the requests. However, other 77 private actors (participants with corporate email affiliation) share source code to solve the software bugs

[5] The term 'Software Patch' refers to source code that is able to resolve software bugs or develop additional features.

6.4.3 Contributions towards Collective Complementarities

As explained earlier, private actors are communicating through the mailing list and their communications lead to contributions. These contributions are: modifying available documentation, report software bugs/ features, solve software bugs and develop software features. And these contributions can be measured in terms of requests added by those private actors into the development portal.

These requests are causing the development of OpenNebula software (as will be discussed in detail in section 7.2). Examining the technical design of OpenNebula software, it was revealed that the software consists of core OpenNebula and other complements software; Core OpenNebula and complements software are integrated through a set of APIs as follows.

OpenNebula software is classified into 21 different components (the technical descriptions of these components are detailed in Table 1 in 'Appendix Empirical 2'). Schilling (2000) explained that a software that consists of components that can be combined together is technically described as a 'modular software'; each component is a module. Baldwin and Clark (2000) explained that modules can be combined through pre-specified interfaces. For example, these components in

OpenNebula are connected with each other using Application Programming Interfaces (APIs[6]).

Examining these 21 modules reveals that these modules can be classified into 'core' and 'complements' modules as described in table 6.4.

Each complement component is integrated with core OpenNebula via APIs and each complement component has different technology options provided (as described in figure 6.10).

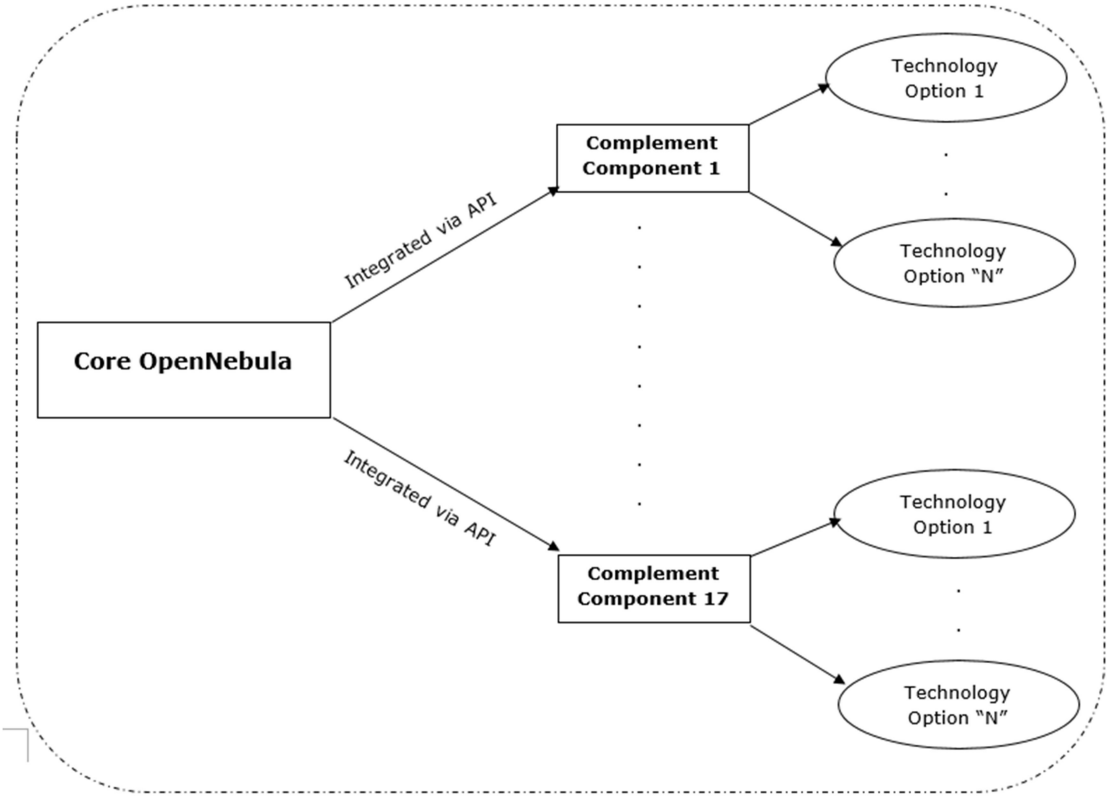

Figure 6.10 Technical design for software complements in OpenNebula

[6] APIs are pieces of source code that is responsible for ease of communication and transfer of information between computer components and applications.

Table 6.4 Core and Complements modules in OpenNebula

	Core	Complements
Function[7]	Modules that deal with the standard function for OpenNebula software which is managing virtualisation for data centres.	Modules that deal with customised features that can be implemented to serve customised needs for virtualised data centres. These modules represent the complements products that can be integrated with core OpenNebula software in order to enhance the applicability of OpenNebula software.
Characteristic	Mandatory (one needs to install these 4 components in order to implement the software)	Optional (it is up to the user to decide which component to install and implement according to the customised needs)
Number of modules	4	17

[7] Refer to table B.1 in Appendix B for technical details

These technology options have different licensing agreements; some are licensed under open source licenses and others under proprietary licenses. Despite these differences in their licenses, private actors choose to contribute to these complements components and share the necessary API in order to integrate the complements with core OpenNebula. For example, one of the complements components provided by OpenNebula is called a virtualization driver. There are many technology options provided by OpenNebula in order to use the virtualization driver such as XEN driver, KVM driver, OpenVZ driver, VMware driver and ESXi driver (see table 6.5). Table 2 in 'Appendix Empirical 2' provides other examples of technology options for complements components.

Table 6.5 The different technology options for virtualization driver (complements module) in OpenNebula

Complements Component	Description	Examples
Virtualization Driver	These drivers are software that virtually imitates a particular computing system (Stair and Reynolds, 2013)	XEN driver (Complement with open source license). For example, AnonyAI3 was concerned about participating in and understanding of networking and storage issues related to XEN virtual machines, understating commands, implementation, and documentation.
		KVM driver (Complement with open source license). For example, AnonyAG3 was reporting problems related to the implementation of "OpenNebula express" with KVM environment under an in-house developed operating system.
		OpenVZ driver (Complement with open source license). For example, AnonyAI3 developed and added OpenVZ source code and documentation to OpenNebula repository.
		VMware Driver (Complement with proprietary license). For example, AnonyAC3 solved different technical problem related to the implementation of OpenNebula 4.0 with VMware virtual machines only.

		ESXi Driver (Complement with proprietary license). For example, AnonyAI1 provided fixes to several bugs related to networking problem with ESXi virtual machine and adding revisions to the code repository of ESXi virtual machine.

So far, it is understood that private actors contribute to the development of OpenNebula software in the form of complementarities: (a) development of core OpenNebula and (b) integration of core OpenNebula with other complements software via different APIs.

Surprisingly, as shown in figure 6.11, core OpenNebula, complements software and APIs are all freely revealed to the collective and this makes OpenNebula as a case study deemed interesting as follows.

Private actors in OpenNebula are core members and employees at a company (refer to section 6.3 for details). Rivalry is high in OpenNebula, for example, core members have rivals within the cloud computing industry such as core members of OpenStack software, Apache CloudStack software, Eucalyptus software…etc; another examples are explained in section 6.3.2.

Core members basically provided 'core OpenNebula' software and employees at company basically integrated 'complements software with core OpenNebula'. In addition, both of them also contributed to whole OpenNebula software (both core and complements) by: modifying documentation, reporting software bugs/features, solving software bugs and developing software features.

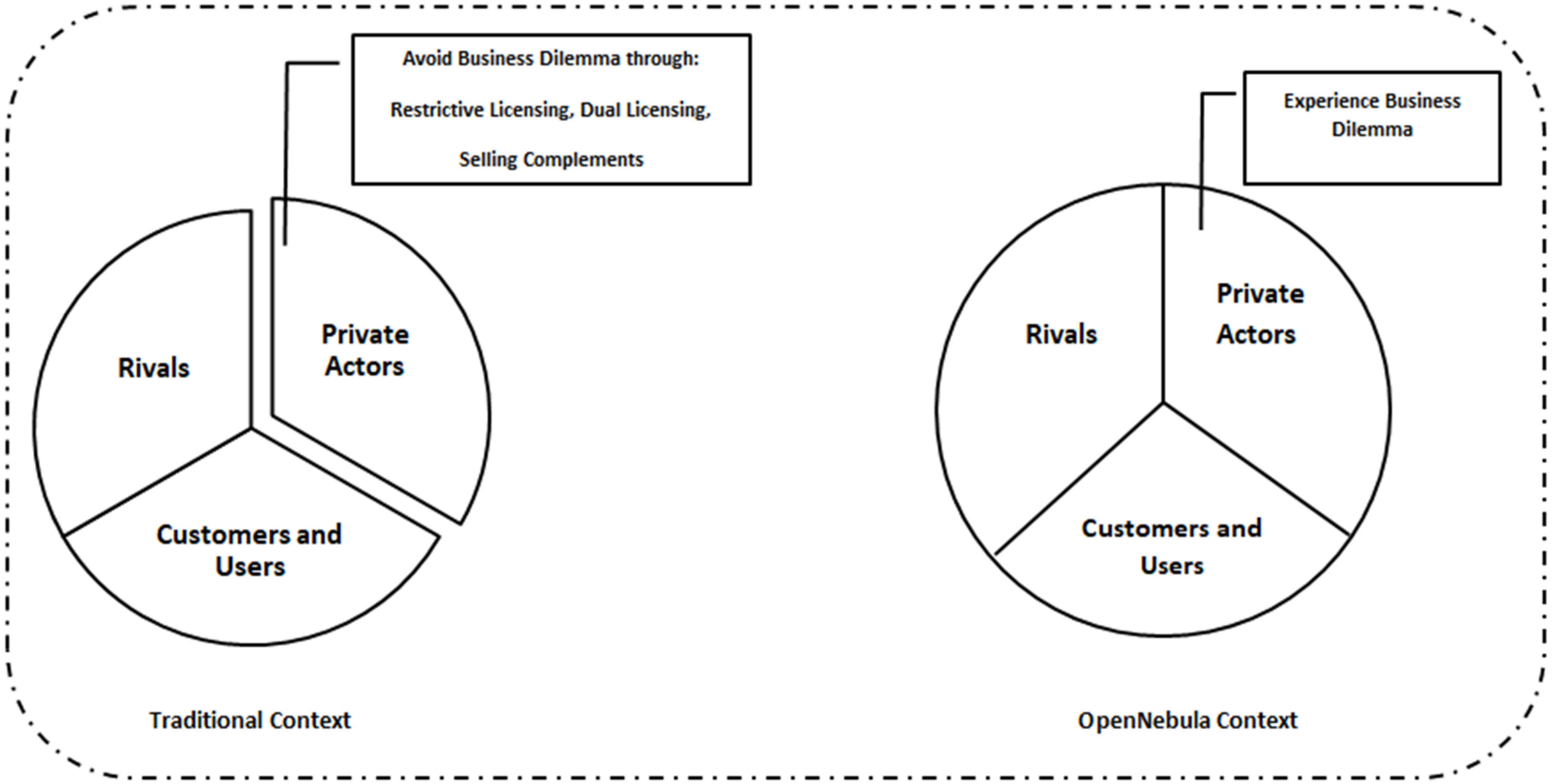

Figure 6.11 The difference between OpenNebula and the traditional context

Core OpenNebula and other integrated complements are private knowledge. They are lines of source code that are considered as an intellectual property for private actors. Therefore, private actors would be facing a business dilemma in this case. Private actors are encouraged to contribute and share their private contributions in open source software because they would reap rewards (as suggested by Von Hippel and Von Krogh (2006)) such as increasing the development skills of employees and gain faster feedback from customers and users.

However, their competitive advantage would be decreased because imitation by rivals would be easy. Rivals can imitate the software by copying the software and modifing it according to their organizational settings. In addition, private actors would be losing their profit from innovation because profits would be shared between private actors and rivals (imitators). Both private actors and rivals would be providing similar software to the market. Customers and profits would be shared accordingly.

In the traditional context of open source, private actors are encouraged to contribute and share their private contributions only if they avoid the business dilemma: mitigate imitation.

First, core members avoid imitation of their core software by rivals through revealing the software under restrictive licenses such as GPL. Core members are encouraged to share their private contributions because restrictive licensing is

protecting them from imitation; any modifications done to the software must be (by law) revealed back to the collective.

Second, core members avoid imitation of their core software by rivals through revealing the software under dual licensing (West and O'Mahony, 2005a). Core members are encouraged to share their private contributions because the software is segmented into two versions: community version and the commercial version (Comino and Manenti, 2011).

For the community version, any modifications done to the software should be revealed back to the collective. For the commercial version, any modifications done to the software should be done through monetary contracts between core members and the other participants.

Third, employees at companies avoid imitation of their complements software by rivals through selling (rather than revealing) the complements and related APIs. In this case, private actors are encouraged to share their contributions to the collective core software rather than reveal their private knowledge about complements software and related APIs (Gruber and Henkel, 2006).

In OpenNebula, it is found that core members and other private actors choose to contribute and share their private contributions to the collective while experiencing the business dilemma for the following reasons.

First, Core members disclose core OpenNebula under a permissive Apache v2.0 license without restrictions on commercial use. For the data collected for this thesis, from

March 2008 up to the end of November 2015, there were found to be 70 different software releases for OpenNebula software. As shown in figure 6.12, these software releases are revealed to the collective through time.

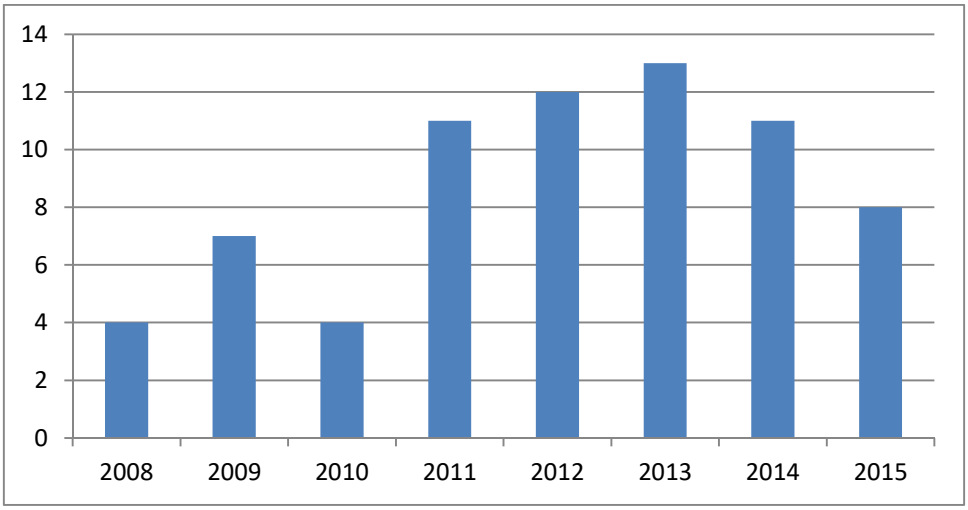

Figure 6.12 Number of software releases in OpenNebula

As discussed in section 6.4, the private contribution from private actors caused the development of these software releases. In addition, all of these releases contained developments and enhancements for core OpenNebula, complements and related APIs.

And all of these releases are released in the official development portal of OpenNebula under an Apache v2.0 license. This license is not restrictive and has no restrictions on commercial use of OpenNebula.

Second, other private actors disclose their complements software and related APIS to the collective under an Apache v2.0 permissive license without selling their integration contributions. Core OpenNebula, complements software and related APIs are all considered as private knowledge that are

freely contributed by the different private actors in OpenNebula.

Private actors in OpenNebula are encouraged to contribute and share their private knowledge with the collective despite the business dilemma they experienced, especially with the existence of rivals. This makes OpenNebula interesting as a case study.

6.5 Conclusion

The analysis in this chapter revealed that participants of OpenNebula are core members and private actors (refer to section 6.3 for details). Private actors are entering in an active communications (with their knowledge, ideas and experiences) with others (refer to sub-section 6.4.1 for details). They introduced themselves and their preferences. They talk, discuss, argue and contribute to OpenNebula software in different ways (refer to sub-section 6.4.2 for details).

As a result of communications, private actors seemed to contribute to OpenNebula (modify documentation, report software bugs/ features, solve software bugs and develop software features). They share their private contributions by adding requests in the development portal of OpenNebula and lead for the development of OpenNebula software in the form of collective complementarities (refer to sub-section 6.4.3 for details).

The development of OpenNebula software in the form of collective complementarities suggested that private actors in OpenNebula did not only focus on their private interest. They also

cared about the collective interest; they were able to lock their private software with OpenNebula software (this implied that they were able to lock their private interest with the collective interest).

Therefore, and informed by Ostrom (1990) evolutionary theory of collective action, it is proposed that the active communication by private actors is a prerequisite through which private actors can bridge their private with the collective interest. Hence, aligning the collective interest with the private interest through 'active communications' can be the first explanation provided in this thesis for how participants in open source software would privately invest in the software without creating the business dilemma.

Chapter 7 The Collective Software

7.1 Overview

This chapter explains how private contributions contribute to the development of the collective software. It theorizes how a transformation process is used to transfer contributions into collective software. In addition, this chapter proposes that the transformation process, through focusing the attention of OpenNebula participants, encourages private actors to reveal their private contributions to the collective.

The presentation of results in this chapter is provided in two sections. Section 7.2 explains the nature of the contributions provided by the private actors. Section 7.3 describes the transformation process. Section 7.4 theorises the impact of the transformation process in encouraging the private contributions in OpenNebula.

7.2 The Nature of requests

A huge amount of requests were added into the development portal of OpenNebula through time. A total of 3,483 requests were extracted from the development page for the period of March 2009 to November 2015.

Those requests were added by different participants of different email affiliations (see figure 9 in previous chapter); nearly 75% of those requests added into the development portal were added by private actors identified earlier from the 21 participants in our chosen sample.

Participants who add a request need to complete a template provided in the development portal of OpenNebula.

"You can report a bug by opening a new issue in GitHub OpenNebula project. You have to complete the template section for bug reports… You can make a feature request by opening a new issue in the GitHub OpenNebula project. You have to complete the template section for feature requests."(OpenNebula Official Webpage)

In the template, the participant is requested to determine the type of the request, the affected OpenNebula release and a description about the request. A request type is either a software bug or a software feature. A software bug is an error or a flaw found in the source code. A software feature is a new piece of source code that enhances performance, functionality, security and/or scope of existing source code.

Figure 7.1 shows that different requests to report software bugs and to add software features have been added into OpenNebula releases. Therefore, participants who added requests are contributing by 'reporting software bugs/ features' (refer to section 1.4 for details about types of contributions in OpenNebula).

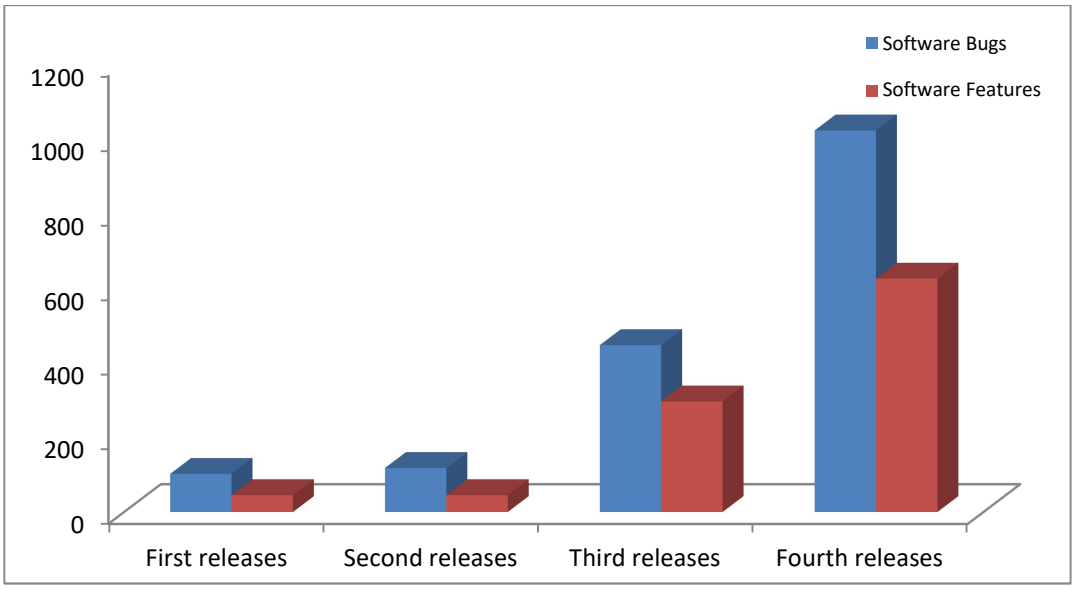

Figure 7.1 Number of 'software bugs' and 'software features' requests added to different OpenNebula releases

However, those different requests were found to be internally analysed by core members as those requests were found to vary in their needs and some could not be accepted in OpenNebula.

*"After an internal analysis, a **Bug** issue can be:*

- ***Pending**, needs to be verified*
- ***Accepted**, the bug has been verified and a priority assigned based on its severity (Low, Normal, High)*
- ***Closed**, the issue is fixed, could not be reproduced (worksforme) or duplicates another one*

*After an internal analysis your **Request** issue is categorized and will be **Pending** in the **Backlog** till:*

- *It is decided that is not in the scope of the project and **Closed***
- *It is interesting for the OpenNebula community and will be added as **Accepted** in the **Backlog**" (OpenNebula Official Webpage)*

Therefore, the analysis in this thesis is directed towards understanding the process under which those requests are

internally analysed and transformed into OpenNebula software under different releases.

7.3 The Transformation Process

As described in the previous chapter, contributions provided by private actors in OpenNebula can be measured by the requests added into the development portal. These requests were found to follow a certain process of transformation into freely revealed software. The process consists of different transformation periods that lead to considerable changes in the requests added into the development portal. These transformation periods transform requests: from raw requests into valid requests; from valid requests into selected requests; from selected requests into developed software (beta version); and from developed software into a freely revealed software in the OpenNebula development portal (as shown in figure 7.2). The result of these changes is the development of the collective software.

