JYU DISSERTATIONS 857

Katja Henttonen

Sustaining the Bazaar

Explorations into Governance and Sustainability in Open-Source-Producing Organisations, Communities, and Ecosystems

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY

JYU DISSERTATIONS 857

Katja Henttonen

Sustaining the Bazaar

Explorations into Governance and Sustainability in Open-Source-Producing Organisations, Communities, and Ecosystems

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi Liikunta-rakennuksen luentosalissa L302
joulukuun 4. päivänä 2024 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in building Liikunta, auditorium L302, on December 4, 2024, at 12 o'clock noon.

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2024

Editors
Marja-Leena Rantalainen
Faculty of Information Technology, University of Jyväskylä
Timo Hautala
Open Science Centre, University of Jyväskylä

ISBN 978-952-86-0413-6 (PDF)
URN:ISBN:978-952-86-0413-6
ISSN 2489-9003

Permanent link to this publication: http://urn.fi/URN:ISBN:978-952-86-0413-6

ABSTRACT

Henttonen, Katja
Sustaining the Bazaar: Explorations into Governance and Sustainability in Open-Source-Producing Organisations, Communities, and Ecosystems
Jyväskylä: University of Jyväskylä, 2024, 107 p.
(JYU Dissertations
ISSN 2489-9003; 857)
ISBN 978-952-86-0413-6

Sustainability has long been a critical, though sometimes latent, theme in FOSS research. Today, FOSS underpins critical systems worldwide, making its sustainability essential for the security and reliability of the technologies our societies rely on. Governance, from day-to-day work coordination to long-term strategic planning, is essential to the sustainability of FOSS. This dissertation explores FOSS governance and sustainability across the community, corporate, and ecosystem levels. Adopting a multi-level approach bridges a gap in the literature which examines these levels separately. This research is grounded in case studies conducted within FOSS-producing communities, organisations, and ecosystems. The case studies triangulate various data sources, such as interviews, observations, and documentation, and primarily employ qualitative analysis methods. The findings enrich our understanding of how community, corporate, and ecosystem governance underpins various forms of sustainability, including infrastructural, resource-based, and interactional aspects of it. Additionally, they shed light on the dynamics between these levels, revealing both synergies and trade-offs. In a synergistic scenario, different governance levels collectively contribute to a positive feedback loop where high-quality products attract more users; users become contributors, and their contributions further enhance product quality. However, the levels also compete for resources, highlighting inherent tensions that must be managed. The findings also illuminate often-overlooked aspects of FOSS governance, such as the differences between public and private sector FOSS, ideological tensions among FOSS-engaged companies, and the role of promotional activities in sustaining FOSS. Furthermore, each case study also has a practical goal of equipping practitioners with models and tools they need to navigate the complexities of FOSS governance in the case context, ultimately enhancing the sustainability of FOSS. The dissertation concludes by suggesting several future research directions, including the expansion of FOSS sustainability to incorporate environmental and social dimensions.

Keywords: free software, FOSS, open source, public-private collaboration, software ecosystem, software governance, sustainability

TIIVISTELMÄ (ABSTRACT IN FINNISH)

Henttonen, Katja
Basaarin säilyttäminen: Tutkimuksia hallinnosta ja kestävyydestä avoimen lähdekoodin organisaatioissa, yhteisöissä ja ekosysteemeissä
Jyväskylä: Jyväskylän yliopisto, 2024, 107 s.
(JYU Dissertations
ISSN 2489-9003; 857)
ISBN 978-952-86-0413-6

Kestävyys on jo pitkään ollut tärkeä aihe vapaiden ja avoimen lähdekoodin ohjelmistojen (VALO) tutkimuksessa. Nykyään VALO toimii maailmanlaajuisesti monien kriittisten järjestelmien perustana, joten sen kestävyys on olennaista yhteiskuntien teknologisen turvallisuuden varmistamiseksi. Hallinnolliset tehtävät, aina päivittäisestä työn koordinoinnista pitkän aikavälin strategiseen suunnitteluun, ovat keskeisiä VALOn kestävyyden kannalta. Tämä väitöskirja tutkii VALOn hallintoa ja kestävyyttä yhteisön, yritysten ja ekosysteemin tasolla. Monitasoinen lähestymistapa paikkaa kirjallisuuden katvealuetta, koska aiempi tutkimus on enimmäkseen tarkastellut näitä hallintotasoja erillään. Työ perustuu avoimen lähdekoodin ohjelmistoja tuottavissa yhteisöissä, organisaatioissa ja ekosysteemeissä tehtyihin tapaustutkimuksiin. Tapaustutkimuksissa hyödynnetään erilaisia tietolähteitä, kuten haastatteluja, havainnointia ja dokumentteja sekä käytetään pääasiassa laadullisia analyysimenetelmiä. Tulokset syventävät ymmärrystä siitä, miten eri hallintotasot muovaavat VALOn kestävyyttä ja vuorovaikuttavat keskenään. Pureutuminen eri hallintotasojen väliseen dynamiikkaan paljastaa sekä synergioita että kompromisseja. Synergisessä tilanteessa eri hallintotasot luovat yhdessä positiivisen palautesilmukan, jossa laadukkaat ohjelmistotuotteet houkuttelevat käyttäjiä ja käyttäjistä tulee aktiivisia osallistujia, joiden panokset edelleen parantavat tuotteen laatua. Toisaalta hallinnon eri tasot kilpailevat myös resursseista, mikä synnyttää toimenpiteitä vaativia jännitteitä. Lisäksi tutkimustulokset valottavat myös useita aiemmin vähän käsiteltyjä VALO-hallinnon osa-alueita, esimerkiksi julkisen ja yksityisen sektorin eroja, yritysten välisiä ideologisia ristiriitoja sekä markkinointityön vaikutusta kestävyyteen. Lisäksi jokainen tapaustutkimus tähtää myös käytännön tavoitteeseen ja tarjoaa toimijoille malleja tai työkaluja VALO-johtamiseen kyseisessä ympäristössä, edistäen siten VALOn kestävyyttä. Väitöskirjassa ehdotetaan lopuksi useita tulevaisuuden tutkimussuuntia, mukaan lukien VALOn kestävyyden määritelmän laajentaminen ympäristöllisiin ja sosiaalisiin näkökulmiin.

Avainsanat: Avoin lähdekoodi, julkis-yksityinen yhteistyö, kestävyys, ohjelmistoekosysteemi, ohjelmistojohtaminen, VALO, vapaa ohjelmisto

Author Katja Henttonen
Faculty of Information Technology
University of Jyväskylä
Finland
katja.m.henttonen@student.jyu.fi
ORCHID: 0000-0001-6912-128X

Supervisors Pasi Tyrväinen
Faculty of Information Technology
University of Jyväskylä
Finland

Mirja Pulkkinen
Faculty of Information Technology
University of Jyväskylä
Finland

Reviewers Slinger Jansen
Department of Information and Computer Science
Utrecht University
The Netherlands

Marko Seppänen
Faculty of Management and Business
Tampere University
Finland

Opponent Björn Lundell
School of Informatics
University of Skövde
Sweden

PREFACE

The journey that culminated in this dissertation has its roots in my early fascination with the world of free and open-source software (FOSS). This began in the late 1990s within the vibrant Finnish demo scene and led me to major European hacker events in the early 2000s, such as Hackers at Large (HAL) in 2001, laying the foundation for my pursuits in the field of FOSS.

Since 2006, I have been fortunate to work professionally with FOSS, soon intertwining this work with academic research. Witnessing the evolution of the FOSS phenomenon over the past two decades has been interesting. The growth and maturation of this field have been profound, and I am grateful for the opportunity to have observed this transformation closely.

I am grateful to many individuals who have helped me along this journey. First and foremost, I would like to thank my supervisors at the University of Jyväskylä, Prof. Pasi Tyrväinen and Dr. Mirja Pulkkinen. Their unwavering support, guidance, and expertise have been instrumental in completing this dissertation. I also extend my heartfelt appreciation to my spouse, Jussi, whose support has been my anchor through the most challenging phases of this process.

I also want to thank all the co-authors of the included articles and all the FOSS communities and organisations that have generously shared their knowledge and experiences. Collaborations with the Decidim Free Software Association and Mahiti Infotech have been particularly insightful and inspiring.

I also warmly remember prior colleagues at VTT from the early part of my journey, including Dr. Pekka Savolainen, Prof. Eila Ovaska, and Dr. Mari Matinlassi, who introduced me to the academic world, and my late friend Nicolas Sahlqvist, who introduced me to hacker culture in the early 2000s.

This dissertation does not reflect my efforts alone but also the collaborative spirit of FOSS communities and the support of people around me.

Katja Henttonen
27.8.2024
Helsinki

FIGURES

FIGURE 1 Main elements of the CO-SLM framework 56
FIGURE 2 Health-sustaining activities in public sector FOSS ecosystems....58
FIGURE 3 Relationships between roles, rules, and tools 58
FIGURE 4 Virtuous growth cycle and FOSS governance levels 71
FIGURE 5 Matrix of FOSS Co-ordination Mechanisms 76

TABLES

TABLE 1 Levels of FOSS governance addressed in each article 14
TABLE 2 Summary of case study designs across articles 45
TABLE 3 Summary of data collection methods...47
TABLE 4 FOSS governance contributions to each sustainability type.........69
TABLE 5 VALO-hallinnon vaikutukset kestävyyden eri tyyppeihin 85

CONTENTS

ABSTRACT
TIIVISTELMÄ (ABSTRACT IN FINNISH)
PREFACE
FIGURES AND TABLES
CONTENTS
LIST OF INCLUDED ARTICLES

1 INTRODUCTION 13
 1.1 Background and Motivation 13
 1.2 Goal and Research Questions 14
 1.3 Terminology 15
  1.3.1 Definitions of Key Terms 15
  1.3.2 Evolution of Terminology 17
 1.4 Structure of this Thesis 18

2 ESSENTIALS OF FREE AND OPEN-SOURCE SOFTWARE 19
 2.1 Competing and Complementary Ideologies 19
 2.2 Types of FOSS Licences 22
 2.3 Evolution of the FOSS Development Model 24

3 FOSS GOVERNANCE WITHIN COMMUNITIES, ECOSYSTEMS, AND ORGANISATIONS 27
 3.1 FOSS Community Governance 27
  3.1.1 Early Research on FOSS Governance 28
  3.1.2 Recognition of Various Governance Models 29
  3.1.3 Motivating and Coordinating Work in FOSS Communities .. 30
 3.2 Organisational FOSS Governance 33
 3.3 FOSS Ecosystem Governance 36
 3.4 Desired Outcomes of Governance 38
 3.5 Summary of Literature Review 41

4 RESEARCH DESIGN AND METHODOLOGY 42
 4.1 Research Approach and Philosophical Stance 42
 4.2 Case Study Designs 44
 4.3 Data Collection Methods 46
 4.4 Data Analysis Methods 48
  4.4.1 Qualitative Content Analysis 48
  4.4.2 Specialised Analytical Methods 49

5 OVERVIEW OF THE INCLUDED ARTICLES 51
 5.1 Article I: Contributing to Eclipse – A Case Study 51

5.2 Article II: Open Source-Based Tools for Sharing and Reuse of Software Architectural Knowledge 52
5.3 Article III: Libre Software as an Innovation Enabler in India – Experiences of a Bangalorean Software SME 53
5.4 Article IV: Managerial Perspective of Open Collaboration and Networked Innovation 54
5.5 Article V: Life-Cycle Management in Government-Driven Open-Source Projects – Practical Framework 55
5.6 Article VI: Health and Orchestration of Public-Sector Open-Source Software Ecosystems: Roles, Rules, and Tools 57

6 RESULTS 59
6.1 FOSS Governance for Infrastructural Sustainability 59
6.1.1 Community Governance and Infrastructural Sustainability .. 59
6.1.2 Ecosystem Governance and Infrastructural Sustainability 60
6.1.3 Corporate Governance and Infrastructural Sustainability 61
6.2 FOSS Governance for Resource-Based Sustainability 61
6.2.1 Governing Input Resources 61
6.2.2 Governing Output Resources 64
6.3 FOSS Governance for Interactional Sustainability 66
6.3.1 Community Governance and Interactional Sustainability 66
6.3.2 Ecosystem Governance and Interactional Sustainability 67
6.3.3 Corporate Governance and Interactional Sustainability 67
6.4 Summary of FOSS Governance and Sustainability 68
6.4.1 Levels of FOSS Governance and Their Inter-dynamics 69
6.4.2 Governance Supporting a Positive Feedback Loop 70
6.4.3 Differences Between Public and Private Sector FOSS Governance 72

7 DISCUSSION 73
7.1 FOSS Governance Levels 73
7.2 FOSS Sustainability Types 77
7.3 Limitations and Methodological Reflections 78

8 CONCLUSION 80
8.1 Contributions to Researchers 80
8.2 Contributions to Practitioners 81
8.3 Future Research Directions 82

YHTEENVETO (SUMMARY IN FINNISH) 84

REFERENCES 87

ORIGINAL PAPERS

LIST OF INCLUDED ARTICLES

I Henttonen, K., & Matinlassi, M. (2007). Contributing to Eclipse: A case study. In W. Bleek, J. Raasch, & H. Züllighoven (Eds.), *Software Engineering 2007 – Fachtagung des GI-Fachbereichs Softwaretechnik*. Gesellschaft für Informatik., 59-70. Regular Research Papers. Hamburg. 27.-30.03.2007. ISBN: 978-3-88579-199-7.

II Henttonen, K., & Matinlassi, M. (2009). Open-source-based tools for sharing and reuse of software architectural knowledge. *The 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA 2019)*. DOI: 10.1109/WICSA.2009.5290790

III Henttonen, K. (2011). Libre software as an innovation enabler in India: Experiences of a Bangalorian software SME. In S. Hissam, B. Russo., M.G.. de Mendonça Neto & F. Kon (Eds), *OpenSource Systems: Grounding Research. OSS 2011*. IFIP Advances in Information and Communication Technology, 365, 220–232. Springer. DOI: 10.1007/978-3-642-24418-6_15

IV Henttonen, K., Pussinen, P., & Koivumäki, T. (2012). Managerial perspective of open collaboration and networked innovation. *Journal of Technology Management and Innovation*, *7*(3), 135–147. DOI: 10.4067/S0718-27242012000300012.

V Henttonen, K., Kääriäinen, J., & Kylmäaho. J. (2017). Lifecycle management in government-driven open-source projects — practical framework. *International Journal of Information Systems and Project Management*, *5*(3), 23–41. DOI: 10.12821/ijispm050302.

VI Henttonen, K., Pulkkinen, M. & Tyrväinen, P. (2024). Health and orchestration of public-sector open-source software ecosystems. Roles, rules and tools. Accepted to the *Scandinavian Journal of Information Systems* (SJIS).

I served as the first author and was responsible for most of the writing across all the included articles. Except for Article III, which I wrote alone, the co-authors' contributions were as follows. Articles I and II were based on the research ideas of Dr. Mari Matinlassi, who also provided substantial guidance. She contributed to the initial structure and some content of Article II, but after she changed employment, I completed the work independently. For Article IV, I received assistance from Mr. Pasi Pussinen, who helped with data collection and analysis, as well as Prof. Timo Koivumäki, who contributed a summary table and advice on the title and the choice of a forum for publication. Article V was significantly influenced by Dr. Jukka Kääriäinen, who supplied the foundational ideas, conducted all the workshops, and commented on a manuscript draft. Additionally, Mr Jani Kylmäaho assisted in data collection and shared his outstanding expertise in the Oskari technology and community. Lastly, for Article VI, Dr. Mirja Pulkkinen and Prof. Pasi Tyrväinen provided substantial guidance and iterative feedback, meticulously reviewing several drafts and offering insightful suggestions that significantly enhanced the manuscript.

1 INTRODUCTION

This dissertation explores the role of governance in improving the sustainability of free and open-source software (FOSS). The chapter begins by briefly describing the motivations for the research, defining important terms, and tracing their development. It then articulates the research questions. The chapter concludes with an introduction to the structure of the remainder of the thesis.

1.1 Background and Motivation

Over the past two decades, sustainability has remained a critical, though sometimes latent, theme in FOSS research (Curto-Millet & Corsín Jiménez, 2023; Krishnamurthy et al., 2014; Maruping et al., 2019). Governance is central to the sustainability of FOSS projects, involving everything from day-to-day work coordination to long-term strategic planning (Chengalur-Smith et al., 2010; Harutyunyan & Riehle, 2021; O'Mahony & Karp, 2022).

In the early 2000s, FOSS introduced a transformative alternative to traditional proprietary models, challenging established norms of software governance and collaborative work (O'Neil et al., 2021). It was perceived as a disruptive force capable of democratising software access, enabling individuals and small organisations to compete with large corporate entities. It became crucial to assess whether this novel governance model, which Demil and Lecocq (2006) termed 'bazaar governance', was viable and sustainable enough to coexist with—or even replace—traditional software development frameworks.

As businesses began integrating FOSS with profit-driven models, understanding the impact of commercial involvement on FOSS sustainability became essential (West & O'Mahony, 2008). Over time, FOSS became mainstream and commercialised—a process Germonprez and Kendall (2013) describe as the 'domestication of FOSS'. Today, FOSS is no longer simply an alternative; it has become fundamental to numerous critical systems, from web infrastructure to national security applications. Ensuring the sustainability of FOSS is vital to the security, stability, and reliability of technologies that our economies and societies rely on. With governments and large corporations' widespread adoption of FOSS, there is increasing interest in how to govern FOSS projects sustainably at scale.

The motivations for studying FOSS sustainability and the role of governance within it have evolved over the years, reflecting broad changes in technology, business, and society. These shifts intersect with my journey with FOSS, which I began as a hobbyist attending international hacker community events before transitioning into a professional role in FOSS software development in 2006. These experiences have endowed me with practical insights into the importance of sustainable FOSS governance, complementing my academic research in the field.

Today, there is a substantial body of research on FOSS governance (as reviewed by Markus, 2007; Aksulu & Wade, 2010; Crowston et al., 2011; Heimburg & Wiesche, 2022; Linåker et al., 2022). However, much of it remains compartmentalised, focusing on organisational, community, or ecosystem governance. This dissertation contributes by adopting a multi-level analysis of FOSS governance, examining how governance practices across the organisational, community, and ecosystem levels influence the sustainability of FOSS. By exploring the dynamics of these levels, this research provides a holistic perspective on how effective governance can enhance the sustainability of FOSS.

1.2 Goal and Research Questions

The goal of the study is to develop a comprehensive understanding of governance approaches in FOSS across the community, organisational, and ecosystem levels, as well as their impact on the sustainability of FOSS initiatives. The central research question guiding this study is as follows:

RQ How do governance approaches across organizational, community, and ecosystem levels influence the sustainability of FOSS?