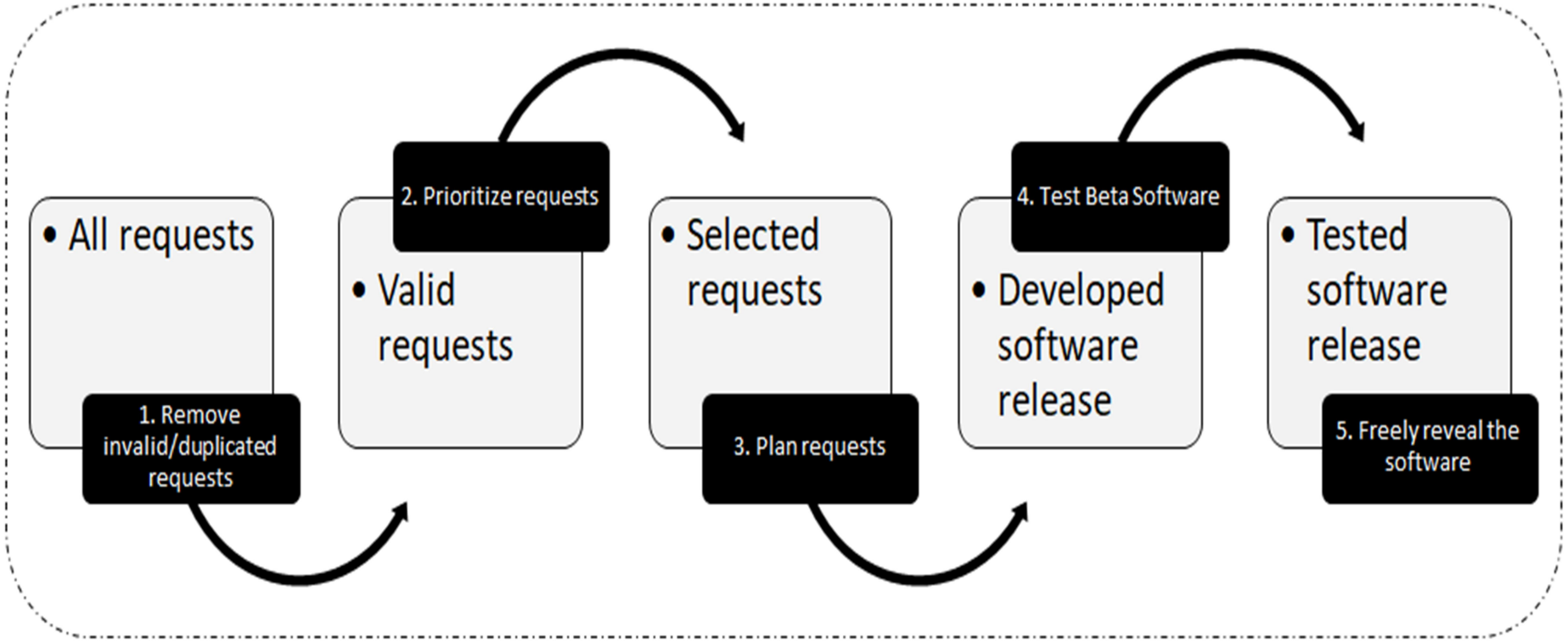

Figure 7.2 Transformation process in OpenNebula

7.3.1 Validation of Contributions

In the first transformation period, validation of requests, reported by participants, is conducted. Validating requests means filtering requests into valid, invalid and duplicated.

To start with, different private actors started their communications on the mailing list by reporting software bugs that they encountered while implementing OpenNebula within their business (refer to section 6.4 for details about the 'report software bugs' contribution). For example, at the early stages of implementing OpenNebula:

AnonyAI1 reported software bugs such as: node installation, networking issues, CPU related problems, commands execution failure, database malfunctioning, and trigger initiation failure.

AnonyAG2 reported software bugs such as: Difference between the documentation explanation and the actual output of "onevm cancel" command, Persistent image state error when deploying "onevm delete" command, and deploying econe-register command.

AnonyAG3 reported software bugs related to Non-interactive bash sessions.

AnonyAC5 reported software bugs such as: Bash processes in opennebula 3.2, Support for oneacct command does not exist, Cannot see images and virtual machines from ozones, Problems when connecting oneadmin in sunstone,

Deleting virtual machines keeps files on the front end, and Migrating onedb to opennebula 3.8.3.

After a while, those private actors became familiar with the technical implementations of OpenNebula. Thus, some of them started sending emails and adding requests in which they suggested ideas for integrating complements software with core OpenNebula (refer to section 6.4 for details about 'report software features' contribution). For example:

AnonyAI1 reported software features such as the integration of Contextualization with Windows VM, Monitoring driver, Hyper- V Driver, Redhat Cgroup, Redhat Enterprise Virtualization (RHEV).

AnonyAI3 reported software features such as the integration of OpenVZ virtual machine.

AnonyAC2 reported software features such as the integration of OpenNebula client packages only and Extending tm driver for ssh.

AnonyAC5 reported software features such as the integration of LVM2 transfer manager driver in opennebula 3.4.x.

As a result, a huge amount of requests (report software bugs requests and report software features requests) were found on the mailing list and also on the development portal. For the period from 2008 up to the end of November 2015, a total of 3,842 requests were added into the development

portal; 57% reporting software bugs and 43% reporting software features.

After that, core members remove[8] invalid and duplicated requests. Removing invalid requests by core members is based on their knowledge about needs and preferences of OpenNebula participants and users. For example,

Among the requests (reporting software features) added by AnonyAI1, core members responded on the mailing list that 'Contextualization with Windows VM' is the only valid request because it complies with the needs of users and the vision of the software.

Core members validate AnonyAI3 request (report software feature) to integrate OpenVZ virtual machine within OpenNebula.

Core members invalidate requests added by AnonyAC4 (reporting software features) because requests did not comply with users' needs. And some of these requests would develop extra bottlenecks for users.

In addition, removing duplicated requests includes re-directing these requests into the valid requests added into the development portal.

[8] Removing invalid / duplicated requests includes blocking any further discussions and development activities through the OpenNebula development portal.

“Our current approach to solve this is to "humanize" the value of memory using M,G suffixes... Closing this as it duplicates #182.” (OpenNebula Development Portal)

Almost all duplicated requests were re-directed into other valid requests; among the 143 duplicated requests only 3 of them were closed without re-directing.

Among the 3,842 requests added into the development portal for the period from 2008 up to the end of November 2015, 3,318 requests were identified as valid requests (as shown in figure 7.3). 54% were valid requests to report software bugs related to core and complements components and 64% were valid requests to add software features into core and complements components in OpenNebula.

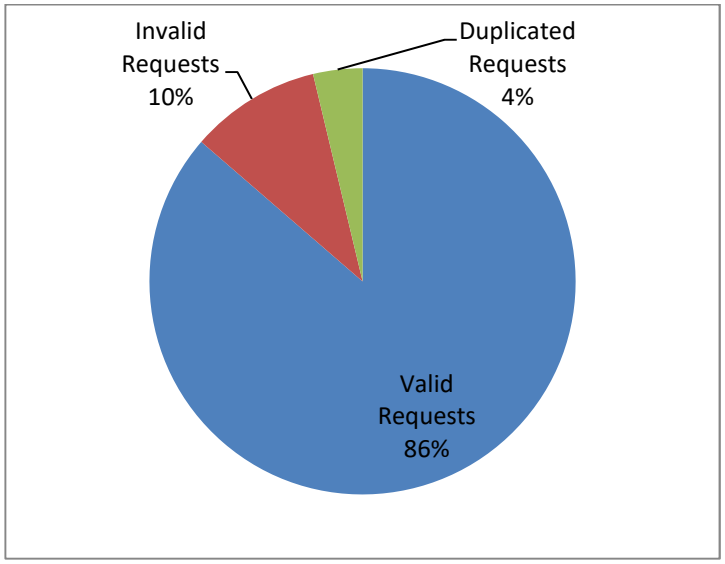

Figure 7.3 Percentage of valid, duplicated and invalid requests in OpenNebula

Having wealth in provided information will lead to “poverty” in human attention (Simon, 1994). This transformation period is beneficial because grouping the huge

amount of requests into valid, invalid and duplicated would direct the attention of private actors towards valid contributions only.

It is proposed that this period encourages private actors to report their suggested contributions because it allows private actors to quickly and precisely understand users' needs. It also allows private actors to concentrate their efforts on necessary contributions, instead of scattering the effort and time private actors would be contributing and sharing their private contributions towards users' needs.

For example, AnonyAI1 provides a consultation service to customers in the field of cloud computing. Because of his career, AnonyAI1 was able to report different software features to OpenNebula. However, AnonyAI1, through the validation of his requests, understood that OpenNebula users required integration of core OpenNebula with virtual machines rather than any other complements software. Therefore, AnonyAI1 was able to choose a highly demanded feature (which is 'contextualization of Windows virtual machine') and focus in its development.

OpenNebula participants would be better understanding their expectations from OpenNebula software and accordingly would be encouraged to share their private contributions.

7.3.2 Selection of Contributions

In the second transformation period, valid requests are prioritized according to the urgent and preferred needs of OpenNebula users and participants. Prioritized requests are determined based on IRC sessions and sponsorship programmes. Then, prioritized requests are selected in order to be incorporated in OpenNebula's next software release.

Core members usually allocate times for IRC session.

"These sessions are held from time to time and announced beforehand in the mailing list and other social tools." (OpenNebula Official Website)

Joining IRC sessions, OpenNebula participants talk and discuss their preferences about features to be incorporated into the next release of OpenNebula software.

"At the beginning of each release cycle we organize a IRC meeting or start a forum thread to discuss the requests for new features and for extending existing features. This valuable input to the planning meeting is used to create the short-term roadmap with the features that will be part of the release cycle." (OpenNebula Official Website)

In addition, sponsors of OpenNebula who supported the software development through sponsorship programmes (e.g. the 'fund a feature' programme and the 'champions' programme) can participate in the selection of requests to be incorporated in the next software release of OpenNebula.

High priority requests resulted from the discussions in IRC sessions and sponsorship programmes are selected in order to be incorporated with the next OpenNebula software release. Among the 3,318 valid requests identified for OpenNebula, a total of 2,437 requests were selected. 1,481 requests related to reporting software bugs and 956 requests related to adding software features were selected to be incorporated into the different releases of OpenNebula as detailed in figure 7.4.

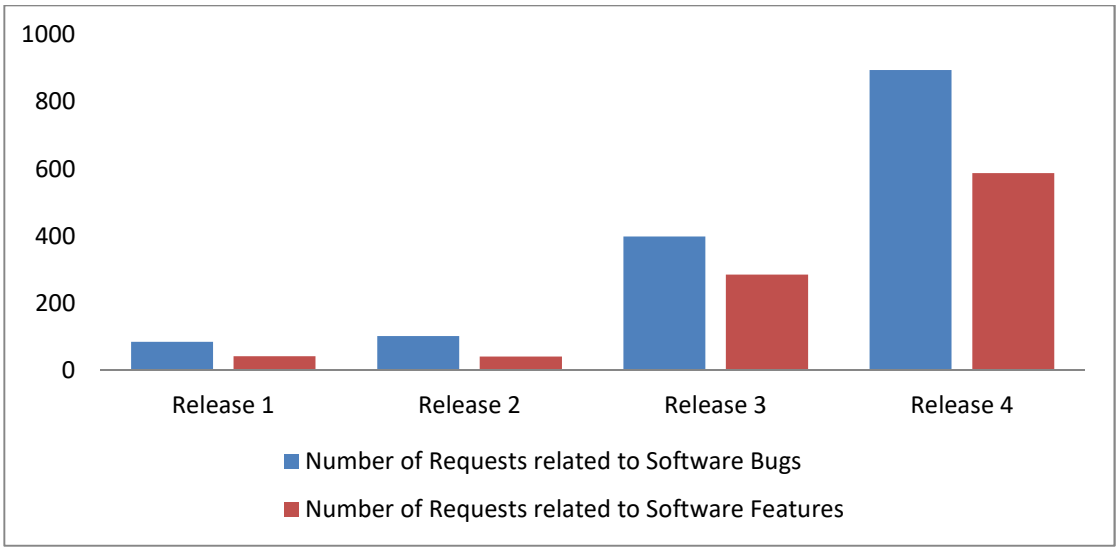

Figure 7.4 Number of valid 'software bugs' and 'software features' requests added to OpenNebula releases

Therefore, this transformation period is deemed beneficial because it allows private actors to push their valid requests (already identified in the validation period) forward. Private actors would be able to incorporate their valid requests into OpenNebula releases.

One interesting characteristic for this period is that the selection of requests is not biased towards a certain technology. For example, one of the items of complements software developed in OpenNebula is called ‘virtual machine’. Through time, different requests had been selected in order to develop virtual machines in OpenNebula that belong to different technologies such as KVM virtual machine, XEN virtual machine, OpenVZ virtual machine…etc.

In the market, these different technologies are competing in the industry of virtual machines. However, in OpenNebula, these technologies are linked together through APIs rather than competing with each other. It is found that participants and users of OpenNebula implement these different technologies alongside each other in order to enhance the performance of their overall information systems. For example,

AnonyAI2 was implementing XEN virtual machine with OpenNebula. Through time, AnonyAI2 faced technical difficulties with the XEN virtual machine. Accordingly, and using the available APIs, AnonyAI2 decided to implement another type of virtual machine, called OpenVZ virtual machine, alongside with XEN in order to overcome the flaws in XEN virtual machine. Both types of virtual machine, XEN and OpenVZ, enhanced the performance of AnonyAI2’s computer network.

AnonyAC5 implemented many virtual machines of different types (KVM and XEN virtual machines) in order to create a backup environment for databases in his computer network. Both types of virtual machine, XEN and KVM, enhanced the backup process of AnonyAC5's computer network.

As explained in the previous example, the selection of requests without biases to a certain technology would encourage private actors to contribute their complements software to the collective. This is justified because their private contributions would be linked with other private contributions that operate in the same market. Private actors would benefit from this synergy rather than being scared of competition.

Other interesting characteristics for this period is that, as shown in table 7.1, the selection of requests always includes requests that would solve software bugs for already developed OpenNebula software.

Table 7.1 Summary of percentage of core and complements requests added into OpenNebula releases

Release	**Selected requests related to software bugs**	
	% Core	**% Complements**
1	52%	48%
2	62%	38%
3	65%	35%
4	65%	35%

This would encourage private actors to share their private contributions to the collective as their private contributions will be part of OpenNebula software which is continually developed and supported by the OpenNebula community over the time.

7.3.3 Development and Testing of Contributions

In the third transformation period, the selected requests are to be developed into lines of source code. This is done through planning these releases; each request is assigned a core member to be responsible for the development of the request into source code. For example, as seen in figure 7.5, Bug #11 is assigned to release 1.0 and 'Javi Fontan' is assigned to develop the request into the intended release.

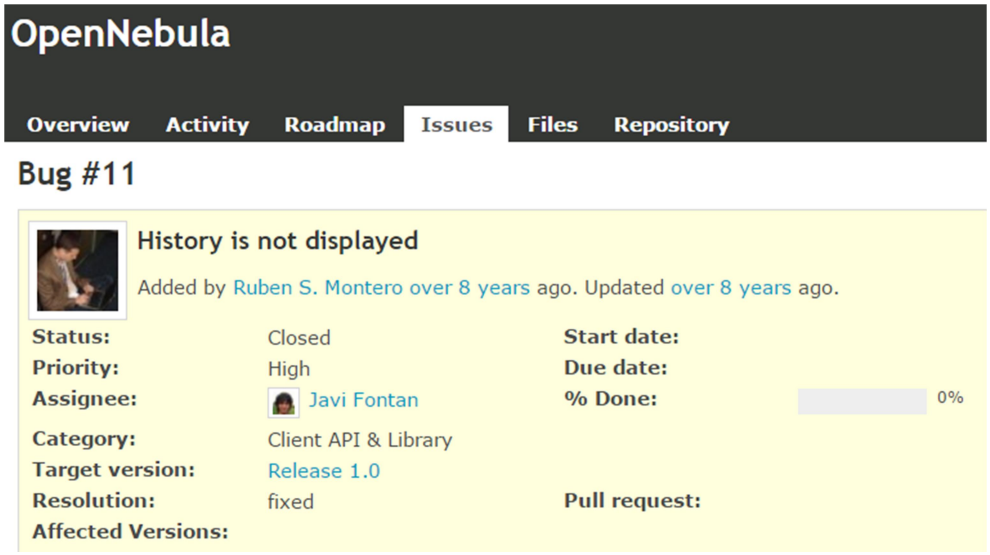

Figure 7.5 An example of the scope rule used in OpenNebula

The assigned core member is responsible for developing a source code that fulfils the request. However, other participants can also contribute by attaching a software patch into the request page or create a pull request that reveals their software patch (refer to 'solve software bugs' and 'develop software feature' contributions in section 6.4).

For example, bug #2503 was added by one of the core members (Daniel Molina) into the development portal of OpenNebula. This request reported a software bug in the software. This request was then added into the plan for OpenNebula release 4.6 and another core member (Carlos Martin) was responsible for fulfilling this request. A participant (a private actor with corporate email affiliation) participated in this request by adding a software patch that could solve the software bug. The software patch was tested by the core member and after some modifications, the patch was added into OpenNebula software; the software bug was resolved and the request was fulfilled.

The result from this transformation period is the actual development of core OpenNebula as well as the actual integration of complements with core OpenNebula. However, the software needs to be tested in order to make sure that the development of the software complies with the needs and desires of OpenNebula participants and users.

Therefore, a last transformation period starts where any further modifications need to be done to the software are conducted and the ready software is freely revealed to the collective. As shown in figure 7.6, the result was the continual development of OpenNebula software (core and complements components) over time[9].

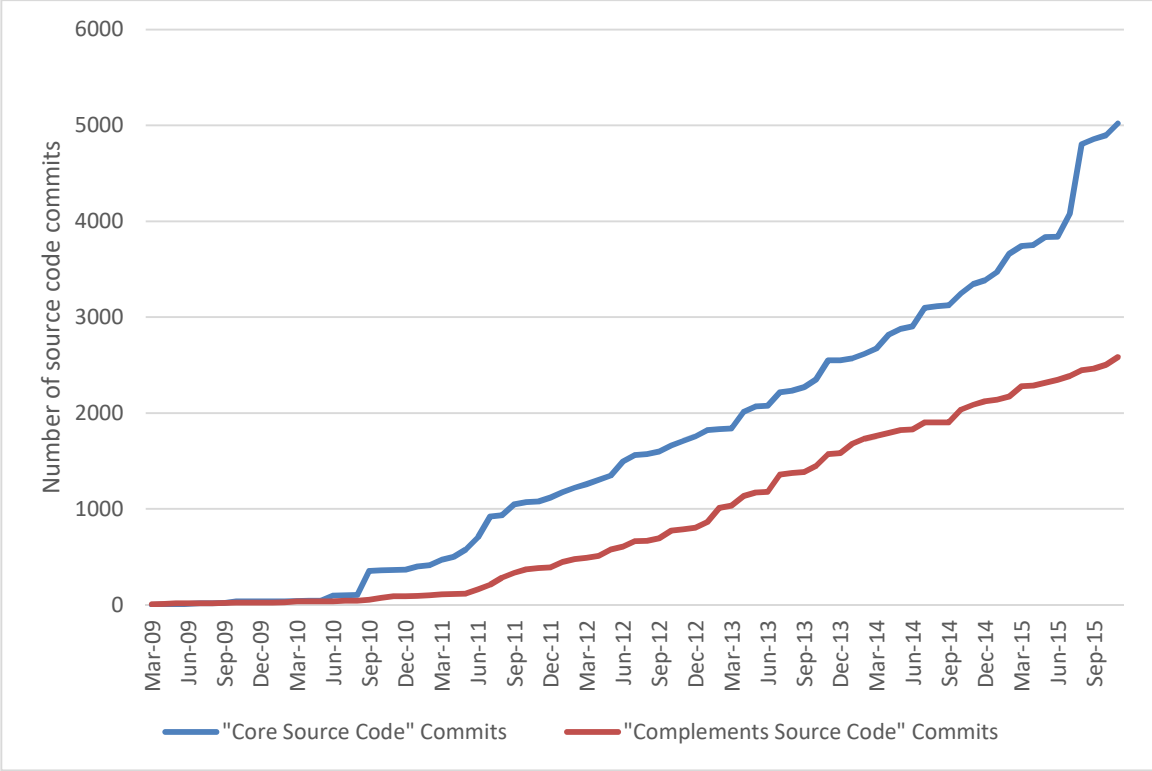

Figure 7.6 Commits added to OpenNebula core and complements software

In addition, different types of complements components were integrated through time (as shown in figure 7.7). Technical details about these complements are discussed in Appendix B.

[9] Development of core OpenNebula is measured by the number of commits added into source code of core OpenNebula through time.

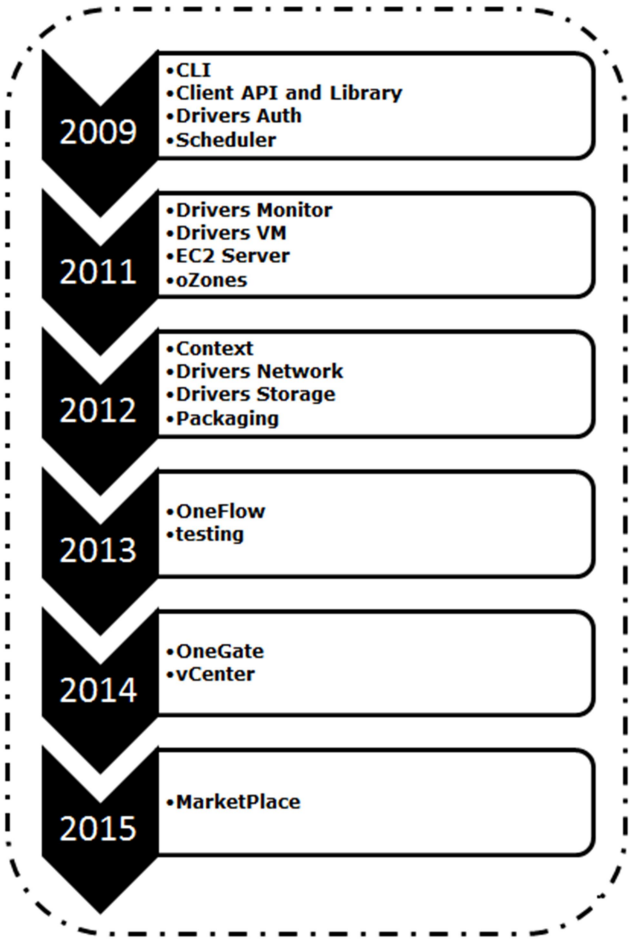

Figure 7.7 The addition of different complements through time in OpenNebula

This transformation period includes developing as well as testing the software. This period represents the actual sharing of private contributions by private actors; in this period, private actors are 'solving software bugs' and 'developing software features' (refer to section 6.4 for details about 'solve software bugs' and 'develop software feature' contributions).

It is proposed that this period is encouraging private actors to share their private contributions because private actors would be making their software (with all its technical design and specifications) within OpenNebula software. OpenNebula software and the private contributions are not

considered as two different pieces of software that are connected with each other; they are representing a single software that serves different needs. As a result, private actors would guarantee free sustainable development, support and enhancements for their software by the OpenNebula community.

AnonyAI1 shared his private contributions for integrating OpenVZ virtual machine within OpenNebula in 2010. Up to the end of November 2015, software bugs found in this virtual machine were reported and solved by participants in OpenNebula. In addition, different enhancements were suggested and developed for this virtual machine such as contextualization feature, LDAP authentication feature, DHCP IP assigning feature…etc.

7.4 Impact of Transformation Process on Contributions

Chapter six explained that private actors entered in an active communications in OpenNebula with their own knowledge, ideas and experiences. After that, they contribute and share their private contributions that lead to the development of the collective software.

In this chapter, it has been discussed how these private contributions are actually transformed into the collective software through a transformation process. It is found that this transformation process is a formal process that changes these requests from one state to another.

Table 7.2 explained that this transformation process functions to align the private interests with the collective interest as will be explained shortly.