Table 1 summarises how each article addresses governance across these three key levels. The table highlights whether each article's emphasis on a particular governance level is primary or supplementary, thus illustrating how the research aligns with the overarching goals outlined in the research question.

TABLE 1 Levels of FOSS governance addressed in each article

Article	**Community governance**	**Organisational governance**	**Ecosystem governance**
I (& II)	Primary	Not addressed	Supplementary
III	Supplementary	Primary	Supplementary
IV	Supplementary	Primary	Supplementary
V	Primary	Supplementary	Supplementary
VI	Supplementary	Supplementary	Primary

Each article also addresses a specific aspect of governance, focusing on key discussions in FOSS research relevant at the time of their publication. The specific research questions for each article are as follows:

RQ-I	How can legal, technical, and social considerations be addressed to ensure the sustainability of a new FOSS community during its establishment?
RQ-II	How can FOSS-based tools best facilitate the sharing and reuse of software architectural knowledge, thereby enhancing the sustainability of FOSS?
RQ-III	How can resource-constraint software SMEs govern FOSS-based innovation in a sustainable manner, particularly within the context of India?
RQ-IV	How do managerial attitudes shape the governance of FOSS-based innovation in software SMEs, and what are the sustainability implications of this?
RQ-V	How can a structured lifecycle management approach improve the sustainability of public sector FOSS communities?
RQ-VI	How do governance activities by orchestrators and keystone players influence the sustainability of FOSS ecosystems in the public sector context?

These research questions collectively contribute to the overarching goal of understanding how governance approaches at various levels influence the sustainability of FOSS.

1.3 Terminology

This section introduces the terminology used in the dissertation and discusses some shifts that occurred during the research process.

1.3.1 Definitions of Key Terms

Free and open-source software (FOSS) is commonly defined as software published under a licence that allows anyone to access, modify, and share the software's source code (Crowston et al., 2012; Osterloh & Rota, 2007; Schreiber, 2023). It is important not to confuse free software with freeware, which refers to software that is available at no cost but for which access to the source code is not available (Wolfenbarger & Smith, 2023). While free software and open-source software refer to the same phenomenon, these terms carry distinct ideological connotations (Fortunato & Galassi, 2021). Although the historical and ideological differences between these terms will be discussed below, they are largely irrelevant to the focus of this study. Therefore, this thesis consistently uses a widely accepted abbreviation, FOSS (e.g., Carosone, 2017; Crowston et al., 2012; Fortunato & Galassi, 2021; Gasson & Purcelle, 2018; Poo-Caamaño et al., 2017).

Reflecting the time and forum of their publication, some articles in the dissertation use alternative terms, such as FLOSS (free/libre open-source software) or OSS (open-source software), synonymously.

A **FOSS project** is a specific initiative to develop a software artefact that is made available with a FOSS licence. Such projects often originate from the needs of an individual or organisation, with the contributor base expanding over time (Franco-Bedoya et al., 2017; West & O'Mahony, 2008). Each FOSS project is characterised by its unique governance structure, design principles, and collaboration tools, facilitating the management of contributions and the project's long-term direction (Jensen & Scacchi, 2007; West & O'Mahony, 2008).

A **FOSS community** is a collective of individuals who contribute directly to a specific FOSS project. Franco-Bedoya et al. (2017) described FOSS communities as the backbone of FOSS projects, taking care of their development, support, and maintenance. Although the term 'community' is used diversely in the FOSS literature, this thesis adopts a definition that is common in FOSS governance studies (e.g. Ljungberg, 2000; Riembauer et al., 2020; West & O'mahony, 2008), in which a FOSS community is comprised of individuals actively engaged in the work of a project.

A **FOSS ecosystem** is defined herein as a set of organisations collaboratively developing software distributed under a FOSS licence. This definition builds on the FOSS definition mentioned above and the software ecosystem (SECO) definition of Manikas and Hansen (2013). These ecosystems are often underpinned by a common technology, such as an extendable software platform (Jansen, 2020; Manikas & Hansen, 2013).[1] In FOSS ecosystems, the technological platform is accessible under a FOSS licence, enabling unrestricted modification and redistribution (Kilamo et al., 2012). A well-known example is the Eclipse ecosystem, in which various stakeholders collaborate on development tools, programming frameworks, and libraries based on the core Eclipse platform (Belo & Alves, 2021; Stevens & Draxler, 2010).

In FOSS and beyond, **governance** can be defined as a shared basis for authority (O'Mahony & Ferraro, 2007). As Shaikh and Henfridsson (2017) note, the term 'governance' in FOSS research typically refers to high-level principles, decision-making structures, and practical management aspects, such as work coordination. In line with Markus (2007), Noni et al. (2011), and O'Mahony and Ferraro (2007), this dissertation adopts a broad perspective on FOSS governance that encompasses work coordination and everyday management.

FOSS governance is examined at three levels: **community governance**, **organisational governance**, and **ecosystem governance**. FOSS research has traditionally focused on community governance, examining decision-making

[1] Manikas et al. (2016) called software ecosystems based on shared technology 'infrastructure-rooted' but also acknowledged the existence of 'actor-rooted' and 'business-rooted' software ecosystems. This observation applies to FOSS ecosystems; for example, the Apache Software Foundation ecosystem is rooted in the interaction between actors instead of a shared technology platform (Digkas et al., 2018). However, the FOSS ecosystems studied in this dissertation could be best described as infrastructure rooted or platform centric.

processes, conflict resolutions and work coordination within FOSS projects (Aksulu & Wade, 2010; Markus, 2007). However, newer studies have broadened the scope of research to include organisational and ecosystem perspectives. An organisational perspective focuses on how FOSS activities are managed within the organisation (Daniel et al., 2018; Harutyunyan & Riehle, 2021), and an ecosystem perspective investigates how governance influences the evolution of broader FOSS ecosystems (Kilamo et al., 2012; Poo-Caamaño et al., 2017).

Sustainability can be generally understood as the capacity to sustain activity and productivity over time (Chengalur-Smith et al., 2010). Definitions of organisational sustainability vary, ranging from those narrowly focused on organisational survival and value creation to those encompassing social, economic, and environmental dimensions (Chengalur-Smith et al., 2010; Curto-Millet & Corsín Jiménez, 2023). For the purposes of this dissertation, I adopt the narrower perspective. The primary goal of the FOSS community or ecosystem is to develop and maintain software products that deliver value to users and other stakeholders (Chengalur-Smith et al., 2010). Consequently, sustainability, within a FOSS context, is defined as the ability to continue producing and maintaining valuable software products over time.

This basic definition of sustainability is enhanced by differentiating between infrastructural, resource-based, and interactional sustainability, in line with Curto-Millet and Corsín Jiménez (2023). **Resource-based sustainability** refers to the capacity of FOSS projects to attract and manage various types of resources, including software assets, human resources, and financial contributions. **Interactional sustainability** refers to maintaining a positive and effective community dynamic in which participation aligns with the shared values and expectations within the community. **Infrastructural sustainability** refers to the capacity to maintain foundational structures and systems that provide essential support and stability for a project[2]. The most important structures are legal frameworks and digital tooling environments.

1.3.2 Evolution of Terminology

In this dissertation, terms are used consistently, as defined previously. However, across the set of included articles, the terminology is not entirely consistent, due to the long timespan of the research and some shifts in focus. I will explain the important terminological shifts in the following.

The use of the term 'community' in FOSS research has evolved over time, and this evolution is reflected in the articles included in this dissertation. Sometimes, the term is used to encompass not only participants in a given FOSS project but, rather, all individuals or organisations involved in a FOSS-related ecosystem or movement. For example, Article I refers to 'the Eclipse community'

[2] The original definition of 'infrastructural sustainability' by Curto-Millet & Corsín Jiménez (2023) encompasses a community's internal infrastructure and strategic integration with broader ecosystems. This thesis concentrates on internal infrastructure at the community level. Additionally, all three types of sustainability are examined at the ecosystem level.

to denote all entities interacting within the expansive Eclipse ecosystem. However, I adhere to a narrower definition of the FOSS community in this thesis. The purpose of this is to capture the difference between more tightly knit communities and broader ecosystems, in which the connections between participants are typically looser.

While the term 'governance' is also mentioned in the original research articles, many of them predominantly use other terms, such as 'hosting' (Article I), 'management' (Articles III, IV, and V), and 'orchestration' (Article VI). These issues fall within the broad definition of governance typical of FOSS studies, as introduced above. Here, FOSS governance encompasses practical work coordination, managerial practices, and organisational structures. Thus, all the included articles contribute to the discourse on FOSS governance. The various governance dimensions will be further elaborated on in the literature review below.

Originally, Articles I and II used the term 'successful' instead of the term 'sustainability', which became popularised in FOSS research after their publication. Article VI uses the term 'health' instead of 'sustainability' to describe a desirable state for a FOSS ecosystem. This choice was made at the time of publication to underscore an ecosystem perspective because most prior research on FOSS sustainability had been centred on individual projects. It also underlined our aim of considering various dimensions of ecosystem well-being beyond prolonged existence. For this thesis, the term 'sustainability' seemed most fitting for encompassing the community, organisational, and ecosystemic perspectives. 'Sustainability' has been defined herein as continued productive activity, not merely continued existence (see 1.3.1), which is close to the definition of FOSS health provided in Article VI. The diverse understandings of 'success', 'sustainability', and 'health' as goals of FOSS governance will be discussed below in the literature review.

1.4 Structure of this Thesis

The rest of this thesis unfolds over several chapters. Chapter 2 presents a brief history of FOSS and describes its essential elements, and Chapter 3 explores the existing academic literature on FOSS governance within communities, ecosystems, and organisations in depth. Chapter 4 describes the research design and methodology employed in this dissertation. Chapter 5 provides an overview of the articles included in this dissertation. Chapter 6 synthesises the findings of these articles, highlighting how governance mechanisms at various levels influence the sustainability of FOSS. Finally, Chapter 7 discusses the critical aspects of these findings, reflecting on their limitations, credibility, and relationship to the prior literature. The thesis concludes with Chapter 8, which summarises contributions to research and practice and outlines directions for future research.

2 ESSENTIALS OF FREE AND OPEN-SOURCE SOFTWARE

This chapter introduces the most fundamental dimensions of FOSS identified by researchers (Medappa & Srivastava, 2020; West & O'Mahony, 2008): ideologies, licences, and the development process. While governance is also recognised as a fundamental dimension by West and O'Mahony (2008), it is the primary focus of this dissertation and will therefore be explored in detail in its own dedicated chapter. The discussion in this chapter will be presented in chronological order to highlight the historical evolution of research and practice.

2.1 Competing and Complementary Ideologies

The history of FOSS can be traced back to the early decades of computing (1950-1960s). This era was characterised by a communal approach to software development, in which software was often developed collaboratively and shared openly among users and developers (Levy, 1984). The associated culture of openness, collaboration, and knowledge-sharing is often referred to as the 'Hacker Ethic' (Himanen, 2004; Levy, 1984). This approach was encouraged because at the time, software was not seen primarily as a commercial product but, rather, as a tool with which to advance computing and research capabilities.

The commercialisation of software began to take shape in the late 1960s and early 1970s, marked by significant events, such as IBM's 'unbundling' decision in 1969 (Ceruzzi, 2003). This decision separated the sale of software and services from hardware, reflecting a shift in the industry's focus towards recognising software as a source of revenue in its own right (Fortunato & Galassi, 2021). This unbundling reflected an inevitable shift in the computer industry's dynamics, which was influenced by the realisation that software development required significant effort and had considerable commercial potential (Ceruzzi, 2003)

When proprietary software development and distribution models became more prevalent, a hacker community began to push back against these changes (Elliott & Scacchi, 2008). This ethos led to the emergence of the free software movement in the 1980s. The movement was spearheaded by Richard Stallman, who had witnessed the shift towards proprietary software in his work at the Massachusetts Institute of Technology (MIT) and was concerned about the deterioration of the culture of sharing and improvement that had defined the early days of computing (Elliott & Scacchi, 2008). Stallman launched the GNU (GNU's Not Unix[3]) project in 1983 to create a completely free Unix-like operating system (Bergquist et al., 2011). This project was not merely a technical effort but, rather, a moral quest to ensure users' freedoms (Bergquist et al., 2011).

In 1984, Richard Stallman and fellow hackers founded the Free Software Foundation (FSF), a nonprofit organisation dedicated to offering legal, organisational, and financial backing for the development of free software, including the GNU Project, among others (Elliott & Scacchi, 2008). Since its inception, the FSF has championed free software for moral and ethical reasons, prioritising user freedom above mere technological benefits. (Fortunato & Galassi, 2021). The well-known phrase 'Free as in free speech, not as in free beer' was coined to dispel terminological confusion and highlight the core concern of the movement: not the cost of software but the essential freedom to utilise, alter, and share the software as a fundamental right (Elliott & Scacchi, 2008).

The FSF defined the four essential freedoms that software must provide to its users (Ljungberg, 2000):

- The freedom to run the program as the user wishes for any purpose (Freedom 0)
- The freedom to study how the program works and change it so that it computes as the user wishes (Freedom 1)
- The freedom to redistribute copies so that the user can help others (Freedom 2)
- The freedom to distribute copies of the modified versions to others. By doing this, the user can give the entire community a chance to benefit from the implemented changes (Freedom 3)

These freedoms can be considered an early definition of FOSS, laying the foundation for the open-source movement a decade later. While intertwined with the free software movement, the history of the open-source software movement has its own narrative. The term 'open source' was coined in the late 1990s to make the free software ideology more appealing to the commercial sector, emphasising the practical benefits of collaboration and transparency in software development over the ethical and philosophical considerations championed by the free software movement (Bergquist et al., 2011). The essay 'The Cathedral and the Bazaar' (Raymond, 1997) articulated the advantages of open, collaborative software development models over traditional, hierarchical models. This essay

[3] Humorous, recursive acronyms were part of the hacker culture of the time (Theodoropoulou, 2008).

captured the essence of the emerging open-source philosophy and crystallised the distinction with earlier free software initiatives.

Eric Raymond and Bruce Parents founded the Open-Source Initiative (OSI) in 1998 to promote the use and development of open-source software, underlining the efficiency, reliability, and innovation associated with open-source development methods (Fortunato & Galassi, 2021; Wolfenbarger & Smith, 2023). It also published the Open-Source Definition (OSD), which is not a licence but, rather, outlines the criteria that a software licence must meet to be considered open source (Perens, 1999). The OSD was adapted from the Debian Free Software Guidelines, which were based on the freedoms previously articulated by the Free Software Foundation: free redistribution, access to the source code, and the ability to modify and share software (Wolfenbarger & Smith, 2023). However, they were rephrased in a manner that highlighted technical access and collaborative development, instead of user freedoms (Fortunato & Galassi, 2021)

Around this time, the terms 'FOSS' (Free and Open-Source Software), 'F/OSS' (Free/Open-source software), and 'FLOSS' (Free/Libre and Open-Source Software) emerged as part of the broader dialogue within the software community about how to describe the best software that was both free, as in freedom, and open source (Fortunato & Galassi, 2021). Scholars adopted these abbreviations, which became used extensively in the academic literature (see, e.g., Amherst et al., 2007; Crowston et al., 2005; Scacchi, 2010). As explained by Fortunato and Galassi (2021), the aim of doing so was to bridge the ideological and philosophical divide between the free software movement, led by the FSF, and the open-source movement, represented by the OSI. The term was chosen for this dissertation to respect the histories of both movements while acknowledging that their ideological differences are irrelevant to this study (see Section 1.3.1).

The 2000s saw a significant increase in corporate engagement with FOSS (Elliott & Scacchi, 2008; Fitzgerald, 2006). Red Hat was one of the first firms to effectively monetise open-source software, with an operating system based on the Linux Kernel and GNU System components (Elliott & Scacchi, 2008). Other trailblazing companies that proved the practicality of open-source-centric business models include the Swedish MySQL AB, which developed one of the world's most widely utilised open-source database engines (Fitzgerald, 2006). Gradually, FOSS evolved from its roots in volunteer hacking into a commercially viable, mainstream form (Bergquist et al., 2011; Fortunato & Galassi, 2021). The relative prominence of the two movements also shifted. The OSI's practical approach gained mainstream traction, whereas the FSF's ideology, which was deeply rooted in the original hacker culture, found its influence somewhat diminished in the face of evolving industry dynamics (Bergquist et al., 2011). However, the term 'FOSS' remains widely used to recognise the historical contributions and value of both movements to the evolution of the phenomenon (Carillo & Bernard, 2015).

2.2 Types of FOSS Licences

FOSS licenses serve as legal frameworks that define the conditions under which software can be freely accessed, used, modified, and shared (Sen et al., 2011). These licences form the bedrock of the FOSS movement, and as discussed afore, the definition of what constitutes 'free and open-source software' is primarily determined by the type of licence under which the software is distributed (Crowston et al., 2012). In academic circles and beyond, FOSS licences are commonly segmented into three categories. Lerner and Tirole (2005) were among the first to provide an in-depth academic analysis of these licence categories, naming them permissive, restrictive, and highly restrictive. The categories have not changed, but subsequent discussions have seen them relabeled as non-copyleft, weak copyleft, and strong copyleft (Sen et al., 2008, 2011). These can be seen as more neutral; many copyleft advocates view the term 'restrictive' as mischaracterising their intent to maximise user freedoms.

Non-copyleft licenses, such as the MIT License (Massachusetts Institute of Technology License) and BSD License (Berkeley Software Distribution License), are characterised by minimal requirements regarding the use, modification, and distribution of software (Lerner & Tirole, 2005; Sen et al., 2011). These licences allow the software and its derivative works to be incorporated into FOSS and proprietary software without requiring the source code to be disclosed or the derivative works to be distributed under the same license (Lerner & Tirole, 2005; Sen et al., 2011). Despite their revival in the late 1990s (Lerner & Tirole, 2005), these types of licences predate copyleft licences (Omar, 2005; Sinclair, 2010). They were initially developed at academic institutions to encourage software's widespread use and adaptation by minimising legal barriers to redistribution and modification (Omar, 2005; Sinclair, 2010).

Strong copyleft licenses, most famously the GNU General Public Licence (GPL), require that any modified versions of the licensed software or software incorporating copyleft-licensed components are released under the same licensing terms (Lerner & Tirole, 2002; Sen et al., 2011). They were a response to the experiences of early FOSS projects, such as BSD Unix, which saw its code used in proprietary products with no contributions being made to the original project (Omar, 2005; Sinclair, 2010). The GPL was initially introduced in 1989 to ensure that all versions of GPL-licensed software remained freely accessible and that the freedom to use, modify, and redistribute the software remained intact (Elliott & Scacchi, 2008). Later, the GPL underwent several revisions, with version 3 being released in 2007 to address new issues, such as patent retaliation (Finney, 2009). Furthermore, the GNU Affero General Public License (AGPL), which was introduced in the same year as GPLv3, adapted to the era of cloud computing and SaaS (Software as a Service) by mandating that the source code be accessible to network users (MacDonald, 2013).