Private actors would be encouraged to contribute because:

1. The transformation process, through validating contributions, directed the attention of private actors towards their contributions that are accepted and necessary by users of the software. Therefore, private actors would be encouraged to contribute by focusing on the most needed contributions, because they know that their suggestions, if needed (such as reporting software bugs and adding useful features) are likely to be validated by core members.
2. The transformation process, through selecting contributions, allows for possible synergies to emerge between the different contributions. Therefore, private actors would be encouraged to contribute because they may receive more than they give, due to potential synergies between the different contributions. If they do not contribute, they do not have a say in the development process and leave the rivals orient the software and benefit from synergies.

Table 7.2 Impact of the transformation process

Contribution	Transformation process	Effect of transformation
Report Software Bugs	Validation of contributions	Direct attention towards necessary contributions based on users' needs
	Selection of contributions	Allow synergy rather than substitution between different contributions
		Ensure continual development, maintenance and enhancements for contributions
Solve Software Bugs	Development and testing of contributions	Integrate contributions as part of OpenNebula software
Develop Software Features		

3. The transformation process, through selecting contributions, ensures the continual development, support and enhancements for contributions provided by private actors. Therefore, private actors would be encouraged to contribute (by solving software bugs and developing software features) in order to integrate their contributions and ensure the continual development of them.

4. The transformation process, through developing and testing contributions, properly integrates contributions within OpenNebula software. Therefore, private actors would be encouraged to contribute so the software would evolve in a direction that is coherent with their private needs. If they do not contribute, the software may not integrate what they need and become less relevant to their business.

As a conclusion, and informed by Ostrom (1990) evolutionary theory of collective action, it is proposed that a transformation process is implemented by core members in OpenNebula in order to align the collective interest with the private interest. Hence, aligning the collective interest with the private interest through 'transformation process' can be the second explanation provided in this thesis for how participants in open source software would privately invest in the software without creating the business dilemma.

Chapter 8 Rules in OpenNebula

8.1 Overview

This chapter explains rules that structure patterns in contributions of private actors in OpenNebula. It theorizes the different ways for executing the different rules that exist between participants in OpenNebula. In addition, this chapter proposes that rules are encouraging private actors to contribute their private contribution because rules are supporting private actors in inducing, verifying, legitimizing and adjusting their private contributions in open source software. Accordingly, private actors, through these rules, seem to work as a collective rather than worrying about free riders.

The presentation of results in this chapter is provided in three sections. Section 8.2 describes rules that exist in OpenNebula. Section 8.3 explains the different practices used in order to execute these rules. Section 8.4 theorises the impact of these rules in encouraging the private contributions in OpenNebula.

8.2 The Rules

Chapters six and seven explained the pattern in contributions of private actors in OpenNebula. Private actors were entering an active communications with other participants in which they communicate their knowledge, ideas and experiences. As a result, they decided to share their

contributions through a transformation process (validation, selection and development and testing of requests).

In this section, it will be discussed that there are a set of rules that seemed to structure the pattern in contributions of the private actors (as suggested in figure 8.1).

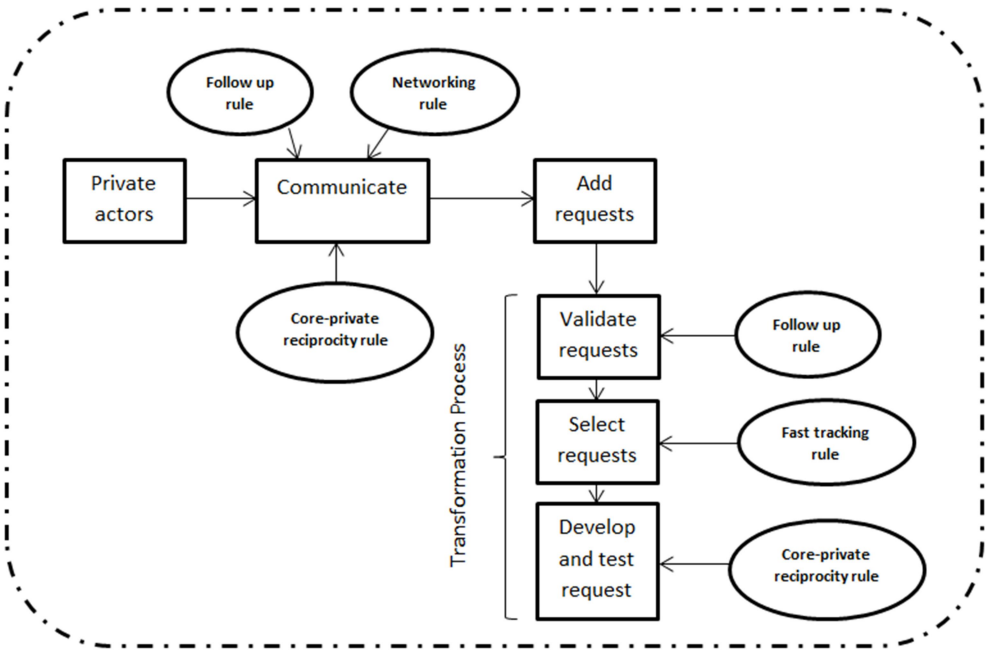

Figure 8.1 A set of rules that structure the patterns in contributions of private actors in OpenNebula

In order to explain and understand these rules, it is important to initially define rules in general then define rules in the context of OpenNebula.

Generally speaking, a rule is an instruction that explains actions that are considered permitted or forbidden to be done by individuals (Cole, 2014). Table 8.1 identifies and defines the set of rules that seemed to exist in OpenNebula.

Table 8.1 Rules in OpenNebula

Rules	Definition
"Fast tracking"	Selection of requests from the development portal into OpenNebula software releases by any participant (other than core members) is encouraged and accepted only if they provide money through 'fund a feature' programme or provide time through 'champions' programme.
"Follow up"	Validation of available documentation (shared knowledge in the mailing list and the development portal) by all participants is encouraged and accepted.
"Networking"	Supporting and opposing (implicit voting) any ideas and suggestions that are shared on the mailing list are encouraged and accepted by all participants.
"Core-private reciprocity"	Exchanging and sharing of developed and tested source code through pull[10] requests or production environments are encouraged and accepted by all participants.

[10] Pull request is providing the changes that are done to the software in order to be reviewed by interested parties.

8.2.1 "Fast tracking" rule

The first rule is the "fast tracking" rule. This rule identifies the conditions under which private actors would be able to select requests, which are added in the development portal, in order to be incorporated in the next OpenNebula software release.

The first condition is to financially sponsor the request(s): If a private actor wants to select requests in order to be incorporated within the software releases of OpenNebula, the private actors must financially sponsor that request.

Seven firms were found to financially sponsor the development of different requests that they choose to be incorporated within OpenNebula releases (see table 8.2 for examples).

Table 8.2 Firms who participated in the 'Fund a feature' programme

Firms	Sponsorship feature (from OpenNebula newsletters)
Unity	*"Virtual Routers functionality was funded by Unity in the context of the Fund a Feature Program"*
BlackBerry	*"Host offline mode, Marketplace, cluster resource sharing and Ceph as system datastore functionalities were funded by BlackBerry in the context of the Fund a Feature Program"*
	"the VM groups functionality is funded by Blackberry"
BIT.nl	*"Qcow2 snapshots implementation was funded by BIT.nl"*
SURFsara	*"GPU devices support was funded by SURFsara."*
Université Catholique de Louvain.	*"Flexible network attributes definition in contextualization was funded by Université Catholique de Louvain."*

The second condition is to voluntarily represent OpenNebula in technical events: If a private actor wants to select requests in order to be incorporated within the software releases of OpenNebula, the private actor must spend some time in representing OpenNebula in identified conferences and technology days as well as acting as a liaison between open source projects and the OpenNebula community.

Twenty-nine private actors were found to volunteer in representing OpenNebula in formal events. Those private actors are referred to as 'champions' in OpenNebula.

"Champions are passionate technology and community leaders that represent OpenNebula, help sustain and grow its user base, and act as a liaison between other open-source projects and its community." (OpenNebula Official Website)

This rule is labelled as "fast tracking" rule because when private actors do certain things (paying or volunteering for the benefits of OpenNebula), their requests are "fast tracked" and incorporated into the next release of OpenNebula software.

8.2.2 "Follow up" rule

The second rule is the "follow up" rule. This rule states that all documentation in OpenNebula shall be updated through excluding invalid and/or proprietary documentation.

Documentation in OpenNebula that is studied in this thesis includes: (a) emails sent on the mailing list and (b) requests added into the development portal. Both of them contain a huge amount of information that is disseminated

between the different participants. All this documentation shall be updated by core members on regular basis. Updates on this documentation are mainly aiming to check the validity of this documentation and exclude proprietary emails and requests.

For the first type of documentation in OpenNebula, it is found that most of the mails sent on the mailing list are replied to by core members.

"However, to enhance the activity of our Forum we have an active support team trying to solve your questions at any time. Feel free to ask and answer any question" (OpenNebula Newsletter)

Through these replies, core members seemed to filter these emails into valid and invalid emails.

Valid emails are suitable emails and further discussions can be attached to them.

A core member replied to an email sent by AnonoyAC1 about a technical problem he experienced while implementing OpenNebula requesting more details about the problem. The result from their discussions was solving the problem and sharing the solution with the community.

A core member replied to an email sent by AnonyAI1 about his suggestion regarding the integration of new virtual machine into OpenNebula requesting him to help them with the development. The result from their discussions was the

development and integration of ESXi virtual machine with OpenNebula.

A core member replied to an email sent by AnonyAE3 about the difference between different commands used in the different releases of OpenNebula. The result was a better understanding of the different commands and their implementations in OpenNebula.

However, invalid emails are emails that contain information that is outside the defined scope for the mailing list or emails that suggest proprietary solutions to technical problems of OpenNebula. Core members explained that the email cannot be processed further.

For example, it is stated in the official website of OpenNebula that the mailing list has a defined scope; emails sent on the mailing list shall be related to: questions about installing and implementing OpenNebula software, discussing new developments, asking questions about technical problems that are experienced by participants…etc. Therefore, any emails that contain information outside this defined scope are considered invalid.

A core member replied to AnonyAG3's emails sent about a technical problem that he experienced while implementing OpenNebula, saying that solving the problem should be done through live chat rather than on the mailing list because of the SLA agreement that they both signed.

A core member replied to an email sent by one of the OpenNebula participants that the email contains marketing details that are not allowed to be disseminated through the mailing list

In addition, through these replies, core members seemed to stop further discussions related to proprietary software.

A core member replied to an email sent by one of the OpenNebula participants, saying that the email provided a proprietary solution that is not allowed to be suggested in a vendor-agnostic mailing list.

For the second type of documentation in OpenNebula, among all requests added into the development portal, 75% have been followed up by core members. For example, core members define: 9% of the requests as invalid requests as they are outside the scope of OpenNebula software, 4% of the requests as duplicated requests as they have already been added by others, 62% of the requests as valid requests that need to be developed within OpenNebula and 25% of the requests are left without any follow up.

This rule is labelled as “follow up” rule because when private actors contribute by sending an email or adding a request, core members “followed up” these contributions by conducting further examinations on these contributions in order to check validity.

8.2.3 "Networking" rule

The third rule is the "networking" rule. This rule defines voting as the social activity that can occur between the different participants in OpenNebula.

Through their discussions on the mailing list, private actors and core members were found to send emails in which they provide suggestions and ideas for developing OpenNebula software. Other private actors were found to reply to those emails by either showing their support for the suggestions or showing their opposition for the suggestions (see table 8.3 for examples).

Table 8.3 Examples of "networking" rule in OpenNebula

Participant	Illustrative quote
AnonyAI2	*"I think this is a great idea for lease pools, countless times I went to the network section just to get a vm id"*
AnonyAI2	*"I agree, modifying the network range would be quite useful."*
AnonyAE3	*"I am bias, here is my "apt" replacement for a market place that will work with open nebular and stratuslab, as I don't believe in lock in."*
AnonyAE5	*"I agree with you that tap:aio should be used instead of file:, but sometimes it is not possible :-("*
AnonyAI1	*"This setup has a fundamental limitation in the sense that we cannot have mixed CPU numbers and assure fair power distribution according to their CPU value."*

Supporting and opposing requests enhance interactions and collaboration between participants.

Through supporting and opposing requests, participants were found to interact by initiating discussions and providing their own insights and different perspective about requests.

For example, while opposing a request suggested by a participant, AnonyAE5 initiated a discussion and provided his own insights about possible problems that may occur because of this suggestion.

While supporting a request suggested by a participant, AnonyAI2 initiated a discussion and provided his own insights about enhancing the computer network in terms of performance and scalability.

As a result, participants are better collaborated together. For example,

While supporting a request suggested by a participant, AnonyAI4 initiated a discussion and provided his own understanding about pros and cons of different performance metrics for OpenNebula. Accordingly, further collaboration was conducted between AnonyAI4 and another participant to develop performance metrics in 'LXC' drivers.

This rule is labelled as "networking" rule because it allows private actors to support or oppose requests, interact a lot with one another and collaborate accordingly.

8.2.4 "Core-private reciprocity" rule

The fourth rule is the "core-private reciprocity" rule. This rule identifies the ways of exchanging and sharing of the source code between private actors and core members of OpenNebula.

The first way of exchanging and sharing of the source code is when private actors develop and conduct changes to

OpenNebula software and reveal their development efforts with core members through 'pull' requests. For example, 691 pull requests were added to OpenNebula software; 537 pull requests were rejected, 117 pull requests were approved and reflected into the source code and 37 pull requests are still waiting for checking.

The second way of exchanging and sharing of the source code is when core members provide source code (either to solve a software bug or to develop a software feature) that needs to be developed and tested and the private actors would do the development and testing in their production environments (see table 8.4).

Table 8.4 Examples of the "core-private reciprocity" rule in OpenNebula

Participant	Illustrative quotes
AnonyAC4	*"Really sorry for late reply, Sounds excellent to merge cloud init with ONE. I will try the metadata server as Ricardo pointed out."*
AnonyAI1	*"Im testing your example and it simply works. Perhaps you can give some > more input on the problem? As I say, it shoudln't be CDATA related if > working with REXML."*
AnonyAC3	*"I have just tried to do the following as another method of testing installing a Windows"*
AnonyAE5	*"After playing a little bit around with ElasticFox and the authentication I can see that at least the user pool is queried if I try do some action in ElasticFox - in the next days I will spent some more time on this issue, maybe I can upload an (AMI?) image to ONE."*

8.3 Execution Practices

The analysis of OpenNebula revealed that not all of these rules are enforced by 'OpenNebula systems' in order to be followed and executed by participants of OpenNebula.

Actually, the analysis revealed that there are some rules that are informally emerged and executed between the different participants. Those rules started as part of the communications in the mailing list, repeated them through time and then changed into an acceptable informal rule that participants agree to follow and execute in the development portal.

For example, participants communicated in the mailing list by sharing the software bugs that they had confronted while implementing OpenNebula software. Other participants offered a development help in order to solve those software bugs. They exchange and share source code in order to solve the software bugs.

As a result, the different participants understand (based on their experience through the mailing list) the benefits of exchanging and sharing of the source code in order to solve software bugs. Accordingly, exchanging and sharing of the source code are not formally required but socially encouraged and accepted through time by the different participants. Exchanging and sharing of the source code is an informal rule that is referred to as 'core-private reciprocity' rule.

Table 8.5 explains the two main execution practices for rules in OpenNebula.

The first practice is a formal execution of the rule. Through this practice, participants strictly follow regulated policy that would generate benefit for them. The second practice is an informal execution of the rule. Through this

practice, participants informally execute rules that would generate benefits for them.

Table 8.5 Execution practices of OpenNebula rules

Execution Practice of rule	Description
Formally executed rule	Rule is formally executed according to an official programme in OpenNebula.
Informally executed rule	Rule is informally executed; rule is socially emerged between participants.

In order to understand these practices, Table 8.6 summarizes rules and their execution practices in OpenNebula.

Table 8.6 Summarizing results for rules and their execution practices in OpenNebula

Rule	Description	Rule Execution practice
"Fast tracking"	Selection of requests through 'Fund a feature' programme	Formally executed rule.
	Selection of requests through 'Champion' programme	
"Follow up"	Excluding invalid documentation	Formally executed rule on the mailing list and informally executed rule in the development portal.
	Excluding proprietary documentation	
"Networking"	Voting about provided suggestions	Informally executed rule.
"Core-private reciprocity"	Sharing the developed source code by private actors through 'pull' requests.	Informally executed rule.
	Sharing the tested source code by private actors in their production environment.	

8.3.1 Execution of the "Fast Tracking" Rule

"Fast tracking" is a formally executed rule. Private actors can only select requests from the development portal only if they officially register in either the 'Fund a feature' programme or the 'champions' programme.

(a) 'Fund a feature' programme

"When we define the roadmap for a new OpenNebula release we listen to all users, trying to prioritize the features demanded by the organizations supporting the open-source project with a commercial subscription. However we cannot guarantee a time frame for their development. The Fund a Feature Program can be used to implement within a given time frame new functionality or enhancements in the code, new or enhanced drivers, or new integrations with existing management, billing and other OAM&P systems." (OpenNebula official website)

(b) 'Champions' programme

"Champions are passionate volunteers who work to connect, teach and spread OpenNebula, throughout the world. Some of the roles that a Champion can play are: …. Participate in local meetups, user groups, etc." (OpenNebula official website)

Both programmes are formally declared by OpenNebula systems in 2013 and 2015, respectively. Both programmes incite private actors to get involved with the development of

the software and to integrate their complements software with core OpenNebula.

Through the 'fund a feature' programme, private actors would accelerate the development of OpenNebula software according to their business needs.

"Funding a feature not only gets you the feature you need faster, but allows you to contribute to the open source project from which you derive so much value." (OpenNebula official website)

Through the 'champions' programme, private actors would integrate their business needs with OpenNebula software. And through the different events in which they volunteer, they would raise the awareness about OpenNebula software and its commercial implementation. Thus, they would be disseminating knowledge about OpenNebula software and their business which is integrated within OpenNebula,

"These events provide a great opportunity to raise awareness for the project and get more of you involved as contributors and users. As we scale the project to the next level, we need your help in spreading the message." (OpenNebula official website)

In sum, "fast tracking" rule is formally executed through either the 'fund a feature' programme or the 'champions' programme.

8.3.2 Execution of the "Follow Up" Rule

The "follow up" rule is a rule that is formally and informally executed in OpenNebula as follows. On the mailing list, invalid and proprietary emails are formally excluded from the mailing list. Invalid and duplicated requests are informally excluded from the development portal.

For the mailing list, it is stated in the official website of OpenNebula that:

"The average response time is within 1 business day." (OpenNebula Official Website)

It is found that this official rule is executed in OpenNebula. For example, as shown in table 8.7, emails sent by participants in OpenNebula are responded to within one business day as officially required.

Table 8.7 Examples of formal execution of the "fast tracking" rule

Sent date	Response date	Sent date	Response date
AnonyAI1: 14th June, 2009	Core member: 15th June 2009	AnonyAC1: 24th April 2014	Core member: 24th April 2014
AnonyAI1: 15th Sept 2009	Core member: 16th Sept 2009	AnonyAC3: 8th July 2013	Core member: 9th July 2013
AnonyAI2: 29th April 2012	Core member: 30th April 2012	AnonyAG2: 17th Jan 2013	Core member: 17th Jan 2013
AnonyAI2: 21st June 2013	Core members: 22nd June 2013	AnonyAE1: 3rd December 2013	Core member: 5th December 2013

In addition, excluding emails that contain proprietary solutions is formally required in the official website of OpenNebula.

"OpenNebula roadmap is completely driven by users' needs with features that meet real demands, and not features that result from an agreement among the different vendors participating in the management board of the project." (OpenNebula Official Website)

However, there is no evidence to show that requests in the development portal are updated according to a formal regulation for updating requests. Thus, it is inferred that requests are updated with no formal regulation that strictly controls updating requests. For example:

Some requests were updated in the same month of adding the request such as request number '1441' was added and updated on September 2012 and request number '1561' was added and updated on October 2012.

Some requests were updated within months of adding the request such as request number '218' was added on April 2010 while updated on July 2010 and request number '183' was added on December 2009 while updated on April 2010.

Some requests were updated within a year or more of adding the request such as request number '115' was added on June 2009 while updated on July 2010 and request number

'1610' was added on October 2012 while updated on March 2015.

In sum, "follow up" rule is formally executed for emails sent on the mailing list but informally executed for requests added into the development portal.

8.3.3 Execution of the "Networking" Rule

For the "networking" rule, voting and collaboration emerged between core members and private actors without being formally enforced by 'OpenNebula systems'.

Private actors are given the opportunity, through the mailing list, to voluntary discuss and implicitly vote with or against some suggestions that may be added into OpenNebula software releases.

"We would love to hear your feedback, so we have time to include possible changes in the next maintenance release. You can reach us through the user mailing list, give it a spin!" (OpenNebula Newsletter)

"Feedback from the community has started trickling" (OpenNebula Newsletter)

"Also, this kind of feedback from the OpenNebula users makes us blush and happy, and willing to keep OpenNebula in the right track!" (OpenNebula Newsletter)

"This feedback was crucial in releasing OpenNebula 4.6.1, which fix bugs present in the first Carina version." (OpenNebula Newsletter)

"specially this last month thanks to the vulnerability discovered by folks at Horst Görtz Institute for IT-Security, Ruhr-University Bochum. This feedback was crucial in releasing OpenNebula 4.6.2." (OpenNebula Newsletter)

All previous examples show that "networking rule" is informally executed between participants of OpenNebula.

8.3.4 Execution of the "Core-Private Reciprocity" Rule

For "core-private reciprocity" rule, there is no formal regulation that enforces exchanging and sharing of the source code between core members and private actors. However, many incidents have been highlighted in which they seem to develop and share their private development in the development portal and the mailing list.

"Microsoft announced in the OSCON 2014 their willingness to tightly collaborate with OpenNebula in order to build Microsoft Azure support within your favourite Cloud Management Platform." (OpenNebula Newsletter)

"Also last month, guys from OneInsight presented their visualization plugin for OpenNebula in the CentOS Dojo in Lyon." (OpenNebula Newsletter)

"We want to give a big thanks from here to Carlo Daffara and Vincent V.d. Kussen for their intensive testing, pushing OpenNebula to its limits." (OpenNebula Newsletter)

"We want to highlight the excellent contribution made by Terradue, in the form of an OpenNebula add-on." (OpenNebula Newsletter)

All previous examples show that "core-private reciprocity" rule is informally executed between participants of OpenNebula.

8.4 Impact of Rules on Contributions

The analysis so far revealed that the community of participants in OpenNebula, both core members and private actors, collectively found their own ways to emerge their rules and to agree on their executions (refer to sections 8.2 and 8.3 for details).

Informed by Ostrom (1990) evolutionary theory of collective action, it is proposed that rules, which structure the patterns in contributions for private actors, emerged based on the local experience of OpenNebula participants.

Through these rules and their executions,

(1) The private interest is achieved because rules encouraged the private investments through "fast tracking", "follow up" and "networking" rules

(2) The collective interest is achieved because rules encouraged the revealing of the private investments through "core-private reciprocity" rule.

As a result, participants would induce, verify and legitimate their private investments while ensuring the co-creation of the collective software (as shown in figure 8.2).

To start with, the "fast tracking" rule stipulates private actors to invest their money or time in order to select requests that would be incorporated with the next OpenNebula software release. Private actors would be encouraged to contribute because, through "fast tracking" rule, private actors would be able to induce their business needs within OpenNebula software. Inducing their business needs within OpenNebula software would indeed support private actors in attracting the attention of OpenNebula users and participants towards their business.

One of the salient examples is BlackBerry Company. Through the "fast tracking" rule, especially 'Fund a feature' programme, BlackBerry was able to induce its contributions such as 'VM groups', 'VM operation permissions (ADMIN,MANAGE and USE)', 'VM history', 'token functionality' and 'LDAP group mapping'. In addition, BlackBerry was able to induce its technologies as a representor from the company was a keynote speaker in OpenNebula conference in 2017.

The "follow up" rule requires core members to exclude invalid emails and to request as well as to reject proprietary

integration. With the huge amount of documentation disseminated on the mailing list and the development portal, the “follow up” rule filters this documentation by excluding invalid and duplicated ones. Private actors would be encouraged to contribute because contributions will be verified. Thus, attention of OpenNebula users and participants would not be distracted with irrelevant and proprietary documentation.

For example, AnonyAI1 suggested contributions related to ‘the integration of Contextualization with Windows VM’, ‘Monitoring driver’, ‘Hyper- V Driver’, ‘Redhat Cgroup’, ‘Redhat Enterprise Virtualization (RHEV)’. Three of these suggested contributions were excluded. Therefore, the attention of OpenNebula participants was directed towards the two remaining verified suggestions. Without AnonyAI1 contributions, the two contributions would not be discussed, verified and accepted.

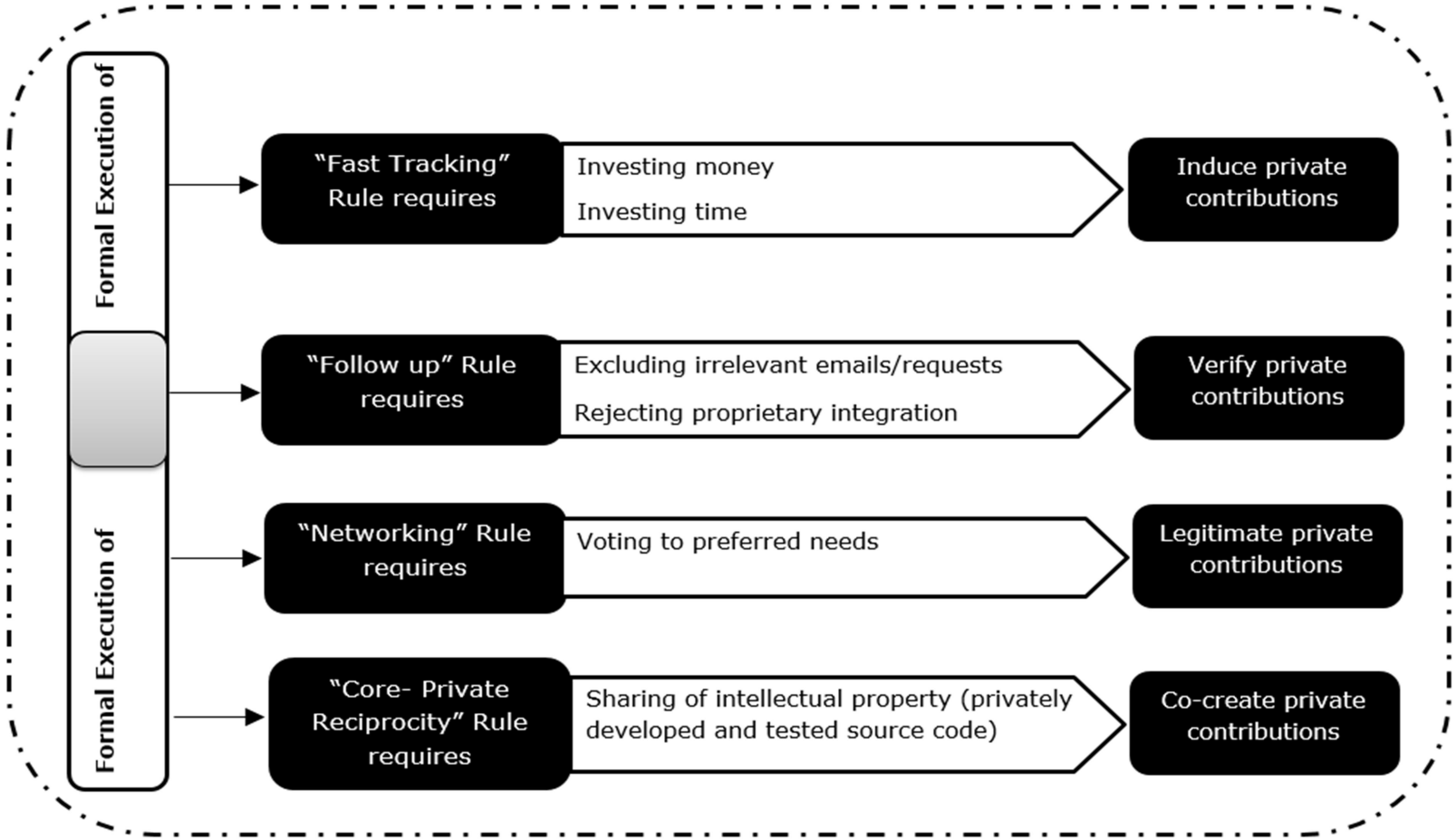

Figure 8.2 Impact of rules in OpenNebula

Valid contributions that are suggested and revealed by private actors represent their needs from OpenNebula software. Those needs are plenty and different as private actors are heterogeneous (as explained in chapter 6). Therefore, the “networking” rule requires private actors to provide feedback and vote for their preferences in order to legitimate the most favoured ones.

Referring to the previous example about contributions of AnonyAI1, core members and OpenNebula participants provided feedback about the two verified contributions from AnonyAI1. Based on their discussions, one of these verified suggestions was accepted as a legitimate contribution that shall be selected and immediately incorporated in OpenNebula software through requests numbers 563, 568, 636 and 701.

Finally, the “core-private reciprocity” rule means exchanging and sharing of source code between core members and private actors. The “core-private reciprocity” rule encouraged private actors to share their private knowledge (the privately developed and tested source code) in order to integrate their contributions with core OpenNebula and co-create OpenNebula software.

OpenNebula software is currently single software that is jointly produced by core members and private actors in which both developed a software that is mutually beneficial. Core OpenNebula and complements are integrated as one software as shown in figure 8.3. Without development and testing

contributions, OpenNebula software would be only core OpenNebula software that is separated from external complements software.

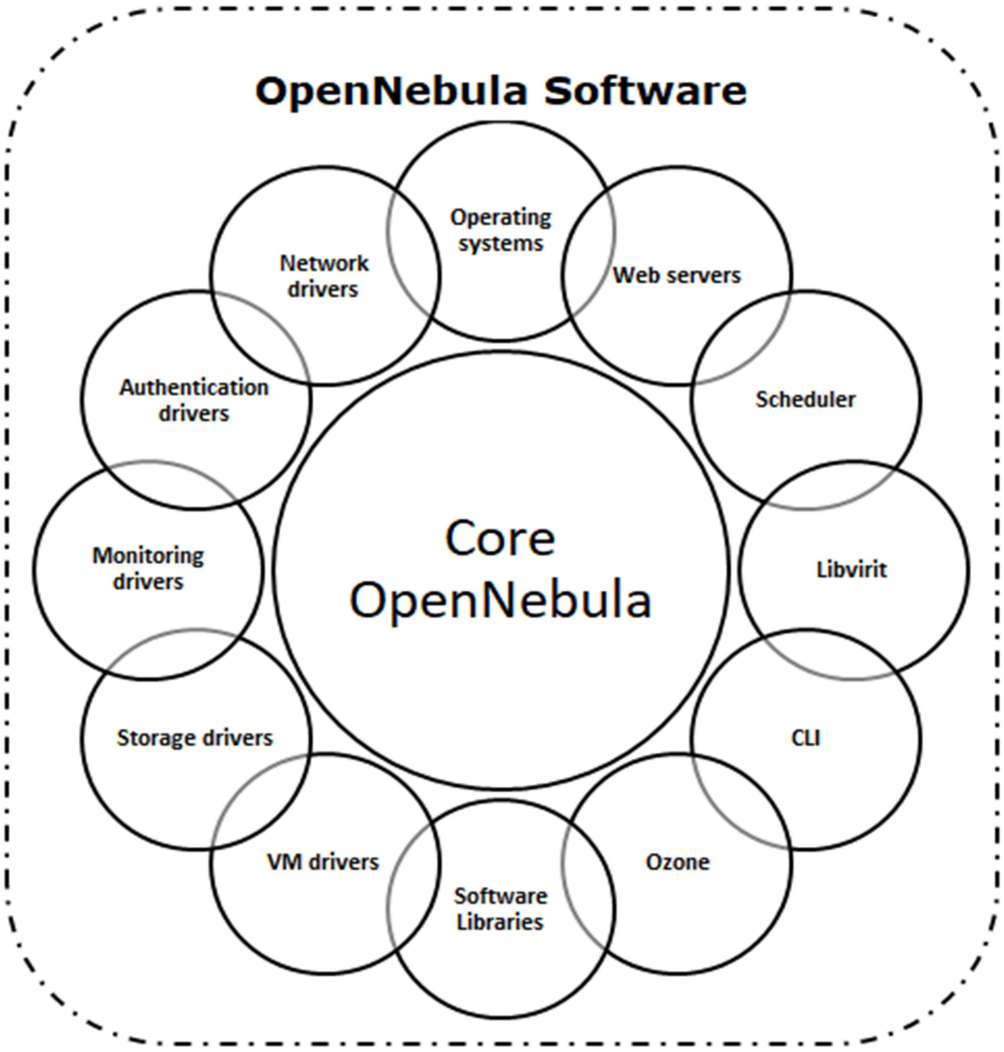

Figure 8.3 The technical components of the jointly developed OpenNebula software

Informed by Ostrom (1990) evolutionary theory of collective action, it is proposed that rules and their execution practices are supporting the private actors to work as a collective in OpenNebula rather than scaring from rivalry and the business dilemma that would be experienced. They work as a collective to induce, verify, legitimate and co-create their

private contributions. They collectively work to support their private interest as well as to ensure the continual development of the collective software.

Hence, aligning the collective interest with the private interest through ‘rules and their executions’ can be the third explanation provided in this thesis for how participants in open source software would privately invest in the software without creating the business dilemma.

Chapter 9 Discussion

9.1 Research Review

Open source software is an innovation in which the development and the usage of the software is delivered for, and developed by, users (von Hippel, 2001). Users are individual participants and private actors who are geographically separated but, using internet, which are collaborating and developing open source software that is declared under either restrictive or permissive license type.

Open source software is the prominent example for the 'private-collective' model of innovation suggested by von Hippel and von Krogh (2003). This model explains that private actors are crucial investors for open source software. This model also emphasizes that revealing the private investments of the software by private actors can be the best course of action that will both increase the private benefit for private actors and will support the development of the collective software.

However, this thesis suggested that investing in open source software, under permissive licensing, and revealing these investments may weaken the 'appropriability regime' for private actors. This is justified because if private actors revealed the software, then the source code would be available to anyone because the software is considered a 'commons'. This would mean that their 'appropriability regime' ; knowledge in terms of know-how would be available

and, accordingly, imitating of their innovation would be uncomplicated (Teece, 1986). Any participant can then copy the source code and start their own software project and of course become a competitor.

Therefore, it is suggested in this thesis that private actors would experience a business dilemma as follows. Private actors need to share their private investments in the open source software in order to benefit from it. However, their contributions can be easily hijacked by rivals. Rivalry would be deleterious because their innovation would be easily imitated and profits will be shared between innovators and imitators.

Thus, this dilemma would theoretically discourage private actors to share their contributions to open source software without a strong guarantee. However, in some cases of reality, this restrain is not observed. For example, statistics show that private actors tend to increasingly contribute and innovate to permissive open source software in different industrial sectors over years (Skok, 2013).

In sum, before conducting this research, it was understood that there is a business dilemma experienced by private actors that theoretically is suggested to discourage private actors from contributing to open source software. But practically, private actors are found to contribute their private knowledge to open source software despite the dilemma. This difference between theory and practice encourages the start of this thesis in order to answer the following questions:

- ‘Why private actors choose to invest and share rather than to free ride in permissive open source software?’
- ‘How can the private actors invest and share in permissive open source software without experiencing a business dilemma?’

9.2 Empirical Findings

The data collection and analysis in this thesis were informed by Ostrom (1990) evolutionary theory of collective action. This theory explains that people are rationale; they can talk to each other and to use their local knowledge and experiences in order to arrange the pattern of their collective action. This is crucial in order to extract themselves from collective action problems that they experienced.

Findings revealed the pattern in contributions of private actors of OpenNebula. Private actors are involved in an active communication, they add requests accordingly then a transformation process transformed these requests into OpenNebula software in the form of ‘collective complementarities’. In addition, a set of rules emerged by private actors in order to structure this pattern.

These findings help in answering the research questions as follows:

First, private actors are voluntary entering in an ‘active communications’ with other participants.

Through this ‘active communications’, they present themselves and their preferences, highlight software bugs and

features, suggest ideas and experiences, solve software bugs, share source code, support and oppose alternatives…etc.

Evidences proposed that an 'active communications' act as a prerequisite for the active private contributions done by private actors.

Second, the private contributions are locked within the collective software in the form of 'collective complementarities' through a 'transformation process'.

Several evidences proposed that this process worked to align the private interests of private actors with the collective interests of the software. The process achieved that private interest because it directed the attention of private actors towards necessary contributions based on users' needs, allowed for synergy between the different contributions, ensured the continual development of contributions. In addition, the process aligned the private interests with the collective interest because it integrated the private contributions as part of the collective software (collective complementarities).

Third, findings revealed that a set of 'rules' are emerged according to the exchanged knowledge and experiences of the private actors and other participants.

Several evidences proposed that these 'rules' worked to support the alignment between the private and the collective interests, which is done through the 'active communications' and the 'transformation processes. Rules achieved the private

interests because it supported the verification, legitimating and induction of private contributions. In addition, rules aligned the private interests with the collective interest because they co-created the collective software through the private contributions.

Fourth, it is proposed that the alignment between the private and the collective interests encourages private actors to share and to link their private software with the collective.

As explained earlier the alignment between the private and the collective interests is done through the 'active communications' and 'transformation process'. The alignment is also supported through a set of 'rules' that emerged based on the experiences and knowledge of private actors.

This alignment worked to lock the private software with the collective software. Through sharing of the private contributions, the private software would be part of the 'collective' complementarities' that are used across the different industries.

9.3 Research Contribution

9.3.1 Theoretical Contributions

Findings of this thesis contribute to the literature of open source software by thriving von Hippel and von Krogh (2003) 'private-collective' model of innovation (performed under permissive licensing) in different ways.

The first broad contribution of this thesis is through the introduction of 'collective complementarities' concept.

In the ‘private-collective’ innovation model, private investments are considered crucial investments for open source software. In addition, revealing the private investments is better than free riding others investments. Thus, private actors are innovators who reveal their innovation to the collective. According to Teece (1986), a competitive advantage for an innovator could be gained by selling proprietary complementarities for example.

Findings in this thesis reveal that, through ‘collective complementarities’, private actors better lock their private software with the collective software. They are bridging their private benefits with the collective benefits. Through sharing their private investments, private actors work alongside with other community members to collectively do something beneficial for them all.

In addition, the development of the software (in the form of ‘collective complementarities’) is positively affecting the competitive advantage of the private actors. this is justified because such software is considered ‘generative’; software that can be implemented according to heterogeneous needs (Boland Jr et al., 2007). And generativity is considered as competitive advantage for developers of the software as it help developers to broaden the implementations of their software across different markets (Yoo et al., 2010).

The second broad contribution of this thesis is through a better understanding of the characteristics of the collective

software (open source software declared under permissive licensing).

Open source software is a ‘commons’ because it is a non-excludable resource (Hess and Ostrom, 2005); the source code is publicly available and discrimination against the use of the source code is not allowed.

Appropriation of the ‘commons’ is usually associated with a collective action problem that is considered harmful to the commons (Ostrom, 2003). For example, O'Mahony (2003) explained that open source software declared under permissive licensing can be appropriated by participants who can privatize the software. This leads to Hardin (1977) ‘tragedy of the commons’ problem that reduces the availability of the software to the collective.

In order to overcome collective action problems that are caused by appropriation of the commons, Ostrom (1990) suggested that contributors to the ‘commons’ can gather and agree on arrangements, without formal intervention, to: (a) minimize rivalry and (b) control the capacity and frequency of appropriation to the ‘commons’.

As shown in table 9.1, this thesis enhance our understanding of how the ‘private-collective’ model of innovation worked in open source software under permissive licensing by explaining open source software as a ‘commons’ that is exposed to appropriation. Details of these differences are explained shortly.

Table 9.1 Differences between open source software and other appropriated 'commons'

	'Commons'	**Open source software as a 'Commons'**
Definition	Excludable resource	Excludable resource
Nature of the 'commons'	Finished 'commons'	Unfinished 'commons'
Appropriation effect	Appropriation is harmful. It causes the depletion of the 'commons'	Appropriation can be beneficial. It enhances the development of the software
Rivalry effect	Rivalry is not encouraged. It accelerates depletion of the 'commons'. Therefore, Contributors are owners who shall exclude others and control use of the 'commons'	Rivalry is encouraged. It enhances the development of the software according to different needs. Contributors do not exclude anyone. Contributors are forfeiting their ownership rights of the software but controlling the future direction for the software

First,contrary to other ‘commons’, appropriation in open source software is encouraged and rivalry is found beneficial to open source software.

This can be explained by the differences of nature for these ‘commons’. Fishery for example is a finished ‘commons’; it is ready to be used immediately after appropriation. Unlike fishery, open source software is an unfinished good. Copying the source code is not enough for anyone to start using it. One need to modify it, amend it, and configure it in order to fit their requirements and computing setting. Accordingly, appropriation of the software is enhancing its value rather than diminishing it; appropriation caused the continual development of the software. On the other hand, rivals are crucial appropriators because their appropriation of the software would properly reflect market needs and requirements.

Second, contrary to other ‘commons’, contributors in open source software are forfeiting their rights by revealing the software to the public but are still forcing constrains on the technical direction of the software for their business benefits.

Usually, contributors to the ‘commons’ are owners of it. They designed their own arrangements in order to keep their own right for using the ‘commons’ and exclude others. But in open source software context, contributors are revealing their private investments to the collective; their privately developed software is no more considered part of their intellectual

property. Anyone can copy it, modify it and combine it with other pieces of software. However, contributors are also forming their agreements, such as active communications, transformation process, rules and their formal and informal execution, in order to control the future direction of the software.

For example, through the formal and informal rules, private actors are inducing their own technologies in the open source market, verify their technologies by open source needs, legitimize their technologies as part of the open source market and co-create their technologies in order to broaden the implementations of their technologies.

This thesis also contributes to the literature of open source software as 'sponsored' software.

Researchers categorises open source software into autonomous and sponsored software (West and O'Mahony, 2005a; O'Mahony, 2007; West and O'Mahony, 2008). Autonomous open source software refers to software initiated by individuals and are self-managed (de Laat, 2007; West and O'Mahony, 2008), while sponsored open source software are those under the authority and control of a profit or non-profit organisation (West and O'Mahony, 2008). As shown in table 9.2, each category has its pros and cons.

Table 9.2 Examples of the advantages and disadvantages of autonomous and sponsored open source software

	Autonomous	Sponsored
Pros	High norms of reciprocity and sharing between participants	The development of the software within commercial attention
	The community is evolving with the software	Ongoing technical and strategic support
Cons	Lack of commercial support and attention	Worry about control over the software
	Limited technical scope for the software	Uncertainty about the future legal arrangements for the software

Advantages revealed for OpenNebula as sponsored software are: incubating the development of the software within commercial and market attention, providing ongoing technical resources to ensure the sustainable development of the software and controlling the software commercial direction and participants heterogeneous priorities.

However, arrangements identified in chapters seven and eight help in bringing the best of the two worlds; the autonomous and the sponsored.

In sponsored software, software is already developed and disclosed by a firm. Thus, participants are directly introduced with complex software and may find it hard to understand the software. However, these rules allow all

participants to grow with the software and the community and learn the software accordingly.

In addition, in sponsored software, participants are worried about control over the future of the software as the technical direction of the software would be serving firms' need and is not be clear for participants. However, these rules allow participants and sponsors to share ongoing control over the development of the software as well as determining the future direction of the software.

Moreover, norms of sharing and reciprocity are usually associated with participants in autonomous software as they all work for the collective. However, rules support the norms of sharing and reciprocity between sponsors and private actors in order to generate collective software.

9.3.2 Practical Contribution

First, this thesis suggests that private actors would be experiencing a collective problem, the business dilemma, which would discourage them from contributing. Therefore, the business dilemma represents a problem of motivation that requires managers and practitioners attention.

This thesis theorises an Information Technology (IT) practice (transformation process in chapter 7) as a solution for this problem. And such theorisation should help practitioners in developing better practices while investing in open source software.

Information Technology practices are examined in the literature of open source software (Yamauchi et al., 2000; Scacchi, 2002). Information Technology practices are usually used as a solution to operational problems such as coordination problems (Markus, 2007). However, this thesis benefits from this information technology practice in order to solve a motivation problem (collective action problem).

Practically, this thesis encourages practitioners to harness information technology practices in organizations to better align the private with the collective interests. Through information technology practices, practitioners can direct attention of private actors towards necessary contributions based on users' needs, allow for synergy between the different contributions, ensure the continual development and enhancement of contributions and integrate contributions as part of the collective software.

Second, this thesis theorises a bundle of rules that would encourage both innovators and rivals to play an active role in the development of the software.

This point is crucial because generally in the literature, researchers focused on the active role for innovators in open source software (West and O'Mahony, 2005a). For example, innovators participate to the development of the software, plan the future direction of the software and are able to re-license the software at any time. On the other hand, other participants are playing a passive role in the software (von Krogh et al., 2012). They can only participate to the

development of the software. They don't have any control over the software; if participants' motivations are similar to the innovator, they would sustain their participations. Otherwise, they will stop their participations to the software.

As shown in table 9.3, findings in this thesis suggested the development of informal rules alongside the formal ones and these rules allow participants to play their active role that benefit them and the development of the software as well.

They are part of the software now. They are not only participating to the software, they can plan the development of the software by inducing their own technologies and co-create their software with the collective software.

Therefore, practically, findings would encourage practitioners (who are innovators of the software) to enforce their formal rules while allowing other participants to informally agree on their rules in order to enhance the development of the collective software and develop better relationships with rivals and partners.