Weak copyleft licences, such as the Eclipse Public Licence (EPL) and the GNU Lesser General Public Licence (LGPL), seek a balance between non-copyleft and strong copyleft licenses, offering a middle ground that allows the licensed code to be integrated within proprietary projects under specific conditions (Lerner & Tirole, 2005; Sen et al., 2011). These licences mandate that any modifications to the FOSS code must be released under the same licence, but they uniquely permit linking this code with proprietary code (Lerner & Tirole, 2005; Sen et al., 2011). They are often utilised to enable proprietary extensions to FOSS platforms or the utilisation of FOSS libraries within proprietary software while still mandating that modifications to the platform or library itself adhere to the same licensing terms (Sen et al., 2008, 2011). The introduction of these licences in the late 90s reflects debates within FOSS communities about the best ways to promote software freedom while accommodating the practical needs of developers and businesses (Sen et al., 2011).

Even though the FSF has traditionally favoured copyleft licences and the OSI has been more inclusive of non-copyleft licenses (Maruping & Matook, 2020), this ideological preference has not influenced the recognition of licences as either 'free' or 'open source.' All widely recognised licences, including those referenced in this study, are classified as 'free' by the FSF (2024) and as 'open source' by the OSI (2024). Consequently, I use the term 'FOSS licence', in line with Fortunato and Galassi (2021), to encompass licences endorsed by both movements. Occasional disputes over nuances in less prevalent licences, such as the Open Watcom licence (Free Software Foundation, 2024), have marginal relevance to the themes explored in this dissertation.

Licensing matters are not merely legal technicalities; they profoundly influence the growth, management, and ultimate success of FOSS projects. Numerous studies have scrutinised the effects of licence selection on project outcomes, highlighting both the advantages and drawbacks of copyleft and non-copyleft licences. For example, it is recognised that a copyleft licence facilitates the attraction of high-quality contributions (Colazo & Fang, 2009; Sen et al., 2008, 2011) and discourages the emergence of closed-source forks, which could jeopardise the sustainability of a FOSS project (Ciffolilli, 2004; MacDonald, 2013). Conversely, projects licensed under non-copyleft licences tend to garner more sponsorship and experience accelerated growth, allowing for diverse revenue models to thrive (Colazo & Fang, 2009; 2005; Stewart et al., 2006).

A relatively recent study conducted by Maruping and Matook (2020) reinforces the abovementioned findings. It also delves deep into the intricate relationship between licence selection, governance structures, and ideological shifts within the FOSS movement. The authors highlight how the re-emergence of non-copyleft licences in the 1990s facilitated greater commercial involvement in FOSS, leading to the rise of FOSS governance models that blend elements of traditional corporate control with efforts to preserve the collaborative ethos inherent in the original movements (Maruping & Matook, 2020). They emphasise that the alignment between a project's licence, its governance model, and the motivations of its contributors is critical to its success (Maruping & Matook, 2020).

2.3 Evolution of the FOSS Development Model

Although the definition distinguishing FOSS from proprietary software concerns licencing, another crucial aspect of FOSS is its distinct development model (Crowston et al., 2012). Weber (2004) wrote, 'The essence of open source is not the software. It is the process by which software is created' (p. 56). The FOSS development process is characterised by open collaboration involving teams of organisationally and geographically dispersed developers. The perceptions and dynamics of the FOSS development model or models, in the plural, as authors now prefer, have evolved significantly over the decades.

The foundational description of the FOSS development model was arguably articulated in the GNU Manifesto (Stallman, 1985). This seminal work envisioned a paradigm in which software development was inherently collaborative, encouraging programmers to share their code and enhancements with the broader community freely. Central to the manifesto's philosophy were the ethics and culture of hacking, as chronicled by Levy (1984). Levy (1984) highlighted the hacker community's commitment to openly sharing knowledge to improve software and hardware collectively. These core principles established the philosophical bedrock for the FOSS development model, setting a framework for collaboration that, while not detailing specific mechanisms, inspired a new approach to software development (Lakhani & Wolf, 2003; Vainio & Vadén, 2012).

In a subsequent development, Eric Raymond (1997; 1999) offered a more detailed examination of the pragmatics underpinning the FOSS development model. His famous essay (Raymond, 1997) described two software development models: the 'cathedral' model, which is akin to traditional software development practices in which the code is developed in a centralised, hierarchical manner and released in carefully planned versions, and the 'bazaar' model, which represents the FOSS approach of open, collaborative development. Utilising the Linux operating system as a case study, Raymond (1997; 1999) argues that the bazaar model, with its open exchange and chaotic evolutionary progress, leads to more effective and rapid development cycles and produces higher-quality software. Furthermore, Raymond articulates several vital principles that are essential for the success of FOSS, including the importance of releasing early and often, leveraging user feedback, and the idea that 'given enough eyeballs, all bugs are shallow' (Linus's Law).

Like Raymond, early academic depictions of FOSS development (Fielding, 1999; Ljungberg, 2000; Mockus et al., 2000; O'Reilly, 1999) portrayed projects as beginning with a single programmer tackling a small problem of personal interest. As the solution gained importance, the programmer would share it with others, leading to the engagement of new users in its ongoing development. These contributors offered code, documentation, bug reports, and translations, among other things, driven by a shared passion for technology, a commitment to software freedom, a desire for recognition, and/or the satisfaction of being part of a community creating something of value (Hars & Ou, 2002; Markus et al., 2000; Ye & Kishida, 2003). Volunteering was thus a foundational pillar and a defining characteristic of the early FOSS movement (Bergquist et al., 2011; Markus et al., 2000).

As scholars sought to understand the groundbreaking phenomenon, the early 2000s also witnessed the emergence of more theoretical accounts of FOSS. Benkler (2002) expanded the conceptual understanding with the introduction of 'commons-based peer production.' This paradigm highlighted how the creative energy of large numbers of people can be coordinated into large, meaningful projects, mostly without traditional hierarchical organisation or financial compensation. Concurrently, an analysis by Lerner and Jean Tirole (2002) delved into the economic principles underlying the FOSS development model, aiming to shed light on the motivations and structural dynamics that support it. Furthermore, Steven Weber's (2004) book "The Success of Open Source" delved into the organisational dynamics and motivations behind open-source collaboration, comprehensively analysing its success factors. This period deepened the academic understanding of the FOSS development model, affirming it as a viable and potentially superior approach to software development.

By the late 2000s, commercialisation was rapidly changing what scholars perceived as the FOSS development model. Brian Fitzgerald (2006) was among the pioneers who highlighted the transformation of FOSS, which was related to increased corporate involvement. He discussed how the FOSS development process had become more structured and less chaotic, with strategic planning and commercial interests being increasingly influential. Similarly, Langlois and Giampaolo Garzarelli (2008) noted that FOSS projects went beyond the binary characterisation of bazaars and cathedrals, embodying hybrid models that combined self-organising production with deliberate planning. Fitzgerald (2006) also outlined several characteristics of the OSS 2.0 era, including a move from generic platforms to specialised domains and the adoption of hybrid models that blended open-source and proprietary software elements.

Over time, as the commercialisation and mainstreaming of FOSS continued and accelerated, the distinction between FOSS and proprietary software development models became blurred (Mäenpää, 2020; O'Neil et al., 2021) or at least less pronounced in certain respects. As part of the phenomenon that Germonprez et al. (2013) call the 'domestication of open source', many company-driven FOSS communities have adopted corporate practices, such as license vetting and formal code reviews (Germonprez et al., 2013; Mäenpää et al., 2016; Medappa & Srivastava, 2020; Schaarschmidt et al., 2015). The proportion of paid developers grew, and the role of volunteering diminished over the years (Carillo & Bernard, 2015). The portion of paid developers varies significantly from one project to another (O'Neil et al., 2021), but for example, paid professionals contributed over 85% of Linux Kernel development by 2017 (Corbet & Kroah - Hartman, 2017). Meanwhile, a few large FOSS communities, such as Debian (O'Neil et al., 2021), and many small ones, such as Pearl/Raku (Hariharan, 2023), have remained volunteer-driven.

However, the influence has been bidirectional, with FOSS development models also influencing proprietary ones. Co-developing solutions with other stakeholders has become commonplace in proprietary settings (Franco-Bedoya et al., 2017; Kilamo et al., 2012). This relates to the growing prevalence of ecosystem-oriented thinking in software production, in which all software ecosystems inherently involve a degree of openness, which is typically achieved through interfaces, necessitating inter-organisational cooperation (Franco-Bedoya et al., 2017; Kilamo et al., 2012). Additionally, many companies have embraced FOSS practices for their own proprietary software development, a phenomenon known as 'inner sourcing' (Buchner & Riehle, 2023; Capraro & Riehle, 2016). This is connected with the use of tools such as Git and Jenkins, which are rooted in FOSS principles, offering an open, collaborative, and efficient development process (Armenise, 2015; Capraro & Riehle, 2016; Kalliamvakou et al., 2015).

Kilamo et al. (2020) describe a trend in which FOSS and proprietary software models converge and swap attributes. They note that projects initially adopting a decentralised, bazaar-style approach tend to evolve into more structured, cathedral-like models with growth and increasing commercial interest. Conversely, initially, closed-source projects increasingly adopt the bazaar model to diversify their contributor base (Kilamo et al., 2020). Several authors (e.g., Carillo & Bernard, 2015; Mäenpää et al., 2016; O'Neil et al., 2021) encourage other researchers to pay attention to the current diversity of FOSS development and governance models, which comprise a variety of practices influenced by factors such as project scale, community objectives, and a project's evolutionary history.

In summary, this chapter has traced the historical developments in FOSS ideologies, licensing, and development models, which have fundamentally shaped how FOSS projects are governed. The ideological foundations of the FOSS movements influence the values that guide governance decisions, such as freedom, transparency, and collaboration. The evolution of licensing practices establishes the legal frameworks that govern software use, modification, and distribution, directly impacting project control and contributor participation. Additionally, the development models reflect various governance structures and practices, ranging from decentralised, community-driven projects to more formalised, corporate-influenced models. By reviewing these elements, I have set the stage for the next chapter, in which I will present a literature review on FOSS governance, the key focus of this dissertation.

3 FOSS GOVERNANCE WITHIN COMMUNITIES, ECOSYSTEMS, AND ORGANISATIONS

This chapter reviews the existing literature on FOSS governance. The literature review was carried out through a combination of database searches and the snowballing method (Jalali & Wohlin, 2012). Initially, 56 key sources were identified using searches[4] on the Web of Science, Google Scholar and the Association of Information Systems Electronic Library (AISeL). The literature base was expanded via backward snowballing (recursively examining reference lists) from newer articles and forward snowballing (examining citing articles) from a few older articles. Some sources were also included based on prior knowledge, the bibliographies of articles included in this dissertation, and AI-generated suggestions.

The review results are organised into subsections on community governance (3.1), corporate governance (3.2), and ecosystem governance (3.3). Lastly, the literature on sustainability as a governance goal is reviewed in Section 3.4. Section 3.4 provides the critical foundation for how the results of this thesis are subsequently presented.

3.1 FOSS Community Governance

This section reviews the literature on FOSS community governance. In defining it, I turn to Markus (2007), who described it as "the means of achieving the

[4] The search terms included various combinations of 'FOSS', including variations such as 'FLOSS' and 'open source'; 'governance', including related terms such as 'management', 'orchestration', and 'coordination'; and 'sustainability', including related terms such as 'success' and 'health'. Some searches were time limited to focus on recent articles. The searches led to a very large number of articles, and not all results were relevant. For example, some articles focused on using ready-made FOSS products for government agencies or on 'open source governance' as the application of open source principles in public administration rather than on the governance of FOSS communities or ecosystems. Articles that provided significant insights into the focus areas of the dissertation or were frequently cited within the FOSS governance literature were prioritised for inclusion.

direction, control, and coordination of wholly or partially autonomous individuals and organisations on behalf of an OSS development project to which they jointly contribute" (p. 152). The first part, 3.1.1, provides a historical perspective, while the actual literature review, conducted using the method explained at the beginning of the chapter, is presented in subsections 3.1.2 and 3.1.3.

3.1.1 Early Research on FOSS Governance

Contrary to the perceptions of chaos sometimes associated with the bazaar metaphor, early FOSS communities already employed forms of governance. Markus et al. (2000) wrote, "Despite the clear potential for chaos, open-source projects are often surprisingly disciplined and successful through the action of multiple, interacting governance mechanisms" (p. 14). This observation was supported by Gallivan (2001), who made a comprehensive content analysis of then-published case studies of FOSS projects. The studies revealed many forms of control being practiced. Some were explicit, such as the rules and norms stated in FAQs, and others were implicit, such as emphasising an individual's professional reputation. Gallivan (2001) argued that FOSS projects rely more heavily on explicit forms of control than trust to regulate contributor behaviour and ensure project success.

Ljungberg (2000) was among the first to recognise various types of governance structures among FOSS communities, including the benevolent dictatorship, rotating dictatorship, and voting committee. The most prevalent model at the time, benevolent dictatorship, was characterised by a single leader—often the project's founder—who retained the final say on key project decisions while negotiating with community members. The rotating dictatorship model extended the benevolent dictatorship by periodically changing the leader, thus distributing responsibilities and decision-making power more evenly among key contributors. The voting committee model represented a democratic approach to decision-making, with co-developers participating in decisions through a voting system. For example, the Apache project used email-based voting with minimal quorum consensus.

Theoretical accounts of the FOSS phenomenon, such as the previously mentioned works of Weber, Benkler and Raymond (see Section 2.3) have argued that FOSS governance is not merely a variation on existing forms of governance but, rather, represents an entirely new structure characterised by a decentralised, non-hierarchical approach that leverages collective intelligence. A few years later, Demil et al. (2006) also described 'Bazaar governance' as an entirely new governance structure that diverged from traditional market, firm, and network governance structures by being grounded in a specific contract, that is, the FOSS license. As also noted by Shaikh and Henfridsson (2017), early theoretical discussions can give the impression that FOSS governance is somewhat monolithic, likely because of the need to underline FOSS governance as something radically new and contrast it effectively with other forms of governance.

3.1.2 Recognition of Various Governance Models

However, the governance of FOSS soon came to be recognised as a multidimensional phenomenon. As FOSS projects grew and aged, scholars (e.g. Amherst et al., 2007; De Laat, 2007; Lattemann & Stieglitz, 2005) observed that their governance models evolved as they matured, following a discernible trajectory. Lattemann and Stieglitz (2005) analysed the changes in governance practices that occurred as FOSS projects went through the four lifecycle phases: introduction, growth, maturity, and decline/revival. In turn, de Laat (2007) divided the evolution of FOSS community governance into three phases: spontaneous, internal, and external governance. In the first phase, governance is informal and heavily relies on a close-knit community's shared norms, values, and work practices. Moving into the second phase, formal governance mechanisms are introduced, and their primary purpose is to coordinate and improve efficiency within larger communities. The third phase witnesses the further formalisation and codification of governance practices, which is primarily driven by the involvement of external parties.

As FOSS underwent commercialisation (see Section 2.3), the governance practices within FOSS projects gradually evolved a more formalised structure, marking the transition to the third phase outlined by De Laat (2007). However, alongside this evolution, significant other changes emerged. O'Mahony (2007) observed that many FOSS communities that are now commercially sponsored no longer operated solely under community management, nor could they be described as self-governed. The authors emphasised the importance of FOSS researchers differentiating between purely community-managed governance and hybrid forms, which amalgamate community-driven and corporate or institutional-driven approaches. An article by West and O'Mahony (2008) further highlighted the differences between what they called 'autonomous' and 'sponsored' FOSS communities. Research on FOSS governance increasingly focused on these sponsored or hybrid communities, which aimed to combine the benefits of FOSS with proprietary ownership and control (Aksulu & Wade, 2010; Shah, 2006).

Markus (2007) also identified the multifaceted nature of FOSS community governance, arguing that it is configurational. Based on a literature review, he proposed six dimensions of FOSS governance, encompassing rules on asset ownership, project chartering, community management, software development processes, conflict resolution, and information/tools management. Tulio and Staples (2011) applied these six dimensions to various real-world FOSS projects and, as a result, distinguished between three distinct configurations of FOSS governance: open, authoritarian, and defined. Open communities are characterised by a decentralised nature and informal or non-existent management structures. In contrast, authoritarian communities mirror the benevolent dictatorship model identified by Ljungberg (2000) and others (e.g. Antikainen et al., 2007), featuring centralised control. Defined communities, meanwhile, are marked by formal management and decision-making processes. In line with de Laat's (2007) evolutionary model, the introduction of formal governance mechanisms is related to communities maturing and growing. More mature or defined forms of governance are typically supported by a FOSS foundation. It has since been noticed, however, that the exact role assumed by a foundation varies extensively from one FOSS community to another (Izquierdo & Cabot, 2018; Luis et al., 2020).

In their work building both on de Laat (2007) and Markus (2007), Noni et al. (2012) proposed four FOSS governance configurations: 'open-source-based governance' (a somewhat counterintuitive term considering all four configurations are within the open-source world), 'tolerant dictatorship', 'collective governance', and 'sponsored governance'. Collective governance and tolerant dictatorship roughly mirror the 'open' and 'authoritative' models identified by Di Tullio and Staples (2013). The two other models relate to situations in which communities mature and formal governance structures are established, which are described as 'external governance' by De Laat (2007) and 'defined' governance by Tulio and Staples (2013). According to Noni (2012), governance trajectories can diverge upon reaching this phase. FOSS communities may adopt an open-source-based model, in which leadership is institutionalised, democratic, and distributed, or a sponsored model, in which elements from traditional institutional governance are incorporated to achieve the strategic goals of commercial sponsors. This confirmed earlier notions (e.g., O'Mahony, 2007; West & O'mahony, 2008) that company-driven FOSS projects often have a different governance structure than volunteer-driven projects.

Shaikh and Henfridsson (2017) argued that despite recognising multiple governance configurations, most articles still assume a singular authoritative structure underlying FOSS community governance. Through a comprehensive literature review, they classify FOSS governance studies into three categories: those assuming libertarian, collectivist, or centralised authoritative structures. Centralised authority research (e.g., Dahlander & O'Mahony, 2010; Di Tullio & Staples, 2013; Weber, 2004) highlights a core group's leadership and decision-making, which are endorsed by others as long as they align with community standards. Libertarian authority research (e.g. De Laat, 2007; Fitzgerald, 2006; Gallivan, 2001; Howison & Crowston, 2014; Noni et al., 2011) focuses on individual freedom, advocating for equal autonomy and expression among all members. Meanwhile, collective authority research (e.g., Fielding, 1999; Hemetsberger & Reinhardt, 2009; O'Mahony & Ferraro, 2007) emphasises the community's collective good, highlighting the fact that governance should reflect the majority's needs and benefits.