Table 9.3 The active role of participants with arrangements (formal and informal rules)

	Without arrangements		**With arrangements**	
	Innovators	**Participants**	**Innovators**	**Participants**
Download	Yes	Yes	Yes	Yes
Participate	Yes	Yes	Yes	Yes
Plan	Yes	No	Yes	Yes
Commit source code	Yes	No	Yes	Partially

9.3 Limitations of the Research

Interpretivism is the philosophical stand for this thesis. This thesis relied on a single case study as a research method. One of the limitations of this thesis is its dependence on a qualitative method for conducting the research; the use of a single case study. A single case study has been criticized for its dependent on researcher subjectivity and its lack of generalisability. Such criticisms are tied to the general critique for qualitative research as being a subjective research contrast to quantitative research that has been valued for its objectivity (Johnson et al., 2006).

Researcher subjectivity is a valid issue in qualitative research especially when dealing with a single case study in collecting and analysing data. Using different methods will enable the researcher to compare and contrast between different case studies. And this will decrease the level of subjectivity through the analysis.

However, it is believed that subjectivity in this thesis is controlled by avoiding pre-judgement of the research through the use of quantitative data that is used to support the qualitative findings. In addition, objectivity should not be treated as an ultimate value by its own for the research especially when research is focusing on understanding how and where rather than generalizing of findings (Berg et al., 2004). And this thesis aims for understanding how the 'private-collective' model of innovation worked in open source software that is declared under permissive licensing

and the use of a single case study helps the researcher to gain in-depth data that are used to build a detailed picture of the case study and data implications.

Another limitation for this thesis is the use of online data as the only source of data to be collected and analysed. The use of other data such as data generated from interviews would enrich the deep understanding about different findings generated from the analysis of online data.

For example, the analysis of online data revealed that participants in OpenNebula have different incentives while participating in OpenNebula. If interviews to be conducted with these participants, it would be beneficial in exploring the mixed incentives of these different participants in OpenNebula as well as the effect of these mixed incentives on the actions performed by them.

Another limitation for this thesis is a time constraint. Time of this thesis was limited to three years of conducting research. And at the end of each year, an annual review needs to be attended in order to officially finish the requirement of the degree. Accordingly, further data collection and analysis was not possible. However, data collected and analysed so far were enough to conduct a rigorous research and answer the research question.

Despite this limitation, it is believed that online data collected for this thesis were collected and analysed rigorously:

First, the main purpose for analysing data in this thesis is to analyse investments done by the different participants through time. And this can be rigorously identified by using archived data such as online mailing list and development portal rather than interviews.

Second, according to Scott (1990) guidelines who suggested to rely only on “authentic, credible, representative, and meaningful” documents in order to ensure the transferability of the data used in the research. Online data collected in this thesis were published on the official website of the OpenNebula software, were published by an official employee and approved by the project leader, and were neatly fitted into a standard format approved by the OpenNebula team.

Third, different sources of online data were collected; “data sources triangulation” (Thurmond, 2001). Triangulation was crucial in order to gain a justified credibility of the qualitative findings, an inclusive view of the phenomenon under study, as well as a low level of potential biases within the research in order to ensure plausibility.

9.4 Direction for Future Research

The findings of this research open new domains of knowledge that can be explored.

First, in terms of methods, researchers are urged to gather data from different open source software that are different in their technical requirement, industry, types of participants, and types of permissive licenses. And they can

then analyse data in order to identify the different rules that are used.

Such direction of future research is crucial for scaling up our understanding of the 'private-collective' model of innovation. Researchers will not only identify rules (which differ based on the context) that can be applied but also compare and contrast their different findings. Accordingly, they can provide general principles for the needed rules.

Second, in terms of methods, quantitative methods of data collection and analysis can be used in order to explore unrevealed knowledge in open source software under permissive licensing. For example, panel data analysis using STATA software in order to test proposition identified in this thesis.

Third, in terms of conceptualisation, a better theorization of open source software (as a 'commons' that is exposed to appropriation) can be achieved. For example, researchers can identify 'rights' of sponsors and private actors as rights are the product of identifying 'rules' in the commons Schlager and Ostrom (1992). Identifying rights based on the rules that are emerged between participants would suggest a balancing act between sponsors, private actors and community members.

Fourth, further research can be conducted to understand and explain the relationships between IT practices and motivations of participants in open source software. And how these practices would change motivations over time?

References

ALEXY, O. 2008. Putting a Value on Openness: The Effect of Product Source Code Releases on The Market Value of Firms. *Entrepreneurship and Innovation - Organizations, Institutions, Systems and Regions.* Denmark.

AMO, M. 2007. *Open Source Software: Critical Review of Scientific Literature and Other Sources.* Master of Science in Computer Science, Norwegian University of Science and Technology.

ARNOLD, J. E. M. 1998. *Managing forests as common property*, Food & Agriculture Org.

ASLETT, M. 2011. *The trend towards permissive licensing* [Online]. Available: https://blogs.the451group.com/opensource/2011/06/06/the-trend-towards-permissive-licensing/ 2016].

ATKINSON, P., & COFFEY, A. 2004. Analysing documentary realities. *In:* SILVERMAN, D. (ed.) *Qualitative Research: Theory, Method and Practice* 2nd Edition ed. London: Sage Publications.

AVISON, D. E. & PRIES-HEJE, J. 2005. *Research in information systems: A handbook for research supervisors and their students*, Gulf Professional Publishing.

BALDWIN, C. Y. & CLARK, K. B. 2000. *Design rules: The power of modularity*, MIT press.

BALDWIN, C. Y. & CLARK, K. B. 2006. The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? *Management Science,* 52**,** 1116-1127.

BARLEY, S. R. & TOLBERT, P. S. 1997. Institutionalization and structuration: Studying the links between action and institution. *Organization studies,* 18**,** 93-117.

BENTON, T. & CRAIB, I. 2010. *Philosophy of social science: The philosophical foundations of social thought*, Palgrave Macmillan.

BERG, B. L., LUNE, H. & LUNE, H. 2004. *Qualitative research methods for the social sciences*, Pearson Boston, MA.

BERGER, P. L., LUCKMANN, T. & ZIFONUN, D. 2007. *The social construction of reality*, na.

BLACKDUCKSOFTWARE. 2015. *2015 Future of Open Source Survey Results* [Online]. Available:

http://www.slideshare.net/blackducksoftware/2015-future-of-open-source-survey-results 2016].
BLACKSTONE, A. 2012. *Principles of Sociological Inquiry– Qualitative and Quantitative Methods*.
BOLAND JR, R. J., LYYTINEN, K. & YOO, Y. 2007. Wakes of innovation in project networks: The case of digital 3-D representations in architecture, engineering, and construction. *Organization Science,* 18**,** 631-647.
BONACCORSI, A. & ROSSI LAMASTRA, C. 2003. Altruistic individuals, selfish firms? The structure of motivation in Open Source software. *The structure of motivation in Open Source software*.
BOYD, C. O. 1993. Combining qualitative and quantitative approaches. *NLN publications***,** 454-475.
CARDENAS, J.-C., AHN, T. & OSTROM, E. 2004. Communication and co-operation in a common-pool resource dilemma: a field experiment. *Advances in Understanding Strategic Behaviour.* Springer.
CASADESUS-MASANELL, R. & LLANES, G. 2011. Mixed Source. *Management Science,* 57**,** 1212-1230.
CHARMAZ, K. 2006. *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis,* U.S.A, SAGE Publications.
CHUA, W. F. 1986. Radical Developments in Accounting Thought. *Accounting Review,* 61**,** 601-632.
COLAZO, J., FANG, Y. & NEUFELD, D. Development Success in Open Source Software Projects: Exploring the Impact of Copylefted Licenses. Americas Conference on Information Systems, 2005.
COLE, D. H. 2014. Formal Institutions and the IAD Framework: Bringing the Law Back In. *Available at SSRN 2471040*.
COMINO, S. & MANENTI, F. M. 2011. Dual Licensing in Open Source Software Markets. *Information Economics and Policy,* 23**,** 234-242.
CROSSAN, F. 2003. Research philosophy: towards an understanding. *Nurse researcher,* 11**,** 46-55.
CROTTY, M. 1998. *The foundations of social research : meaning and perspective in the research process,* London, Sage Publications.
CROWSTON, K., WEI, K. N., HOWISON, J. & WIGGINS, A. 2012. Free/Libre Open-Source Software Development: What We Know and What We Do Not Know. *Acm Computing Surveys,* 44.
CUNLIFFE, A. L. & JUN, J. S. 2005. The need for reflexivity in public administration. *Administration & Society,* 37**,** 225-242.

CURTIS, G. & COBHAM, D. 2008. *Business information systems: Analysis, design and practice*, Pearson Education.
DAHLANDER, L. 2005. Appropriation and appropriability in open source software. *International Journal of Innovation Management,* 9**,** 259-285.
DAHLANDER, L. & MAGNUSSON, M. 2008. How Do Firms Make Use of Open Source Communities? *Long Range Planning,* 41**,** 629-649.
DAHLANDER, L. & MAGNUSSON, M. G. 2005. Relationships Between Open Source Software Companies and Communities: Observations from Nordic Firms. *Research Policy,* 34**,** 481-493.
DANIEL, S., MIDHA, V., BHATTACHERHJEE, A. & SINGH, S. 2018. Sourcing knowledge in open source software projects: The impacts of internal and external social capital on project success. *The Journal of Strategic Information Systems*.
DE LAAT, P. B. 2007. Governance of Open Source Software: State of the Art. *Journal of Management & Governance,* 11**,** 165-177.
DEMSETZ, H. 1967. Toward A Theory Of Property Rights. *American Economic Review,* 57**,** 347-359.
DENZIN, N. K. & LINCOLN, Y. S. 2008. *Collecting and interpreting qualitative materials*, Sage.
DENZIN, N. K. & LINCOLN, Y. S. 2011. Introduction: The Discipline and Practice of Qualitative Research. *In:* DENZIN, N. K. & LINCOLN, Y. S. (eds.) *The SAGE Handbook of Qualitative Research.* USA: SAGE Publication, Inc.
EASTERBY-SMITH, M., GOLDEN-BIDDLE, K. & LOCKE, K. 2008. Working with pluralism determining quality in qualitative research. *Organizational Research Methods,* 11**,** 419-429.
EISENHARDT, K. M. 1989. BUILDING THEORIES FROM CASE-STUDY RESEARCH. *Academy of Management Review,* 14**,** 532-550.
FEENY, D., HANNA, S. & MCEVOY, A. F. 1996. Questioning the assumptions of the ''tragedy of the commons'' model of fisheries. *Land Economics,* 72**,** 187-205.
FELLER, J., FINNEGAN, P. & HAYES, J. 2008. Delivering the'whole product': business model impacts and agility challenges in a network of open source firms. *Journal of Database Management,* 19**,** 95.
FINLAY, L. 2002. Negotiating the swamp: the opportunity and challenge of reflexivity in research practice. *Qualitative research,* 2**,** 209-230.

FLYVBJERG, B. 2006. Five misunderstandings about case-study research. *Qualitative Inquiry,* 12**,** 219-245.
FRANCK, E. & JUNGWIRTH, C. 2003. Reconciling rent-seekers and donators – The governance structure of open source. *Journal of Management and Governance,* 7**,** 401-421.
GATES, B. 1976. *An Open Letter to Hobbiyst* [Online]. Homebrew Computer Club Newsletter. Available: http://www.digibarn.com/collections/newsletters/homebrew/V2_01/gatesletter.html 2013].
GHOSH, R. A. 2005. Understanding free software developers: Findings from the FLOSS study. *Perspectives on free and open source software***,** 23-46.
GOLDKUHL, G. What kind of pragmatism in information systems research. AIS SIG Prag Inaugural Meeting, 2008.
GOLDKUHL, G. 2012. Pragmatism vs interpretivism in qualitative information systems research. *European Journal of Information Systems,* 21**,** 135-146.
GRUBER, M. & HENKEL, J. 2006. New ventures based on open innovation – an empirical analysis of start-up firms in embedded Linux. *International Journal of Technology Management,* 33**,** 356-372.
GUBA, E. G. & LINCOLN, Y. S. 1994. Competing Paradigms in Qualitative Research. *Handbook of qualitative research,* 2**,** 163-194.
HARDIN, G. 1977. The Tragedy of the Commons. *In:* HARDIN, G. & BADEN, J. (eds.) *Managing the Commons.* New York: W. H. Freeman and Company.
HARS, A. & OU, S. Working for Free? Motivations for Participating in Open Source Projects. 34th Hawaii International Conference on System Sciences, 2001.
HAUGE, Ø., AYALA, C. & CONRADI, R. 2010. Adoption of open source software in software-intensive organizations–A systematic literature review. *Information and Software Technology,* 52**,** 1133-1154.
HECKER, F. 1999. Setting up shop: The business of open-source software. *IEEE software,* 16**,** 45.
HEMETSBERGER, A. 2002. Fostering Cooperation on the Internet: Social Exchange Processes in Innovative Virtual Consumer Communities. *In:* BRONIARCZYK, S. M. & NAKAMOTO, K. (eds.) *Advances in Consumer Research, Volume Xxix.*
HERTEL, G., NIEDNER, S. & HERRMANN, S. 2003. Motivation of Software Developers in Open Source

Projects: An Internet-Based Survey of Contributors to the Linux Kernel. *Research Policy,* 32**,** 1159-1177.
HESS, C. & OSTROM, E. 2005. A Framework for Analyzing the Knowledge Commons. *In:* HESS, C. & OSTROM, E. (eds.) *Understanding Knowledge as a Commons: from Theory to Practice.* Cambridge MA: The MIT Press.
HIBBERT, P., COUPLAND, C. & MACINTOSH, R. 2010. Reflexivity: recursion and relationality in organizational research processes. *Qualitative Research in Organizations and Management: An International Journal,* 5**,** 47-62.
ISAAC, R. M. & WALKER, J. M. 1988. Communication and free-riding behavior: The voluntary contribution mechanism. *Economic inquiry,* 26**,** 585-608.
JICK, T. D. 1979. Mixing qualitative and quantitative methods: Triangulation in action. *Administrative science quarterly***,** 602-611.
JOHNSON, P., BUEHRING, A., CASSELL, C. & SYMON, G. 2006. Evaluating qualitative management research: Towards a contingent criteriology. *International Journal of Management Reviews,* 8**,** 131-156.
JOHNSON, P. & CASSELL, C. 2001. Epistemology and work psychology: New agendas. *Journal of Occupational and Organizational Psychology,* 74**,** 125-143.
JOHNSON, R. B. & ONWUEGBUZIE, A. J. 2004. Mixed methods research: A research paradigm whose time has come. *Educational researcher,* 33**,** 14-26.
KENNETH, C., LAUDON, L. & LAUDON, J. P. 2001. *Management Information Systems: Organization and Technology in Networked Enterprise*, Higher Education Press.
KLEIN, H. K. 2004. Seeking the New and the Critical in Critical Realism: Dé´jà` vu? *Information and Organization,* 14**,** 123-144.
KOTHARI, C. R. 2004. *Research methodology: Methods and techniques*, New Age International.
KUK, G. 2006. Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List. *Management Science,* 52**,** 1031-1042.
LAKHANI, K. R. & WOLF, R. 2005. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. *Perspectives on free and open source software,* 1**,** 3-22.
LATTEMANN, C. & STIEGLITZ, S. Framework for Governance in Open Source Communities. 38th

Hawaii International Conference on System Sciences, 2005.
LEE, G. K. & COLE, R. E. 2003. From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development. *Organization Science,* 14**,** 633-649.
LERNER, J., PATHAK, P. A. & TIROLE, J. 2006. The Dynamics of Open-Source Contributors. *American Economic Review,* 96**,** 114-118.
LERNER, J. & TIROLE, J. 2001. The Open Source Movement: Key Research Questions. *European Economic Review,* 45**,** 819-826.
LERNER, J. & TIROLE, J. 2002. Some Simple Economics of Open Source. *Journal of Industrial Economics,* 50**,** 197-234.
LERNER, J. & TIROLE, J. 2005a. The Economics of Technology Sharing: Open Source and Beyond. *Journal of Economic Perspectives,* 19**,** 99-120.
LERNER, J. & TIROLE, J. 2005b. The Scope of Open Source Licensing. *Journal of Law Economics & Organization,* 21**,** 20-56.
LINCOLN, Y. S. & GUBA, E. G. 1985. *Naturalistic inquiry*, Sage.
LLORENTE, I. M. 2014. *OpenNebula vs. OpenStack: User Needs vs. Vendor Driven* [Online]. Available: http://opennebula.org/opennebula-vs-openstack-user-needs-vs-vendor-driven/ 2013].
LOVEJOY, T. E. 2006. Protected Areas: A Prism for a Changing World. *Trends in Ecology & Evolution,* 21**,** 329-333.
MANN, C. & STEWART, F. 2000. *Internet communication and qualitative research: A handbook for researching online*, Sage.
MARKUS, M. 2007. The Governance of Free/Open Source Software Projects: Monolithic, Multidimensional, or Configurational? *Journal of Management and Governance,* 11**,** 151-163.
MARKUS, M. & SILVER, M. 2008. A foundation for the study of IT effects: A new look at DeSanctis and Poole's concepts of structural features and spirit. *Journal of the Association for Information systems,* 9**,** 609.
MARSTON, S., LI, Z., BANDYOPADHYAY, S., ZHANG, J. & GHALSASI, A. 2011. Cloud computing—The business perspective. *Decision Support Systems,* 51**,** 176-189.

MICROSOFT. *What is a Driver?* [Online]. Available: http://windows.microsoft.com/en-gb/windows/what-is-driver#1TC=windows-7 2015].
MILOJICIC, D., LLORENTE, I. M. & MONTERO, R. S. 2011. OpenNebula A Cloud Management Tool. *Ieee Internet Computing,* 15**,** 11-14.
MINGERS, J. 2001. Combining IS Research Methods: Towards a Pluralist Methodology. *Information Systems Research,* 12**,** 240-259.
MITCBELL, E. S. 1986. Multiple triangulation: a methodology for nursing science. *Advances in nursing science,* 8**,** 18-26.
MYERS, M. D. 2013. *Qualitative Research in Business & Management*, SAGE Publications.
NAGLE, F. 2018. Open Source Software and Firm Productivity. *Management Science*.
NAKAKOJI, K., YAMAMOTO, Y., NISHINAKA, Y., KISHIDA, K. & YE, Y. Evolution patterns of open-source software systems and communities. Proceedings of the international workshop on Principles of software evolution, 2002. ACM, 76-85.
NAU, D. S. 1995. Mixing methodologies: can bimodal research be a viable post-positivist tool? *The Qualitative Report,* 2**,** 1-6.
NOLAN, M. & BEHI, R. 1995. Triangulation: the best of all worlds? *British Journal of Nursing,* 4**,** 829-832.
O'MAHONY, S. 2003. Guarding the Commons: How Community Managed Software Projects Protect their Work. *Research Policy,* 32**,** 1179-1198.
O'MAHONY, S. 2007. The Governance of Open Source Initiatives: What Does it Mean to be Community Managed? *Journal of Management & Governance,* 11**,** 139-150.
OLSON, M. 1967. *The Logic of Collective Action,* Cambridge, MA., Harvard University Press.
OPENNEBULASYSTEMS. 2012. *Packaging OpenNebula 3.6 (Lagoon)* [Online]. Available: http://archives.opennebula.org/software:rnotes:rn-rel3.6 2015].
OPENNEBULASYSTEMS. 2013. *Command Line Tweaks for OpenNebula 4.0* [Online]. Available: http://opennebula.org/command-line-tweaks-for-opennebula-4-0/ 2015].
OPENNEBULASYSTEMS. 2015a. *Advanced Contextualization* [Online]. Available: http://docs.opennebula.org/4.12/user/virtual_machine_setup/cong.html 2015].

OPENNEBULASYSTEMS. 2015b. *Amazon EC2 Driver* [Online]. Available: http://docs.opennebula.org/4.12/advanced_administration/cloud_bursting/ec2g.html 2015].
OPENNEBULASYSTEMS. 2015c. *Authentication Driver* [Online]. Available: http://docs.opennebula.org/4.12/integration/infrastructure_integration/devel-auth.html 2015].
OPENNEBULASYSTEMS. 2015d. *Basic Contextualization* [Online]. Available: http://docs.opennebula.org/4.12/user/virtual_machine_setup/bcont.html 2015].
OPENNEBULASYSTEMS. 2015e. *Java OpenNebula Cloud API* [Online]. Available: http://docs.opennebula.org/4.12/integration/system_interfaces/java.html 2015].
OPENNEBULASYSTEMS. 2015f. *Monitoring Driver* [Online]. Available: http://docs.opennebula.org/4.12/integration/infrastructure_integration/devel-im.html 2015].
OPENNEBULASYSTEMS. 2015g. *Networking Driver* [Online]. Available: http://docs.opennebula.org/4.12/integration/infrastructure_integration/devel-nm.html 2015].
OPENNEBULASYSTEMS. 2015h. *OneFlow* [Online]. Available: http://docs.opennebula.org/4.12/advanced_administration/application_flow_and_auto-scaling/oneapps_overview.html 2015].
OPENNEBULASYSTEMS. 2015i. *OneFlow Server API* [Online]. Available: http://docs.opennebula.org/4.12/integration/system_interfaces/appflow_api.html 2015].
OPENNEBULASYSTEMS. 2015j. *OneGate* [Online]. Available: http://docs.opennebula.org/4.12/advanced_administration/application_insight/onegate_overview.html 2015].
OPENNEBULASYSTEMS. 2015k. *OpenNebula 3.4 (Wild Duck)* [Online]. Available: http://archives.opennebula.org/software:rnotes:rn-rel3.4 2015].
OPENNEBULASYSTEMS. 2015l. *OpenNebula Quality Assurance* [Online]. Available: http://opennebula.org/software/testing/ 2015].
OPENNEBULASYSTEMS. 2015m. *OpenNebula Sunstone: The Cloud Operations Center 2.2* [Online]. Available:

http://archives.opennebula.org/documentation:archives:rel2.2:sunstone 2015].
OPENNEBULASYSTEMS. 2015n. *OpenNebula Zones Overview 3.0* [Online]. Available: http://archives.opennebula.org/documentation:archives:rel3.0:ozones 2015].
OPENNEBULASYSTEMS. 2015o. *Ruby OpenNebula Cloud API* [Online]. Available: http://docs.opennebula.org/4.12/integration/system_interfaces/ruby.html 2015].
OPENNEBULASYSTEMS. 2015p. *Scheduler* [Online]. Available: http://docs.opennebula.org/4.12/administration/references/schg.html 2015].
OPENNEBULASYSTEMS. 2015q. *Self-service Cloud View* [Online]. Available: http://docs.opennebula.org/4.12/administration/sunstone_gui/cloud_view.html 2015].
OPENNEBULASYSTEMS. 2015r. *Storage Driver* [Online]. Available: http://docs.opennebula.org/4.12/integration/infrastructure_integration/sd.html 2015].
OPENNEBULASYSTEMS. 2015s. *Virtualization Driver* [Online]. Available: http://docs.opennebula.org/4.12/integration/infrastructure_integration/devel-vmm.html 2015].
OPENNEBULASYSTEMS. 2015t. *VMwar vCenter Drivers* [Online]. Available: http://docs.opennebula.org/4.10/administration/virtualization/vcenterg.html 2015].
OPENNEBULASYSTEMS. 2015u. *Windows Contextualization* [Online]. Available: http://docs.opennebula.org/4.12/user/virtual_machine_setup/windows_context.html 2015].
OPENNEBULASYSTEMS. 2015v. *XML-RPC API* [Online]. Available: http://docs.opennebula.org/4.12/integration/system_interfaces/api.html 2015].
OPENNEBULASYSTEMS. 2016a. *OpenNebula AppMarket* [Online]. Available: http://marketplace.c12g.com/appliance [Accessed 2015.
OPENNEBULASYSTEMS. 2016b. *OpenNebula Documentation* [Online]. Available: http://opennebula.org/documentation/ 2015].
ORLIKOWSKI, W. J. & BAROUDI, J. J. 1991. Studying Information Technology in Organizations: Research

Approaches and Assumptions. *Information systems research,* 2**,** 1-28.
OSTROM, E. 1990. *Governing the Commons: The Evolution of Institutions for Collective Action,* New York, Cambridge University Press.
OSTROM, E. 2003. How Types of Goods and Property Rights Jointly Affect Collective Action. *Journal of Theoretical Politics,* 15**,** 239-270.
OSTROM, E. 2007. Institutional rational choice: An assessment of the institutional analysis and development framework.
OSTROM, E. 2009. Design Principles of Robust Property Rights Institutions: What Have We Learned? *In:* INGRAM, G. K. & HONG, Y. H. (eds.) *Property Rights and Land Policies.* Cambridge,MA: Lincoln Institute of Land Policy.
OSTROM, E. 2010. Polycentric systems for coping with collective action and global environmental change. *Global Environmental Change-Human and Policy Dimensions,* 20**,** 550-557.
OSTROM, E. & HESS, C. 2007. Understanding knowledge as a commons. *From Theory to Practice, Massachusetts*.
OSTROM, E. & WALKER, J. 1991. Communication in a commons: cooperation without external enforcement. *Laboratory research in political economy***,** 287-322.
PLATT, J. 1981. EVIDENCE AND PROOF IN DOCUMENTARY RESEARCH.1. SOME SPECIFIC PROBLEMS OF DOCUMENTARY RESEARCH. *Sociological Review,* 29**,** 31-52.
RAYMOND, E. 1999a. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary,* USA, O'Reilly Media, Inc.
RAYMOND, E. 1999b. *The Magic Cauldron* [Online]. Available: http://www.catb.org/esr/writings/magic-cauldron/magic-cauldron.html#toc1 2014].
ROBERTS, J. A., HANN, I. H. & SLAUGHTER, S. A. 2006. Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science,* 52**,** 984-999.
SANTOS, C., KUK, G., KON, F. & PEARSON, J. 2013. The attraction of contributors in free and open source software projects. *The Journal of Strategic Information Systems,* 22**,** 26-45.

SCACCHI, W. 2002. Understanding the Requirements for Developing Open Source Software Systems. *IEE Proceedings - Software,* 149**,** 24-39.
SCHILLING, M. A. 2000. Toward a general modular systems theory and its application to interfirm product modularity. *Academy of management review,* 25**,** 312-334.
SCHLAGER, E. & OSTROM, E. 1992. Property-Rights Regimes and Natural - Resources - A Conceptual Analysis. *Land Economics,* 68**,** 249-262.
SCHWEIK, C. M. 2007. Free/open-source software as a framework for establishing commons in science. *Understanding Knowledge as a Commons***,** 277.
SCOTT, D. 1990. Practice Wisdom - The neglected source of practice research. *Social Work,* 35**,** 564-568.
SEN, R., SUBRAMANIAM, C. & NELSON, M. L. 2008. Determinants of the Choice of Open Source Software License. *Journal of Management Information Systems,* 25**,** 207-239.
SEN, R., SUBRAMANIAM, C. & NELSON, M. L. 2011. Open Source Software Licenses: Strong-Copyleft, Non-Copyleft, or Somewhere in Between? *Decision Support Systems,* 52**,** 199-206.
SHAH, S. K. 2006. Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science,* 52**,** 1000-1014.
SIGGELKOW, N. 2007. Persuasion with case studies. *Academy of Management Journal,* 50**,** 20-24.
SILVERMAN, D. 2006. *Interpreting qualitative data: Methods for analyzing talk, text and interaction*, Sage.
SILVERMAN, D. 2013. *Doing Qualitative Research,* Great Britain, SAGE Publications.
SIMON, H. A. 1994. The bottleneck of attention: connecting thought with motivation.
SKOK, M. 2013. *2013 Future of Open Source - 7th Annual Survey results* [Online]. Available: http://www.mjskok.com/resource/2013-future-open-source-7th-annual-survey-results.
SRIVASTAVA, P. & HOPWOOD, N. 2009. A practical iterative framework for qualitative data analysis. *International journal of qualitative methods,* 8**,** 76-84.
STAIR, R. & REYNOLDS, G. 2013. *Principles of information systems*, Cengage Learning.
STALLMAN, R. 1985a. *What is Free Software?* [Online]. Available: http://www.gnu.org/philosophy/free-sw.html [Accessed 03/03/2014 2014].

STALLMAN, R. 1985b. *Why Open Source Misses the Point of Free Software.* [Online]. Available: http://www.gnu.org/philosoghy/open-source-misses-the-point.html [Accessed 03/03/2014 2014].
STOL, K. J. & BABAR, M. A. 2009. Reporting empirical research in open source software: the state of practice. *In:* BOLDYREFF, C., CROWSTON, K., LUNDELL, B. & WASSERMAN, A. I. (eds.) *Open Source Ecosystem: Diverse Communities Interacting.*
STRAUSS, A. L. & CORBIN, J. 1990. *Basics of Qualitative Research, Grounded Theory Procedures and Techniques,* New York, SAGE Publications.
SUDDABY, R. & GREENWOOD, R. 2009. Methodological Issues in Researching Institutional Change. *In:* SYMON, G. & CASSELL, C. (eds.) *Qualitative Organizational Research: Core Methods and Current Challenges.* Los Angeles: Sage Publications.
TEECE, D. J. 1986. PROFITING FROM TECHNOLOGICAL INNOVATION - IMPLICATIONS FOR INTEGRATION, COLLABORATION, LICENSING AND PUBLIC-POLICY. *Research Policy,* 15**,** 285-305.
THOMAS, D. R. 2006. A general inductive approach for analyzing qualitative evaluation data. *American journal of evaluation,* 27**,** 237-246.
THORPE, R. & HOLT, R. 2007. *The Sage dictionary of qualitative management research*, Sage.
THURMOND, V. A. 2001. The point of triangulation. *Journal of nursing scholarship,* 33**,** 253-258.
TURBAN, E., RAINER, R. K. & POTTER, R. E. 2007. Introduction to Information Systems: Supporting and Transforming Business.
VON HIPPEL, E. 1994. Sticky Information and the Locus of Problem- Solving - Implications for Innovation. *Management Science,* 40**,** 429-439.
VON HIPPEL, E. 2001. Innovation by User Communities: Learning from Open-Source Software. *Mit Sloan Management Review,* 42**,** 82-86.
VON HIPPEL, E. 2007. *The sources of innovation*, Springer.
VON HIPPEL, E. & VON KROGH, G. 2003. Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science,* 14**,** 209-223.
VON HIPPEL, E. & VON KROGH, G. 2006. Free revealing and the private-collective model for innovation incentives. *R&D Management,* 36**,** 295-306.

VON KROGH, G., HAEFLIGER, S., SPAETH, S. & WALLIN, M. W. 2012. Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *MIS Quarterly,* 36**,** 649-676.

VON KROGH, G. & SPAETH, S. 2007. The Open Source Software Phenomenon: Characteristics That Promote Research. *Journal of Strategic Information Systems,* 16**,** 236-253.

WALSHAM, G. 1993. *Interpreting information systems in organizations*, Wiley Chichester.

WALSHAM, G. 1995. Interpretive case studies in IS research: nature and method. *European Journal of information systems,* 4**,** 74-81.

WARING, T. & MADDOCKS, P. 2005. Open Source Software implementation in the UK public sector: Evidence from the field and implications for the future. *International Journal of Information Management,* 25**,** 411-428.

WEST, J. & O'MAHONY, S. Contrasting Community Building in Sponsored and Community Founded Open Source Projects. HICSS - Hawaii International Conference on System Sciences, 2005a.

WEST, J. & O'MAHONY, S. Contrasting Community Building in Sponsored and Community Founded Open Source Projects. Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05), 2005b Hawaii. IEEE.

WEST, J. & O'MAHONY, S. 2008. The Role of Participation Architecture in Growing Sponsored Open Source Communities. *Industry and Innovation,* 15**,** 145-168.

YAMAUCHI, Y., YOKOZAWA, M., SHINOHARA, T. & ISHIDA, T. 2000. Collaboration with Lean Media: How Open Source Software Succeds. *ACM 2000 Conf. Comput. Supported Cooperative Work.* Philadelphia: ACM Press.

YIN, R. K. 2014. *Case Study Research: Design and Methods*, SAGE Publications.

YOO, Y. J., HENFRIDSSON, O. & LYYTINEN, K. 2010. The New Organizing Logic of Digital Innovation: An Agenda for Information Systems Research. *Information Systems Research,* 21**,** 724-735.

ZEITLYN, D. 2003. Gift Economies in the Development of Open Source Software: Anthropological Reflections. *Research Policy,* 32**,** 1287-1291.

ZHU, C. 2011. *Authoring Collaborative Projects: A Study of Intellectual Property and Free and Open Source*

Software (FOSS) Licensing Schemes from a Relational Contract Perspective. Doctor of Philosophy, The London School of Economics.

Appendix A: The Analysis of Contributions in OpenNebula

Table A. 1 The analysis of the different categories about contributions performed by participants

	Category Label	Category Description	Examples of Texts associated with Category
1	Announce new features in the project	This category represents actions where participants declare (in the mailing list) the addition of new features (in the development page) within the project.	AnonyAI1 said that they are going to change the management system for OpenNebula from "TRAC" to "Readme" because this will increase the functionality that will be useful for the development of the project. Accordingly, and due to this migration, AnonyAO1 announced that the portal of the project will be down for that day.
			AnonyAO4 announces the availability of a new release for the project. He encourages other participants to use the new releases and provide feedback if possible.
2	Correcting documentation	This category represents actions where participants corrected wrong information in the documentation available for the project and share these corrections in the mailing list.	AnonyAO1 corrected documentation related to "cluster" infrastructure in OpenNebula
			AnonyAO4 updated the "XEN" documentation based on the modifications done to the source code
			AnonyAO5 modified the documentation related to the "etables" in "KVM"

			machines
			AnonyAI4 corrected the documentation related to failures of virtual machine. He mentioned the exact requirements needed and the steps needed to recover failed virtual machines
3	Fix software bugs	This category represents actions where participants fix software bugs in the source code of the project and share these fixes in the mailing list	One participant sent an email about an error message appeared on his computer network after he implemented OpenNebula virtual machine. After discussions between AnonyAO2 and that participant, AnonyAO2 found that the error message appeared was explained by a software bug existed in the source code of the project. Accordingly, AnonyAO2 resolved this software bug and reflected the changes in the source code of the project.
4	Develop enhancements in the project	This category represents actions where participants develop a new source code for the project and share it in the mailing list or the development page for the project.	AnonyAO3 developed a new source code in order to enhance the search in the XML library of the project
			AnonyAI1 helps in developing a new source code to integrate ESXi 4.0 server with OpenNebula project.
			AnonyAI3 helps in developing a new source code related to OpenVZ virtual machine and integrates this code with the OpenNebula source code
			AnonyAI1 offered help in developing and testing source code related to using “Ceph” as a distributed file system in OpenNebula.
5	Develop new documentation	This category represents actions where participants write a new documentation for the project and share this documentation in the mailing list	AnonyAO2 added a new documentation related to the accounting information for the project in order to enhance the implementation and usability of the project

6	Share detailed technical knowledge	This category represents action where participants share detailed knowledge with each other such as technical solution, technical patches, steps for implementations…etc	AnonyAO2 shared a fine and detailed knowledge about using Java in OpenNebula project as a response to the huge amount of questions from different participants.
			AnonyAE4 replied to an email that requested some details about the use of "oneadmin" command in OpenNebula. . AnonyAE4 extensively explain the use of this command, he explained the needed file/folder/directory. He also explained the group that he included in this command. In addition, he explained the procedure he followed to add access permissions through this command. He lastly provides needed documentation that will help in a better understanding as well
			AnonyAE3 shared his technical knowledge about "contextualization" function in OpenNebula. He explained how to use all variables defined in the template file for the virtual machines, the definition of the virtual network used, the scripts (lines of code) he had written to improve the functionality of contextualization in his own network.
			AnonyAC3 shared a data store problem with OpenNebula. in his email, he shared the network setting for his computer network, the log file of the system, and the components used in the project
			A participant requested a comparison between the use of two file systems, namely MooseFS and GlusterFS , to be used in OpenNebula. Several technical concepts, such as redundancy, virtualization, images connection with virtual machines, the existence of support, distributed replication servers, up to time reliability, and metadata server, are discussed by AnonyAE4 and more file systems, such as Lustre, Shipping dog, and ExtremeFS, are added to the

			comparison process. The discussions and comparisons were fruitful and reflected in the OpenNebula official blog website to be used as agreed upon by interested users
			AnonyAI5 explained the configuration of the OpenVSwitch he was deploying within the project. Moreover, he explained the network masking scheme he was using. AnonyAI5 shared this information in order to provide the big picture of the technical problem he faced while implementing the project.
			AnonyAI2 shared his understanding about creating a network of virtual machines and his vision of classifying Internet Addresses (IPs) between these components. He gave three options for the network with pros and cons for each option. He explained that his suggestions are based on his understanding about the project from the available documentation as well as following emails in the mailing list but not from own experience.
7	Explain uses of the project components	This category represents actions where participants clarify how parts of the projects can be implemented in the business	AnonyAO4 explained the use of Libvirt system in OpenNebula. He said that the implementation of the libvirt API provides an abstraction of a whole cluster of resources used in the project. In this way, one can use any libvirt tool (e.g. virsh, virt-manager) and libvirt XML domain descriptions with OpenNebula at a distributed level
8	Explain technical ideas	This category represents actions where participants share their technical ideas in the mailing list and clarify these technical ideas; advantages and justifications for implementations.	Based on the request of a participant to do modifications for the image base for virtual machines in OpenNebula, AnonyAO1 explained that currently they do not support virtual images in other places than files. And the reason behind this decision is that they cannot think of any other scenario were virtual images were made available as logical volumes to all the computers on the physical cluster. So they decide that their approach will be to

			recommend one base image and then cloning this one whenever required
			A participant requested a comparison between the use of two file systems, namely MooseFS and GlusterFS , to be used in OpenNebula. Several technical concepts, such as redundancy, virtualization, images connection with virtual machines, the existence of support, distributed replication servers, up to time reliability, and metadata server, are discussed by AnonyAE4 and more file systems, such as Lustre, Shipping dog, and ExtremeFS, are added to the comparison process. The discussions and comparisons were fruitful and reflected in the OpenNebula official blog website to be used as agreed upon by interested users
9	Determine project roadmap	This category represents actions where participants decide the upcoming technical plan for the project different releases.	AnonyAO2 reviewed different suggestions from different participants related to the use of persistent virtual machines with OpenNebula. AnonyAO2 decided that, the “oneimage clone” shall be added as a feature to the project roadmap
10	Develop private maintenance releases	This category represents actions where participants change the license of the project releases from open source license (Apache v 2.0) into a private license	Releases 3.2.2, 3.4.2, 3.4.3, 3.4.4, 3.6.1, 3.6.2, 3.6.2, and 3.8.2 are private releases in which sponsors resolve some bugs and do some enhancements to the source code in order to fit their private users’ needs
11	Solve problems via live chat	This category represents actions where participants choose a private live chat instead of the public mailing list in order to solve problems faced by other participants.	AnonyAG3 had a live chat with the OpenNebula development team to provide support and help. This chat is part of the service level agreement signed with the sponsor
12	Share the computer network settings	This category represents actions where participants share the technical characteristics of their computer network	AnonyAC3, through one of his emails, explained the computer network he is using with OpenNebula. He described that the network consists of 3 data stores, 2 VMware ESXi hosts, and a collection of SAN desks. And then he

		with other participants	explained the problem he is facing
13	Help others by explaining network settings	This category represents actions where participants help other participants to solve implementations problems by modifying the network setting of the project (rather than the source code of the project)	A participant discussed the idea of assigning Internet Addresses (IPs) from both OpenNebula and "DHCP" server. AnonyAC3 commented on this idea by providing a technique that can be used in this case. He used this technique as a temporary solution as OpenNebula has not provided a permanent solution yet
14	Get engaged with others discussions	This category represents actions where participants participated in others' discussions through sharing opinions or providing solutions and suggestions	A participant discussed the idea of assigning Internet Addresses (IPs) from both OpenNebula and "DHCP" server. AnonyAC3 commented on this idea by providing a technique that can be used in this case. He used this technique as a temporary solution as OpenNebula has not provided a permanent solution yet
			AnonyAE4 replied to an email that requested some details about the use of "oneadmin" command in OpenNebula. . AnonyAE4 extensively explain the use of this command, he explained the needed file/folder/directory. He also explained the group that he included in this command. In addition, he explained the procedure he followed to add access permissions through this command. He lastly provides needed documentation that will help in a better understanding as well
			AnonyAI4 did not get much attention from the OpenNebula team so AnonyAI4 tries to get engaged by replying to others enquiries (depending on documentation guidelines) and get engaged in their context; sharing mutual knowledge and gaining information from the two way comments.
15	Share knowledge	This category represents actions where participants	Among the emails sent between a participant and AnonyAI1 about the

	about a system to support suggested ideas	suggest technical ideas to be implemented in the project. and support their suggestions with examples of information systems that can help in effectively implementing their ideas.	partnership between both “RedHat” and “Eucalyptus”. AnonyAI1 was wondering if this partnership can be linked to OpenNebula project as well and what would be the effects
			A participant requested a comparison between the use of two file systems, namely MooseFS and GlusterFS , to be used in OpenNebula. Several technical concepts, such as redundancy, virtualization, images connection with virtual machines, the existence of support, distributed replication servers, up to time reliability, and metadata server, are discussed by AnonyAE4 and more file systems, such as Lustre, Shipping dog, and ExtremeFS, are added to the comparison process. The discussions and comparisons were fruitful and reflected in the OpenNebula official blog website to be used as agreed upon by interested users
			AnonyAI1 shared information about “cgroups” in order to encourage the OpenNebula team to take his suggestion into consideration.
16	Reveal own experience in the project	This category represents actions where participants share their experiences in the mailing list with other participants regarding the uses and implementations of the project	AnonyAI3 shares solutions related to image management, wrong image information, suspending state for virtual machines, problems of CPU storage…etc
			AnonyAC3 shares solutions related to Deploying Windows host from vmdk file, Assigning IP address through DHCP servers, Use of Ceph/RBD for system datastore, Virtual machines deletion on ESXi hosts…etc
			AnonyAE4 replied to an email that requested some details about the use of “oneadmin” command in OpenNebula. . AnonyAE4 extensively explain the use of this command, he explained the needed file/folder/directory. He also

			explained the group that he included in this command. In addition, he explained the procedure he followed to add access permissions through this command. He lastly provides needed documentation that will help in a better understanding as well
			AnonyAE3 share solutions related to bug for LDAP implementation, create CentOS base image on OpenNebula, live migration…etc.
17	Reveal code contributions	This category represents actions where participants share the source code that they have developed. They share it in the mailing list or the development page of the project.	AnonyAI3 shares solutions related to image management, wrong image information, suspending state for virtual machines, problems of CPU storage…etc
			AnonyAE3 share solutions related to bug for LDAP implementation, create CentOS base image on OpenNebula, live migration…etc.
			AnonyAC3 highlighted that virtual machines cannot boot properly after adding “pcibridge stanza” appliance to the template of virtual machines. As a way to overcome this problem, he developed a solution and asked this solution to be added as a new feature to the project
			AnonyAE5 test the implementation of OpenSUSE 11.1 with OpenNebula. He noticed that the problem with the implementation was due to “gem” file. AnonyAE5 worked on solving the problem and shared the solution in the mailing list.
			AnonyAC3 shares solutions related to Deploying Windows host from vmdk file, Assigning IP address through DHCP servers, Use of Ceph/RBD for system

			datastore, Virtual machines deletion on ESXi hosts…etc
18	Upload patches for software bugs	This category represented actions where participants share source code in order to solve software bugs existed in the project. This source code is called a patch. Participants share patches in the mailing list or the development page of the project.	AnonyAI3 shares solutions related to image management, wrong image information, suspending state for virtual machines, problems of CPU storage…etc
			AnonyAC3 shares solutions related to Deploying Windows host from vmdk file, Assigning IP address through DHCP servers, Use of Ceph/RBD for system datastore, Virtual machines deletion on ESXi hosts…etc
			AnonyAC3 highlighted that virtual machines cannot boot properly after adding "pcibridge stanza" appliance to the template of virtual machines. As a way to overcome this problem, he developed a solution and asked this solution to be added as a new feature to the project
			AnonyAE3 share solutions related to bug for LDAP implementation, create CentOS base image on OpenNebula, live migration…etc.
19	Reveal technical solution	This category represents actions where participants share their suggested solutions to problems faced by other participants.	AnonyAI3 shares solutions related to image management, wrong image information, suspending state for virtual machines, problems of CPU storage…etc
			AnonyAE4 replied to an email that requested some details about the use of "oneadmin" command in OpenNebula. . AnonyAE4 extensively explain the use of this command, he explained the needed file/folder/directory. He also explained the group that he included in this command. In addition, he explained the procedure he followed to add access permissions through this command. He lastly provides needed documentation that will help in a better

			understanding as well
			AnonyAC3 shares solutions related to Deploying Windows host from vmdk file, Assigning IP address through DHCP servers, Use of Ceph/RBD for system datastore, Virtual machines deletion on ESXi hosts…etc
			AnonyAC3 highlighted that virtual machines cannot boot properly after adding "pcibridge stanza" appliance to the template of virtual machines. As a way to overcome this problem, he developed a solution and asked this solution to be added as a new feature to the project
			AnonyAE3 share solutions related to bug for LDAP implementation, create CentOS base image on OpenNebula, live migration…etc.
20	Suggest new idea or feature	This category represents actions where participants suggested the addition of new feature or idea to the project	AnonyAC4 suggests the addition of dynamic firewall in OpenNebula project. Moreover He suggested the addition of OCCI server as a new layer to be added above OpenNebula.
21	Suggest new system integration	This category represents actions where participants suggest the addition of new information systems to be integrated with Core OpenNebula.	AnonyAI1 suggested the integration of "Eucalyptus" of Amazon's S3 with OpenNebula.
			AnonyAI3 suggested the addition of OpenVZ. He helped in developing a new source code related to OpenVZ virtual machine and integrated this code with OpenNebula source code.
			regarding the use of "InfiniBand" technology with OpenNebula in order to achieve high performance computing, AnonyAI2 suggests several technologies to be used such as Linux iSER, TGTD, and Mellanox VSA