3.1.3 Motivating and Coordinating Work in FOSS Communities

Markus (2007) categorised FOSS governance research of from the early 2000s into two subthemes: solving collective action dilemmas and coordinating collaborative work. He also pondered then-emerging studies of developer climate and motivation as a potentially new subtheme. However, with hindsight, these can also be viewed as extensions to the first subtheme, collective action dilemmas. In this section, I will present resolving collective action dilemmas and practical work coordination as separate subthemes of FOSS community governance research.

3.1.3.1 Resolving Collective Action Dilemmas

A significant research topic within the theme of resolving social action dilemmas is how various governance models and mechanisms influence the initial and sustained motivations of contributors (Franck & Jungwirth, 2003; Linåker et al., 2018; Lumbard, 2018, Mäenpää, 2020; Riehle et al., 2014; Von Krogh et al., 2012). This is a recurring theme in FOSS research because governance plays a crucial role in influencing contributor motivation. Governance can offer private benefits, such as improved professional reputation, technological control, or learning opportunities, but perhaps even more importantly, it can establish a working environment that respects the intrinsic, collectivist motivations of contributors and is aligned with their ideological beliefs (Daniel et al., 2018; Linåker & Runeson, 2020; Mäenpää, 2020; Von Krogh et al., 2012).

If FOSS community governance fails to engage developers effectively, this may result in the premature abandonment of the project or the neglect of essential but less appealing maintenance tasks (English & Schweik, 2007; Linåker et al., 2018a; Von Krogh et al., 2012). This has often happened, prompting researchers to argue that, in the context of FOSS, the 'tragedy of the commons' is related to underproduction rather than the depletion of resources (Amherst et al., 2007; English & Schweik, 2007).

In the context of the developer motivation and emotional climate in the FOSS community, interpersonal trust is frequently mentioned (Dabbish et al., 2012; Orsila et al., 2009; Sajadi et al., 2023; Sapkota et al., 2019). Community trust plays a crucial role in attracting new contributors and keeping them engaged (Dabbish et al., 2012; Lane et al., 2004; Orsila et al., 2009). Research has explored the factors influencing trust (Alarcon et al., 2020; Antikainen et al., 2007) and how trust manifests in FOSS projects (Sajadi et al., 2023; Sapkota et al., 2019). The manifestations of mistrust have also been studied (Miller et al., 2022).

In company-driven FOSS projects, sustaining the motivation of volunteers is closely related to the challenges of bridging the often-conflicting interests between volunteer developers and companies (Bonaccorsi & Rossi, 2005; Capra et al., 2008, 2011; Dahlander & Magnusson, 2006; O'Mahony & Bechky, 2008; Riehle et al., 2014). It has been observed that the increasing involvement of companies influences FOSS governance in ways that are not always favourable from the community's perspective (Dahlander & Magnusson, 2005; 2006; Daniel et al., 2018; Yu, 2020). Finding the appropriate balance between openness and freedom, which sustains the motivation of external participants, and control, which enables the appropriation of private profit, has been recognised as a key governance challenge in sponsored FOSS communities (Capra et al., 2011; Noni et al., 2011; O'Neil et al., 2021; Shah, 2006; West & O'Mahony, 2008; Zhang et al., 2022). This is closely related to corporate FOSS governance, which will be discussed below.

3.1.3.2 Work Co-ordination and Quality Assurance

Within the theme of work coordination, many questions are related to how the work is distributed among developers in FOSS projects (Crowston et al., 2005; Dahlander & O'Mahony, 2010). Traditionally, volunteer contributors to FOSS projects select tasks based on their interests and skills, with prioritisation primarily being determined through community discussions (Crowston et al., 2012; den Besten et al., 2008; Jensen & Scacchi, 2005; 2007). This process is enabled by modular software architecture and collaborative tools, such as mailing lists, forums, and issue trackers (Crowston et al., 2012; Dinh-Trong & Bieman, 2005; Langlois & Giampaolo Garzarelli, 2008). However, with the maturation and growth of FOSS projects, work coordination practices have also evolved (Aksulu & Wade, 2010; Germonprez et al., 2013; Scacchi, 2010; Scholtes et al., 2016).

In response to these changes, research attention has been directed towards a work-coordination approach known as 'open superposition', which helps to maintain developer autonomy in large and complex projects (Howison & Crowston, 2014; Li et al., 2020; Maruping & Matook, 2020; Medappa & Srivastava, 2019). Open superposition involves layering sequential, individual, and motivationally independent contributions over time and relies on strategically deferring complex tasks (Howison & Crowston, 2014; Li et al., 2020). This gradually develops a cumulative artefact with increasing functionality and complexity (Howison & Crowston, 2014; Li et al., 2020). Open superposition differs from traditional software development approaches, which prioritise collaborative and simultaneous task execution through modular design and typically address complexity upfront (Howison & Crowston, 2014; Li et al., 2020). It has been found to increase the project's attractiveness among developers (Medappa & Srivastava, 2019), thus also touching on collective action dilemmas.

On the other hand, in some FOSS projects, the increased involvement of companies and the growing number of paid developers have led to the adoption of more authoritative work coordination mechanisms, which prioritise control and predictability over developer autonomy (Aksulu & Wade, 2010; O'Neil et al., 2021; Scacchi, 2010). This shift involves introducing formal work assignment processes and structured project management practices, such as sprints (predefined development cycles) and road maps (Germonprez et al., 2013; Kilamo et al., 2012; Riehle et al., 2014).

Another important work coordination issue is quality assurance. The peer review of code through pull requests (PRs) is a standard practice in which project maintainers or other contributors evaluate contributions before they are merged (Crowston et al., 2005; Yu et al., 2016). Mechanisms relating to the evaluation and lifespan of pull requests in FOSS have received significant research attention (e.g., Alami et al., 2020; Jiang et al., 2019; Moreira Soares et al., 2021; Terrell et al., 2017; Yu et al., 2016). Alami et al. (2020) proposed three PR governance styles: protective, lenient, and equitable. They examined each style's underlying beliefs and norms and the relationship between merits and social connections in the PR acceptance process. The authors concluded that despite their fundamental differences, all PR governance models are intended to ensure code quality and the software's evolution per the community roadmap. They also note that PR governance styles have a non-straightforward relationship with the overall governance style of the community.

Shaikh and Henfridsson (2017) demonstrate that various coordination processes co-exist and overlap, reflecting the multiplicity of authority structures within a single project. Based on a literature review and a single case study of Linux Kernel's development, four coordination mechanisms are established as follows (ibid): Autocratic clearing involves a singular point of entry and exit for contributions, making the central figure a 'clearing house' for all changes. In oligarchic recursion, a group of trusted individuals essentially acts as a bridge between the general contributor base and the central figure(s), which is critical in integrating code obtained from contributors, managing patches, and resolving implementation details. Federated self-governance grants autonomy to sub-projects within the larger project, as exemplified by the semi-autonomous management of different branches of the Linux Kernal project. Meritocratic idea-testing promotes an environment in which contributions are collectively and transparently evaluated on their merits, regardless of the contributor's status within the community. They argue that a single project may contain different coordination mechanisms, with some of these supporting centralised or decentralised authority structures.

3.2 Organisational FOSS Governance

Early scholars quickly recognised the significance of companies developing well-considered strategies for engaging with FOSS. In the nascent literature on FOSS, various critical factors, such as business models and licensing, were thoroughly discussed (e.g., Hecker, 1999; Lerner & Tirole, 2005; Välimäki, 2002; West, 2003). Subsequently, a series of studies delving into the intersection of corporate interests and FOSS communities emerged (e.g., Dahlander & Magnusson, 2005; 2006; O'Mahony, 2007; West & O'Mahony, 2008). An improved understanding of corporate FOSS governance has been attained through the application of established management theories, such as the resource-based view (Alexy et al., 2018; Ghapanchi et al., 2014; Schoder et al., 2019) and institutional theory (Marsan et al., 2012; Nevo & Chengalur-Smith, 2017).Today, organisational or 'corporate' FOSS governance is a solidified research field that focuses on the practices organisations implement to effectively manage their FOSS engagements (Harutyunyan & Riehle, 2021; Lundell et al., 2017).

The popularisation of open innovation theory (Chesbrough, 2003; 2004) has significantly influenced the study of organisational FOSS governance. The theory underscores the importance of absorbing external ideas and technologies to enhance internal innovation processes (outside-in processes) and, on the other hand, bringing internal ideas and technologies to market through external channels (inside-out processes). The original theory leaned heavily on a well-regulated intellectual property (IP) regime, envisioning IP as a commodity to be traded in the market. While Chesbrough (2017) never embraced the connection, other scholars, such as West and Gallagher (2006) and Henkel (2006), identified FOSS as a unique manifestation of open innovation. Following this, companies were encouraged to weave FOSS into their internal innovation management processes. This integration became a focus of scholarly inquiry into how organisations can capitalise on FOSS to drive innovation while navigating its inherent challenges (Faridian, 2023; Schreieck et al., 2023; Tang et al., 2021).