			AnonyAE5 suggested the integration of ElasticFox system with the project.
			AnonyAI1 suggested the development of ESXi server. He helped in developing a new source code to integrate ESXi 4.0 server with OpenNebula project.
22	Test others contributions	This category represents actions where participants test the quality of source code developed by others. They test the source code in order to be free of software bugs and achieve its intended functions.	AnonyAI1 offered help in developing and testing source code related to using "Ceph" as a distributed file system in OpenNebula
23	Add new feature	This category represents actions where participants add a new feature to core OpenNebula. They add the new feature on the mailing list or directly to the development page of the project.	AnonyAC3 highlighted that virtual machines cannot boot properly after adding "pcibridge stanza" appliance to the template of virtual machines. As a way to overcome this problem, he developed a solution and asked this solution to be added as a new feature to the project
24	Request details about a technical problem	This category represents actions where participants ask each other in the mailing list to explain the error message they are facing while implementing the project	An email was sent about a connectivity problem for virtual machines implemented with OpenNebula. AnonyAI4 replied to this email by asking questions related to the definition of virtual machines in OpenNebula, the computer ports used for the virtual machines, isolation of virtual machines into a private network, and the output of the list reports resulted in OpenNebula
25	Comment and ask for feedback	This category represents actions where participants commented on others emails and ask for feedback about their comments	AnonyAI4 comment on using the "bash" technique in Linux systems and asked for developers' feedback about this technique and its usability in OpenNebula.
26	Use documentation to	This category represents actions where participants refer	AnonyAI4 refers in more than 30 emails that he is following the available

	express ideas	to documentation of the project or other information systems to support their discussions in the mailing list	documentation to answer and respond to questions as well as getting engaged with the different discussions.
			AnonyAI2 shared his understanding about creating a network of virtual machines and his vision of classifying Internet Addresses (IPs) between these components. He gave three options for the network with pros and cons for each option. He explained that his suggestions are based on his understanding about the project from the available documentation as well as following emails in the mailing list but not from own experience
27	Report fault documentation	This category represents actions where participants corrected wrong information in documentation of the project.	AnonyAI4 corrected the documentation related to failures of virtual machine. He mentioned the exact requirements needed and the steps needed to recover failed virtual machines
28	Share opinions	This category represents actions where participants share their opinions about a problem or an information system in the mailing list	AnonyAI2 shared his opinion about a suggestion provided by another participant related to the implementation of the Long Term Support (LTS) for Ubuntu server and desktop with OpenNebula
			One of the emails for AnonyAI2 in which he discussed a technical issue of OpenNebula release 3.8 and this release functionality with a file system called “GlusterFS”, AnonyAI2 highlighted that there exist a crucial documentation in a certain forum and refer to this forum. In this forum they discuss the implementation of this particular file system. AnonyAI2 said “This is already being done or already done in newer versions of libvirt, see KVM forum 2012 for latest on glusterfs and libvirt”
29	Support ideas	This category represents actions where participants	AnonyAI2 shared his opinion about a suggestion provided by another participant related to the implementation of the Long Term Support (LTS) for

		support other participants on their ideas.	Ubuntu server and desktop with OpenNebula.
			AnonyAI2 supports the idea suggested by another participant related to the addition of an extra column in the list related to virtual machines. This column will add the ID for the virtual machines used in the project. AnonyAI2 supported this idea because he stated that he wasted so much time searching for the IDs used for his virtual machines.
			Different suggestions had been provided to improve contextualization in OpenNebula project. And AnonyAI4 voted for one of them.
			AnonyAI2 supports the extension for the range of Internet Addresses (IPs) used in the project as this will be useful for the computer network used in the project
30	Share technical steps for added features	This category represents actions where participants explain the steps to implement a certain feature with the project.	AnonyAE4 replied to an email that requested some details about the use of "oneadmin" command in OpenNebula. . AnonyAE4 extensively explain the use of this command, he explained the needed file/folder/directory. He also explained the group that he included in this command. In addition, he explained the procedure he followed to add access permissions through this command. He lastly provides needed documentation that will help in a better understanding as well.
31	Debug and Predict Solutions	This category represents actions where participants try to detect sources of software errors and provide solutions to these errors	AnonyAE4 replied to an email that requested some details about the use of "oneadmin" command in OpenNebula. . AnonyAE4 extensively explain the use of this command, he explained the needed file/folder/directory. He also explained the group that he included in this command. In addition, he explained the procedure he followed to add access permissions through this

			command. He lastly provides needed documentation that will help in a better understanding as well.
			AnonyAI2 helped different participants to use the Proper releases of OpenNebula according to different context.
			AnonyAI2 helped other participant who faced a migration problem for the OpenNebula project. AnonyAI2 notice that access permission is the reason behind the migration problem because this permission is different between the source and destination.
32	Highlight missing information in the documentation	This category represents actions where participants show that important information is missing in the documentation of the project.	AnonyAE4 replied to an email that requested some details about the use of "oneadmin" command in OpenNebula. . AnonyAE4 extensively explain the use of this command, he explained the needed file/folder/directory. He also explained the group that he included in this command. In addition, he explained the procedure he followed to add access permissions through this command. He lastly provides needed documentation that will help in a better understanding as well.
33	Share technical advice	This category represent actions where participants share an advice with other participants	AnonyAI2 advised one of the participants to use a certain file system with OpenNebula because this file system has a huge storage bytes that can serve the business needs.
			one participant asked a question about the best "file system" to be implemented with OpenNebula. And AnonyAE4 replied that file systems can be evaluated based on different metrics such as file extensions; hardware used lines of code, or rich documentation. AnonyAE4 then define these

			metrics and provide his advice.
34	Go against ideas	This category represents actions where participants oppose other participants on their ideas.	AnonyAE5 did not support the use of enterprise server edition for SUSE operating system because it has not included the latest version for bug fixing.
35	Explain disadvantages	This category represents actions where participants describe the advantages of using an information system or technical idea.	AnonyAE5 did not support the use of enterprise server edition for SUSE operating system because it has not included the latest version for bug fixing
36	Share fine information (best practice and performance)	This category represents actions where participants share their point of view regarding performance and best practice of the project.	One participant asked a question about the best “file system” to be implemented with OpenNebula. And AnonyAE4 replied that file systems can be evaluated based on different metrics such as file extensions; hardware used lines of code, or rich documentation. AnonyAE4 then define these metrics and provide his advice.
37	Forge some alternatives	This category represents actions where participants neglect some of the possible alternatives for information systems suggested in the mailing list. Participants explain their justification for forging some alternatives.	AnonyAI2 forge the use of heavy weight virtual machine for hosting OpenNebula server because it can break the whole computer cloud for OpenNebula.
38	Show different possible perspectives	This category represents actions where participants provide different alternative for systems in order to use with OpenNebula.	Regarding the use of “InfiniBand” technology with OpenNebula in order to achieve high performance computing, AnonyAI2 suggests several technologies to be used such as Linux iSER, TGTD, and Mellanox VSA.
39	Use the mailing list as a documentation space	This category represents actions where participants use the mailing list to share more documentation needed in order to properly implement the project.	One of the emails for AnonyAI2 in which he discussed a technical issue of OpenNebula release 3.8 and this release functionality with a file system called “GlusterFS”, AnonyAI2 highlighted that there exist a crucial documentation in a certain forum and refer to this forum. In this forum they

			discuss the implementation of this particular file system. AnonyAI2 said “This is already being done or already done in newer versions of libvirt, see KVM forum 2012 for latest on glusterfs and libvirt”.
40	Open new requests in the development page	This category represents actions where participants use discussions in the mailing list in order to add a new request in the development page.	Discussions between participants and AnonyAI2 about the use of “InfiniBand” with OpenNebula cause the addition of InfiniBand source code and integrate it with OpenNebula. This new source code was added by AnonyAI2 upon the request from sponsors
41	Announce system down	This category represents actions where participants announce that the project will be down due to maintenance purposes.	AnonyAO1 announced that the team will migrate the project management portal into Redmine. Therefore, the website will be down for maintenance purposes.
42	Encourage others’ contributions to add features in the project	This category represents actions where participants encourage other participants to reflect their needs by adding new requests in the development page.	AnonyAO4 thanked different participants for their work on new features for the project. Examples of these features were the development of Debian package and the development of scripts that support LVM exporting.
43	Link to requests already created	This category represents actions where participants refer discussions to requests that are already added previously in the development page.	AnonyAE4 answered a request initiated by a participant in the mailing list by referring to requests in the development page.
44	Alert for potential problems	This category represents actions where participants alert for potential problems that may resulted from existing implementations of the project.	AnonyAI2 explained that using having different permissions for the source and destination files will cause many management problems for the project.
			AnonyAI1 explained that having mixed CPU numbers will cause unfair distribution of power between different machines.
45	Ask for explanation	This category represents actions where participants share	AnonyAI2 asked a participant about using clustered LVM instead of normal

	regarding others code revealing	the source code that they developed in the mailing list. And other participants asked them to explain the logical meaning of the source code.	LVM for the project
46	Ask for source code from other participants	This category represents actions where participants request other participants to share the source code that they had already developed.	AnonyAE4 asked another participant to share his experience with a sunstone failure and to share the patch to solve this software bug.
47	Show own effort	This category represents actions where participants share their efforts in developing source code and thinking about solutions to problems in the mailing list.	AnonyAE4 shared his company efforts to use GlusterFS in their distributed file system. And explained how the OpenNebula project can benefit from their work. AnonyAE5 test the implementation of OpenSUSE 11.1 with OpenNebula. He noticed that the problem with the implementation was due to “gem” file. AnonyAE5 worked on solving the problem and shared the solution in the mailing list
48	Show that the problem is shared	This category represents actions where participants highlighted that the problem shown in the mailing list is the same as the problem they have personally had while implementing the project.	AnonyAE5 shared the same problem with other participants. He explained his network setting that might lead to such a problem.
49	View log file	This category represents actions where participants attach the “log file” with their emails.	Different participants uploaded the log file for their network in order to get as much help as they can. Examples of these participants were: AnonyAC3 and AnonyAI4.
50	Ask for technical	This category represents actions where participants	AnonyAG2 was looking to understand the code for changing default settings of a virtual machine and the actual implementation behind “onevm cancel”

	details	request more details about a topic sent in the mailing list.	command.
51	Comments on technical ideas	This category represents actions where participants comment on ideas suggested by other participants in the mailing list	AnonyAC4 responded to one of the emails, sent about “federation” in OpenNebula, which he will connect another cluster of nodes with VPN to save the public internet addresses given to physical nodes in the network.
52	View different alternatives for system integration of same problem	This category represents actions where participants suggest different systems to be integrated with OpenNebula.	Regarding the use of “InfiniBand” technology with OpenNebula in order to achieve high performance computing, AnonyAI2 suggests several technologies to be used such as Linux iSER, TGTD, and Mellanox VSA.
53	Request Features	This category represents actions where participants request the addition of new feature to the project.	AnonyAG1 requested new features such as a dynamic firewall and a procedure to detect failure in virtual machines.
54	Ask for clarifications about others ideas	This category represents actions where participants request more details about ideas suggested by other participants in the mailing list.	One participant suggested the use of marketplace with OpenNebula project. Accordingly, AnonyAE3 asked the participants to explain the meaning of marketplace and why shall it be implemented with OpenNebula.
55	Suggest customization in the project	This category represents actions where participants require customizations to be done to the project in order to fir their special needs.	AnonyAO2 suggested the addition of “contextualization” for virtual machine as a way to customize the access for virtual machines through “ssh” authentication.
56	Suggest best practice	This category represents actions where participants suggest the best implementation of the project	AnonyAI2 explained that crashes on the system were due to faulty hardware. He suggested the use of “memtest86+” in order to test hardware before using a high input and output loads.
			AnonyAI2 helped different participants to use the Proper releases of OpenNebula according to different context.

57	Suggest other system deployment	This category represents actions where participants suggest the implementation of a certain information system with OpenNebula in certain cases.	AnonyAE5 suggested the use of a specific interface in order to fit his business needs.
			AnonyAI4 suggested the deployment of GlusterFS file system with Unix machines in the project.
58	Suggest improvements of new systems integration	This category represents actions where participants suggest improving the project by integrating the project with other specialized information systems.	AnonyAE2 suggested Bright computing option for OpenNebula
			AnonyAG1 suggested automation of putting a virtual machine operating system images in the images repository instead of the manual process already existed.
59	Suggest new features to overcome bottlenecks	This category represents actions where participants the addition of certain features into the project in order to overcome bottlenecks in the project	AnonyAC4 shared with other participants different bottlenecks that the project was facing. These bottlenecks were: extending tm driver for ssh, OpenNebula client packages only, and automation of some processes such as creations of registered users and related keys for them.
60	Suggest system to support old suggested ideas	This category represents actions where participants suggest certain information systems in order to support ideas that had been suggested previously.	AnonyAG1 suggested automation of putting a virtual machine operating system images in the images repository instead of the manual process already existed.
			AnonyAI2 suggested the integration of several systems with the project. examples of these systems were OpenVSwitch, MooseFS, Cloudera Manager…etc.
61	Correct malfunctions in the project	This category represents actions where participants fix failures in the project.	AnonyAO3 updated the deployment files in the transfer drivers of the project because these files were not copying the log files properly into the project.

62	Debug problems faced by participants	This category represents actions where participants identify sources of errors in the project.	The OpenNebula team members (AnonyAO1, AnonyAO2, AnonyAO3, AnonyAO4, and AnonyAO5) debug problems reported in the mailing list and helped others to solve these problems.
63	Report software bug	This category represents actions where participants declare the existence of software bug in the source code of the project. They declare software bugs in the mailing list as well as the development page of the project.	AnonyAG3 reported a bug in non-interactive bash sessions.
64	Improve lines of code	This category represents actions where participants work on devoting the source code of the project in order to ensure that the source code is properly functioning.	AnonyAE4 corrected quota patch OpenNebula source code.
65	Highlight a problem or software bug	This category represents actions where participants identify the existence of a software bug in the project. They identify software bugs in the mailing list.	AnonyAE5 test the implementation of OpenSUSE 11.1 with OpenNebula. He noticed that the problem with the implementation was due to “gem” file. AnonyAE5 worked on solving the problem and shared the solution in the mailing list.
66	Test others' work to solve same faced problem	This category represents actions where participants help other participants to solve their implementation problems by testing the source code used in the development.	AnonyAE5 test the implementation of OpenSUSE 11.1 with OpenNebula. He noticed that the problem with the implementation was due to “gem” file. AnonyAE5 worked on solving the problem and shared the solution in the mailing list
67	Exchange benefits	This category represents actions where participants agree to mutually help and benefit from the efforts that they share regarding a specific implementation of the project.	One participant shared a problem he faced when deploying virtual machine with OpenNebula. AnonyAI1 explained that he was planning to use this type of virtual machine in his project. Thus, he explained that he would like to help in solving this problem in order to use the whole setting later on his own

			network.
68	Offer development help	This category represents actions where participants propose to develop a source code and add it to the project.	AnonyAI1 said that he would be happy to contribute to the development of the ESXi 4.0 server with the OpenNebula project.
69	Ask others to help in the development	This category represents actions where participants request development help from other participants in the mailing list.	AnonyAO1 asked a participant to develop a source code to implement QoS policies in the project.
70	Ask others to test the project	This category represents actions where participants request other participants in the mailing list to test the source code they are using in the project.	AnonyAO1 asked a participant to test a software patch that AnonyAO1 shared in order to ensure that the patch was working properly.
71	Assign tasks to participants	This category represents actions where participants ask other participants to solve or to develop a new source code for the project.	The OpenNebula team assign task for different participant to solve software bugs that were reported in the development page.
72	Improve others' work to solve same faced problem	This category represents actions where participants modified the source code used by other participants in order to be properly used in the project.	AnonyAC1 suggested that the use of “SHA1” authentication with ElasticFox would provide more security for the project.
73	Review different perspectives	This category represents actions where participants review the different alternative suggested in the mailing list and share their opinion about these alternatives.	AnonyAE3 reviewed different issues related to the implementation of market place in OpenNebula.
74	Suggest testing new system integration	This category represents actions where participants request others to test the source code thay have	The OpenNebula team asked participants to test new systems added into the project. AnonyAI1 participated by testing the metadata server integrated

	efforts	developed in order to have different opinions about the same source code implementations.	with OpenNebula.
75	Reflect on the work of others in the project	This category represents actions where participants share their technical opinion about others ideas and development efforts.	AnonyAI2 shared his opinion about a suggestion provided by another participant related to the implementation of the Long Term Support (LTS) for Ubuntu server and desktop with OpenNebula.

Using the previous table, similar categories are grouped together in order to identify codes that represents contributions performed by participants in OpenNebula. Figures A.1 to A.4 exhibit those different contributions: ‘Modify Available Documentation’ code, ‘Determine a Software Roadmap’ code, ‘Report Software Bugs/ Features’ code, ‘Solve a Software Bug’ code, and ‘Develop a Software Feature’ code, respectively.

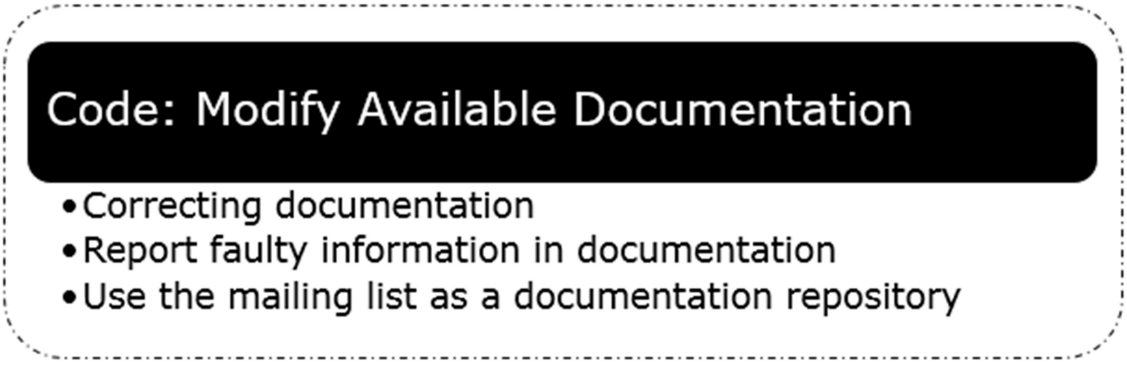

Figure A. 1 'Modify Available Documentation' Code

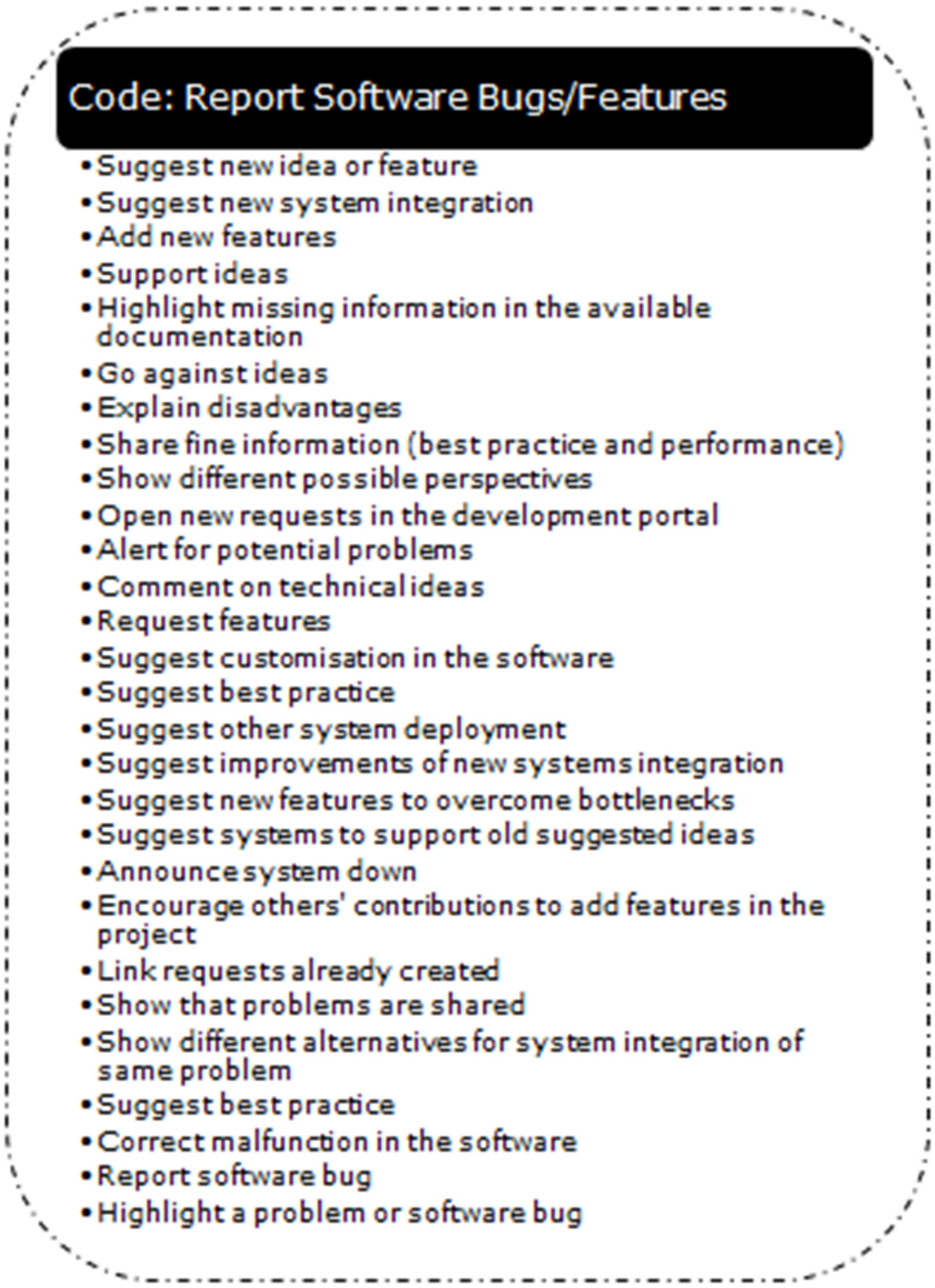

Figure A. 2 Report Software Bugs/ Features

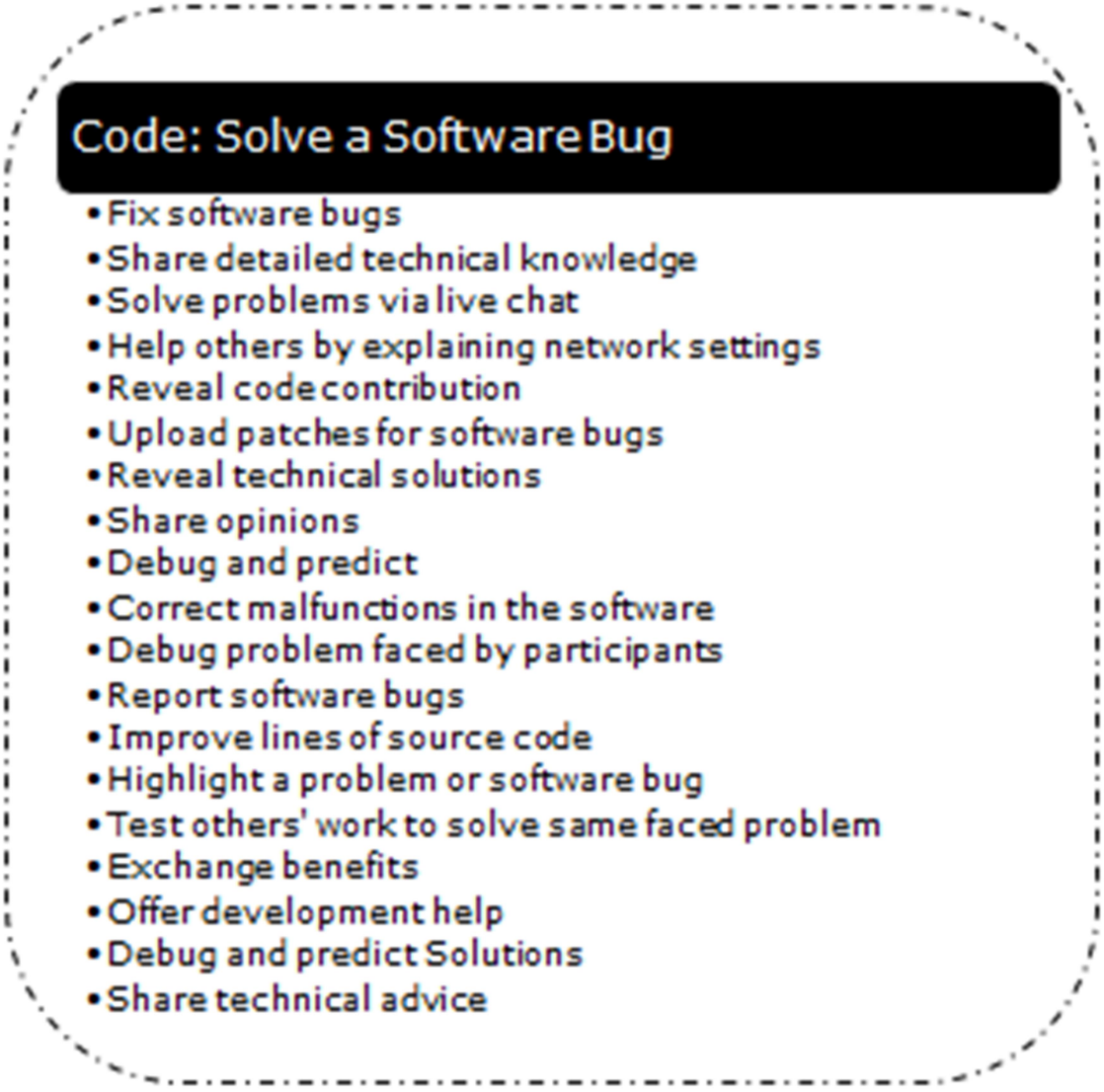

Figure A. 3 Solve Software Bugs

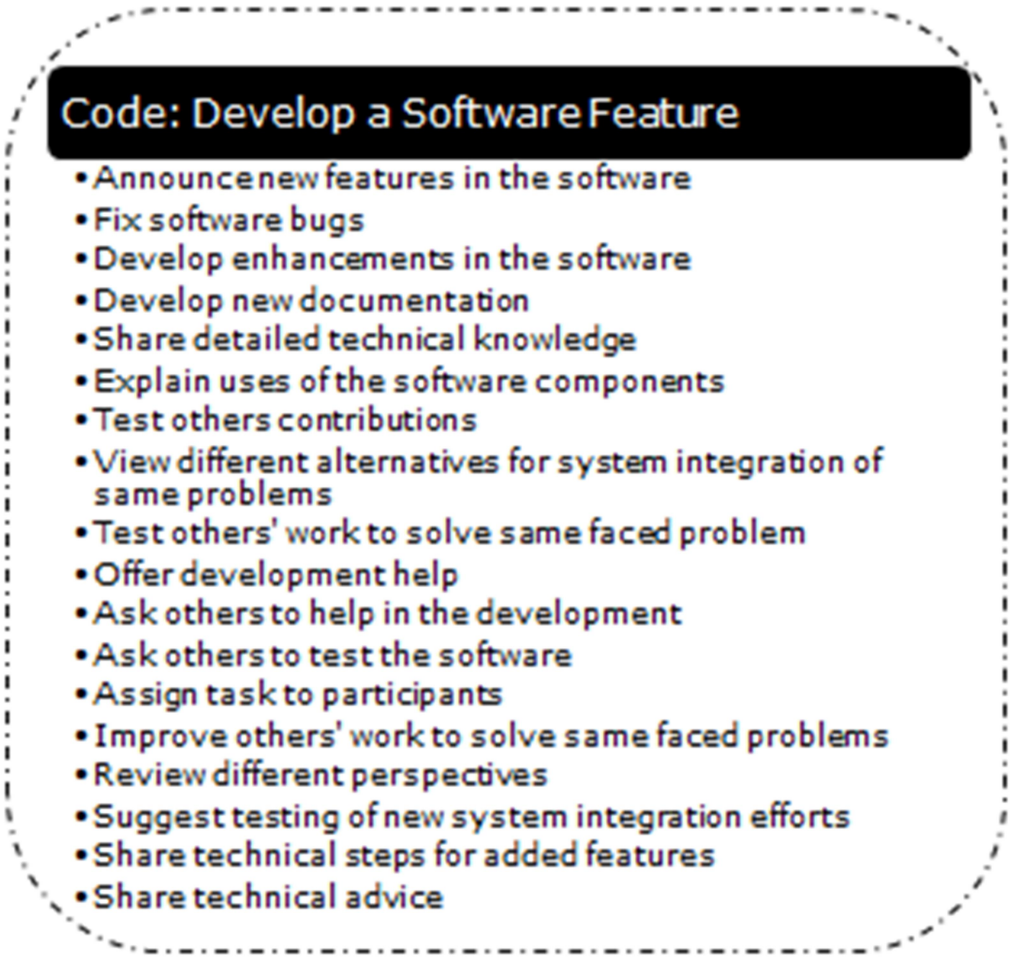

Figure A. 4 Develop Software Features

Appendix B: The Analysis of Software Complementarities

From the analysis for categories of the different requests in the development page and the documentation available for OpenNebula, I was able to:

First, describe the 21 categories that have been extracted from the development page.

Second, identify the main technical components that make up OpenNebula. And link these components with categories.

B.1 Categories identified in the development page

There are 21 identified categories in the development page of OpenNebula. The analysis of the different requests in the development page as well as the analysis of the available documentation helped me in technically understand the differences between these categories. A technical description of these categories as revealed from the analysis is listed in table B.1.

Table B. 1 Technical description of the categories as identified from the analysis of the development page

Categories	Description
Core & Systems	• This category represents a source code of the project that manages virtualization in OpenNebula project. Examples for the management of virtualization are handling failure in accessing virtual machines, deleting virtual machines, initiating states of virtual machines, building databases for virtual machines and managing these database, Creating templates for different virtual machines , Multi-user support to improve authentication and authorization, Migrating between the different drivers , Physical and Internet addresses for the different machines …etc. • This category has a source code that develops the database for the virtualization management function for the project (e.g., OpenNebulaSystems, 2015k). • This category composes of the essential components that **must** to be installed in order for users to start working the OpenNebula project.
Scheduler	• This category represents a source code of the project that is used for scheduling function; scheduling function is needed to schedule the use of different drivers for the project. • The scheduler is **optionally** installed by users who are willing to manage the allocation of pending virtual machines to the proper hosts (For details see OpenNebulaSystems, 2015p). Users may use other scheduling applications other than the scheduler defined for OpenNebula.
Drivers- Auth	• This category represents a source code of the project that allows a computer to communicate with authentication services for a computer network (For details see OpenNebulaSystems, 2015c). • This driver is **optionally** used based on users' needs. An example of this driver is the x509 authentication driver.
CLI	• This category represents a source code of the project that allows users to communicate and manage virtual machines through computer commands. Users can manage virtual machines, define access lists, and cluster virtual machines through this application (e.g., OpenNebulaSystems, 2013).

	• CLI is **optionally** installed by users.
Client API & Library	• This category represents a source code that allows different applications to make use of the source code of virtualization management (first category identified in this table) of the project. A collection of APIs and Libraries were needed to integrate applications with the project. examples are plenty for APIs used in OpenNebula (e.g., OpenNebulaSystems, 2015v, OpenNebulaSystems, 2015o, OpenNebulaSystems, 2015e, OpenNebulaSystems, 2015i) • APIs are **optionally** installed and used based on users' needs.
Documentation	• This category represents the non-technical component in the project. And it is **already existed** in the project website (OpenNebulaSystems, 2016b).
Sunstone	• This category represents a source code of the project that is used to manage the whole project through a Graphical User Interface (GUI) (For details see OpenNebulaSystems, 2015m). • This is an essential part of the project that **must** to be installed along with the source code of both the virtualization management and the database.
Cloud View	• This category represents an option installed with the source code for managing virtualization in the project. this option provides customers the ability for creating their own environment through the internet instead of installing the source code of the project on users' computers (For details see OpenNebulaSystems, 2015q). • This option is installed **by default** with the core source code of the project (first row in this table).
Context	• This category represents a source code of the project that provides contextualization function for virtual machines. Three types for contextualization are available in OpenNebula; basic, advanced, and Windows (See examples OpenNebulaSystems, 2015d, OpenNebulaSystems, 2015a, OpenNebulaSystems, 2015u). • This category is **optionally** installed and used based on users' needs.
Drivers- Monitoring	• This category represents a source code of the project that allows the collection of monitoring data for different virtual machines used in the project (For details see OpenNebulaSystems, 2015f). • This driver is **optionally** used based on users' needs.
Drivers- Network	• This category represents a source code of the project that allows the configuration of properties of a network for the different virtual machines used in the project (For details see OpenNebulaSystems, 2015g).

	• This driver is **optionally** used based on users' needs.
Drivers- Storage	• This category represents a source code of the project that allows management and storage of images for the different virtual machines used in the project (For details see OpenNebulaSystems, 2015r). • This driver is **optionally** used based on users' needs.
Drivers- VM	• This category represents a source code of the project that allows the virtualization of the different computer components used by users (For details see OpenNebulaSystems, 2015s). Examples of these drivers are XEN, KVM, Amazon EC2, and VMware. • This driver is **optionally** used based on users' needs.
EC2 Server	• This category represents a source code of the project that allows the access for the source code of the project even without installing the source code on users' computers (For details see OpenNebulaSystems, 2015b). • This category is **optionally** used and installed based on users' needs.
MarketPlace	• This category represents a source code of the project that distribute and deploy several applications and appliances that are ready-to-run with OpenNebula (For details see OpenNebulaSystems, 2016a). • This category is **optionally** used and installed with OpenNebula based on users' needs.
OneFlow	• This category represents a source code of the project that allow the management of multi-tiered applications in the project (interconnected computer network) (For details see OpenNebulaSystems, 2015h). • This category is **optionally** installed and used based on users' needs.
OneGate	• This category represents a source code of the project that manages the ability of virtual machines of pushing monitoring data to OpenNebula (For details see OpenNebulaSystems, 2015j). • This category is **optionally** installed and used based on users' needs.
oZones	• This category represents a source code of the project that allows the centralized management for distributed components of the computer network among different network zones (For details see OpenNebulaSystems, 2015n). • This category is **optionally** installed and used based on users' needs.
Packaging	• This category represents a source code of the project that integrate the different components with core OpenNebula (e.g., OpenNebulaSystems, 2012).

	• These components are **optionally** installed and used; these components vary based on users' needs and OpenNebula different releases.
Testing & Infrastructure	• This category represents a source code for the quality assurance of the several components making up the whole project (For details see OpenNebulaSystems, 2015l). • This category varies based on **users' needs** and requirements.
vCenter	• This category represents a source code of the project that allows the use of advanced features in the project such as vMotion and DRS scheduling (For details see OpenNebulaSystems, 2015t). • This source code is **optionally** installed and used based on users' needs.

B.2 The technical components in OpenNebula

The descriptions identified in table 8 helped in identifying the components of the project. The analysis so far revealed that: (1) some categories are essential for the project while other categories are optional, and (2) these categories can be grouped into six main components for the project: core project, drivers, servers, platforms, applications, and Application Program Interfaces (APIs). These categories will be discussed in details shortly.

- Core OpenNebula. This is the main component of the project that performs the cloud management function. Cloud management aims for managing the different virtual resources used by users/firms. This component must be installed in order to start working in the project. It consists of four categories: “core & systems”, “documentation”, “sunstone”, and “cloud view”.
- Drivers are “software that allows a computer to communicate with a device or service” (Microsoft). These drivers represent computer resources that are virtualized by users. Drivers are optionally installed by users. Drivers consist of five categories: “Drivers-Auth”, “Drivers- monitoring”, “Drivers- Network”, “Drivers- Storage”, and “Drivers- VM”.
- Servers are “computers that provide access to various services available on the network” (Turban et al., 2007p. 162). Servers are optionally installed by users. Servers are represented by “EC2 Server” category.
- Operating systems are “a set of computer programs that controls the computer hardware and acts as an interface with application programs” (Stair and Reynolds, 2013 P.625). Operating systems consist of two categories: “packaging” and “testing and infrastructure”.

- Applications are “a piece of software designed to carry out a standard business function” (Curtis and Cobham, 2008 P.95). Applications consist of 11 categories: “scheduler”, “CLI”, “Client API & Library”, “Context”, “MarketPlace”, “OneFlow”, “OneGate”, “oZones”, “packaging”, “testing & infrastructure”, and “vCenter”.
- Application Program Interfaces (APIs) are “interface that allows applications to make use of the operating system” (Stair and Reynolds, 2013 P.617). The Category that represent “API” component is: Client API & Library.

To start working with the project, one need install the source code for the project or access the project through the internet. If a user chooses to install the source code of the project, he/she needs to install the source code on the operating system for his/her computer. These operating systems have suitable computing capabilities to run computer commands and lines of code. Examples of operating systems that can be integrated with OpenNebula project are:

- Ubuntu. For example, AnonyAC1 was interested in implementing OpenNebula with Ubuntu and LDAP authentication method. He added a request in the development page to solve a software bug about upgrading Ubuntu. Another example, AnonyAC2 shared technical source code for a new “Apache passenger package” for users who run sunstone with Apache passenger on Ubuntu operating systems.
- Debian. For example, AnonyAI5 participated in the correction of the documentation related to Debian Squeeze operating system in OpenNebula project.
- OpenSUSE. For example, AnonyAG5 suggested and participated in developing a source code to integrate OpenSUSE project with OpenNebula.
- CentOS. For example, AnonyAE1 participated in the development of a source code to integrate CentOS operating system with OpenNebula 3.9.9 and ESXi virtual machines.

On the other hand, if a user chooses to access the source code of the project through the internet, without installing the source code, he/she can access the project through the project web servers. Web servers for OpenNebula are:

- EC2 Server. For example, AnonyAG2 was using X509 security for authentication via commands of EC2 server in order to connect to OpenNebula.
- OCCI Server. For example, AnonyAE4 configured his virtual local machines using OCCI server and related interfaces.

After accessing the project, a user uses the main application for the project (labelled as Core OpenNebula). The main application of the project consists of: Sunstone application, virtualization management application, and a database.

- Sunstone is an application used to manage the different components used in the project. It is a Graphical User Interface (GUI). GUI is "the part of the operating system users interact with that uses graphic icons and the computer mouse to issue commands and make selections" (Kenneth et al., 2001 P.199). For example, AnonyAG5 participated in solving technical problems related to managing virtual machines through Sunstone application.
- Virtualization management application is a source code used to manage virtualization in the project. For example, AnonyAO3 developed a source code that modifies "time out" when the template of virtual machines is updated. Another example, AnonyAO4 participated in the development of a source code that migrate KVM, IM, and EC2 drivers into the new driver engine in the project.
- A database is a storage used to logically store and retrieve information such as MySQL. For example, AnonyAI1 participated in solving several problems for MySQL malfunctions with OpenNebula.

Accordingly, users will use Core OpenNebula either through operating systems or web servers. In both cases," Core OpenNebula" and "operating systems or web servers" require Application Program Interfaces (APIs) in order to be able to communicate. For example, AnonyAI1 solved several software bugs related to accessing the project through EC2 server and EC2 API.

Each user can access and use the core application of the project. At that stage, users will have two possible scenarios to use the project:

Scenario 1: a user wants to use core OpenNebula only to manage virtual computer network that is already created.

Scenario 2: a user wants to use core OpenNebula to manage his virtual computer network. Moreover, he wants to create more virtual computers into his network (virtualization drivers), and/or use additional management options that are not basically provided by core OpenNebula (management applications). This scenario shows that OpenNebula project consists of core OpenNebula as well as two optional services. These services are virtualization drivers and management applications.

Five virtualization drivers are provided by the project (see table B.2); virtualization driver, storage driver, monitoring driver, network driver, and authentication driver. Each driver has its own services.

Table B. 2 Types of Drivers in OpenNebula

Driver Type	Description	Examples
Virtualization Driver	These drivers are software that virtually imitates a particular computing system (Stair and Reynolds, 2013)	XEN driver. For example, AnonyAI3 was concerned about participating and understanding of networking and storage issues related to XEN virtual machines, understating commands, implementation, and documentation as well.
		KVM driver. For example, AnonyAG3 was reporting problems related to the implementation of "OpenNebula express" with KVM environment under an in-house developed operating system.
		OpenVZ driver. For example, AnonyAI3 developed and added OpenVZ source code and documentation to OpenNebula repository.
		VMware Driver. For example, AnonyAC3 solve different technical problem related to the implementation of OpenNebula 4.0 with VMware virtual machines only.
		ESXi Driver. For example, AnonyAI1 provided fixed to several bugs related to networking problem with ESXi virtual machine and adding revisions to code repository of ESXi virtual machine.
Storage drivers	These drivers are software that create computer images for the virtual machines used in	Data store driver. For example, AnonyAO1 announced the addition of a new data store driver for iSCSI. Another example, AnonyAI5 reported a software bug in Ceph data store driver.

	the project and manage these images.	Transfer Manager. For example, AnonyAC5 suggested the addition of LVM2 transfer manager driver in OpenNebula 3.4.x
Monitoring drivers	These drivers are software that is responsible of preparing and communicating monitoring information about the different computer devices used in the project.	An example of monitoring driver used in the project is the IM driver. For example, AnonyAI1 deployed and solved software bugs related to IM storage driver implemented with his OpenNebula environment; Ubuntu KVM, ESXi 3.5, ESXi 4.0, scheduler, EC2 API, and Libvirt.
Network drivers	These drivers are software responsible for managing the network of the different devices that are used in the project.	An example of this driver is VNM driver. For example, AnonyAI5 was interested in solving technical problems related to the use of contextualization template in network driver
Authentication driver	These drivers are software that communicates with authentication services in a computer network	"X509". For example, AnonyAG2 used X509 security for authentication via commands of EC2 server in order to connect to OpenNebula.
		"SHA1". For example, AnonyAI1 contributed to SHA1 hash and plain password implementation in OpenNebula
		"LDAP". For example, AnonyAE3 corrected documentation of OpenNebula LDAP authentication

In additions to the drivers provided by the project, the project also provided several management applications. Examples found for these applications are the scheduler application, and Libvirt application. The scheduler is an application that assigns pending virtual machines to needed computer devices. Libvirt is an application for remotely managing authentication for virtual machines.

If a user uses any of the additional services provided by the project, he/she require the use of related APIs in order to integrate these services with the project.

Based on this, categories found in the development page of OpenNebula (see table 2) can be classified into core categories and complements categories (see figure 1).

A Core category is an essential source code in the software that **must** to be installed in order for users to start working with the OpenNebula. This source code is crucial for using the project as a cloud computing technology.

A complement category is a source code that is **optionally** installed by users based on their needs for their computer network.

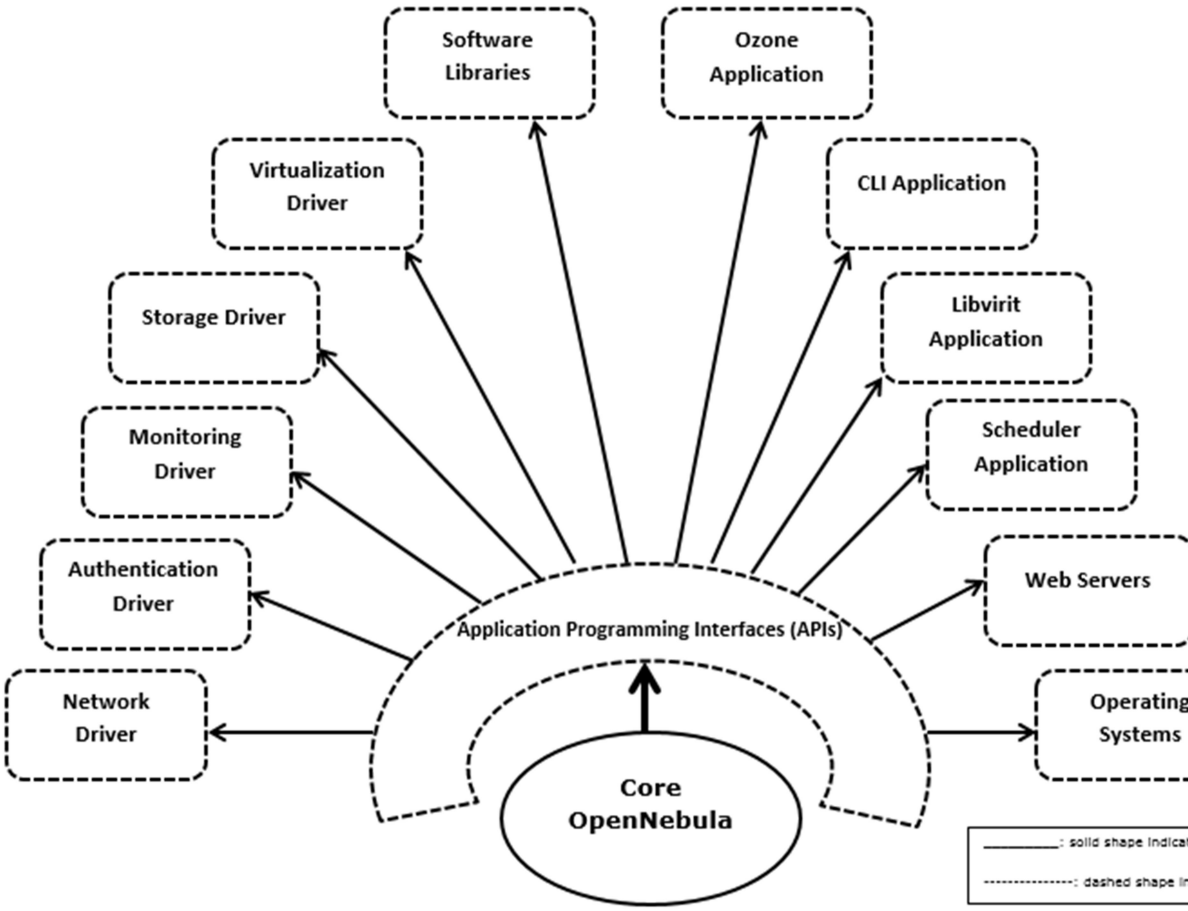

Figure B. 1 Classification of the software components according to core and complement