Recent research by Harutyunyan (2022) categorised corporate FOSS governance into four subfields: general governance, supply chain governance, inbound governance, and outbound governance. General FOSS governance pertains to overarching FOSS policies and organisational structures, such as open-source program Offices (see, e.g., Munir & Mols, 2021). Supply chain governance ensures that suppliers' FOSS policies are aligned with the company's (Harutyunyan, 2019; 2020). The inbound and outbound governance subfields, which appear to draw from open innovation theory, are commonly employed in FOSS governance research. Their contents are further examined in the following paragraphs.

Studies on inbound FOSS governance (e.g., Harutyunyan & Riehle, 2019; Lundell et al., 2017; Petersen et al., 2018) examine how organisations integrate open-source components into their proprietary software projects and the methodologies used in selecting FOSS components (Harutyunyan & Riehle, 2019; Spinellis, 2019), compliance with open-source licensing (Alspaugh & Scacchi, 2010; Fendt & Jaeger, 2019; Gangadharan et al., 2012), and strategies for managing security vulnerabilities (Cowan, 2023; Wang et al., 2019). It is noted that while FOSS components may initially appear to be cost free, further assessment reveals indirect financial costs, operational challenges, and strategic complexities, necessitating effective corporate governance practices (Harutyunyan, 2019).

Research on outbound FOSS governance addresses processes and policies via which organisations can contribute to FOSS projects or release their products in open-source form. This includes decisions regarding project open-sourcing (Alamer & Alyahya, 2017; Jansen et al., 2012), licensing choices (MacDonald, 2013; Scacchi & Alspaugh, 2012; Stewart et al., 2006), and community engagement strategies (Gençer & Oba, 2011; Bergquist et al., 2011; Linåker et al., 2018; Maruping et al., 2019). As emphasised by Schaarschmidt et al. (2015), outbound contributions to FOSS development are investments, for which there must be returns. Consequently, much research has focused on FOSS business models (Duparc et al., 2022; Krishnamurthy, 2005; Shahrivar et al., 2018; Thomas et al., 2017). Today, they are often based on developing FOSS platforms and benefiting from surrounding ecosystems (Duparc et al., 2022). Business models influence governance structures: some depend on broad external participation, while others benefit from maximum control over the core platform (Duparc et al., 2022).

In a commercial setting, all outbound FOSS processes necessitate a degree of control over the relevant community or ecosystem (Schaarschmidt et al., 2015). Well-resourced companies may establish their FOSS communities or ecosystems by open-sourcing previously closed products, as seen with IBM's Eclipse and Sun Microsystems's OpenOffice (Joo et al., 2012). Another approach involves companies investing in existing ecosystems and gradually increasing their influence within them (Schaarschmidt et al., 2015). Studies have examined both scenarios: companies launching entirely new FOSS collaborations (Belo & Alves, 2021; Kilamo et al., 2012; West & Wood, 2013) and companies gradually gaining control by participating in pre-existing FOSS communities (Schaarschmidt et al., 2015; Zhang et al., 2022; Zhou et al., 2016). The mechanisms via which companies influence existing FOSS communities often involve human resource tactics, such as hiring influential community members or having employees contribute to the community to gain influence (Daniel et al., 2018; Schaarschmidt et al., 2015).

However, companies can also undermine the community by exerting disproportionate influence on the project's direction, often prioritising their own business interests over those of the community (Butler et al., 2022; Lundell et al., 2017; Zhang et al., 2022; Zhou et al., 2016). This can leave projects particularly vulnerable if a company later withdraws its support. In their seminal article, Dahlander and Magnusson (2006) developed a typology of firm–community relationships, identifying symbiotic, communalistic, and parasitic approaches. Subsequent literature (Daniel et al., 2018; Zhang et al., 2020, 2022; Zhou et al., 2016) consistently shows that corporate involvement in FOSS communities can impact their sustainability positively or negatively, with the direction of impact being largely dependent on corporate governance decisions, namely how companies choose to engage.

While most research on organisational FOSS governance has focused on commercial companies (Linåker et al., 2023), public sector FOSS governance has also been addressed. Most studies on FOSS governance in public sector organisations have traditionally focused on inbound factors, particularly adoption factors, which have been thoroughly reviewed by Sánchez et al. (2020). The role of public procurement practices in FOSS adoption has also been analysed recently (e.g., Lundell et al., 2021). Fewer studies have focused on outbound FOSS governance, such as how public sector organisations can produce FOSS. For example, there are studies on how public sector organisations govern involvement in a particular FOSS project (e.g., Feldman & Horan, 2011; Gamalielsson et al., 2021) and how national policy frameworks influence the readiness of public sector organisations to produce FOSS (Blind et al., 2021; Favario, 2023; Scott & Rung, 2016). Mergel (2015) and Viseur and Jullien (2023) studied how government agencies share code within a broader network. The scarcity of research on outbound FOSS governance may stem from the fact that public sector organisations have only recently shifted from being mere consumers of FOSS to also being producers, as described by Favario (2023).

3.3 FOSS Ecosystem Governance

As ecosystems gained prominence in explaining modern software production, the study of FOSS governance increasingly shifted to examining ecosystems rather than individual projects (Franco-Bedoya et al., 2017; Jansen, 2014). A considerable portion of the research on FOSS ecosystem governance is now situated within broader examinations of software ecosystems (Manikas, 2016) and digital platform ecosystems (Hein et al., 2020). This trend likely mirrors the diminishing distinction between FOSS and proprietary production (see Section 2.3), a phenomenon that is particularly pronounced within the ecosystem framework. However, several attributes rooted in platform openness continue to differentiate FOSS ecosystems from other software ecosystems (Alves et al., 2017).

Regarding the definition of FOSS ecosystem governance, we turn to van Angeren et al. (2013), who describe software ecosystem governance as "the use of strategic procedures and processes to control, maintain, or change the ecosystem" (p. 4). Van Angeren et al. (2016) add that a software ecosystem's governance encompasses technical and business aspects, including managing the software platform and its interfaces and defining business and partnership models. Many authors use the terms 'ecosystem orchestration' and 'ecosystem governance' interchangeably (Bazarhanova et al., 2018; Manikas, 2016;Alvet et al., 2017). Some, like Mukhopadhyay and Bouwman (2019), seem to imply a subtle distinction, with ecosystem governance referring to the overarching design and meta-organisation and orchestration focusing on practical coordination processes. Given my broad definition of governance (see Section 1.3), I consider practical coordination activities to be part of ecosystem governance.

Software ecosystems, which are akin to other digital platform ecosystems, are essentially comprised of a platform owner (the orchestrator, focal firm, or keystone), complementors, and end-users (Heimburg & Wiesche, 2022; Manikas & Hansen, 2013). The platform owner manages and develops the software platform, setting the stage for complementors, who, driven by their own motivations, join the ecosystem to enhance the platform with their products or services (Autio, 2022). End-users leverage these offerings to meet their needs. While this simplified triad can also characterise the structure of FOSS ecosystems, there are notable differences. Unlike most proprietary software ecosystems, FOSS ecosystems often feature multiple platform owners (e.g., Hein et al., 2020; Jansen, 2014; Karger, 2023). Orchestration is often delegated to a foundation, although the extent of involvement on the part of FOSS foundations varies considerably (Izquierdo & Cabot, 2018; Luis et al., 2020). Free and open-source software projects also have various types of complementors, with some contributing directly to platform development (Müller et al., 2019; Teixeira et al., 2015).

Building on an extensive literature review, Hein et al. (2020) identify three governance models for digital platform ecosystems: centralised, consortia, and peer-to-peer. The centralised model places control in the hands of a single key player, such as Facebook or Apple, and it is typically associated with proprietary ecosystems. FOSS ecosystems can also be governed centrally, as arguably exemplified by projects under the Qt group (Carillo & Bernard, 2015; Kilamo et al., 2012). However, the two other models are more common in FOSS ecosystems. In the 'consortia' model, multiple key players share ownership and decision-making responsibilities. Hein et al. (2020) illustrate this with the example of the open-source Cloud Foundry project, in which prominent technology companies, such as Cisco, SAP, Dell. IBM, and VMware, collaboratively oversee the ecosystem through a dedicated foundation. The peer-to-peer model disperses control and decision-making authority across various stakeholders. Due to its highly decentralised decision-making structures (Zanotti & Vélez, 2020), the Gnome ecosystem may be considered an example of peer-to-peer governance.

Jansen (2020) developed a maturity model for software organisations to use in assessing and refining their ecosystem governance, which has been validated through case studies encompassing proprietary and FOSS ecosystems. The author suggests that large-scale, industry-friendly FOSS platforms are managed similarly to closed platforms, a conclusion stemming from their empirical investigation focusing on industrially driven FOSS ecosystems, such as Eclipse and Android. Although these ecosystems are formally associated with a consortium governance model, a single company (IBM for Eclipse and Google for Android) will significantly dominate the others (Bettenburg et al., 2015; Mizushima & Ikawa, 2011).

However, studies on other large-scale, industry-supported FOSS ecosystems, such as Apache (Gharehyazie et al., 2015; Wang et al., 2023), OpenStack (Zhang et al., 2020; 2022), and Gnome (Oliveira & Alves, 2021; Zanotti & Vélez, 2020) have found that they operate without the control of a single company, unlike Eclipse or Android, instead benefiting from the support of various stakeholders, including corporations, individuals, and nonprofit organisations. Their governance models incorporate several mechanisms to foster open, inclusive, and meritocratic participation, preventing overt control by a single actor. For example, the Gnome Foundation enforces diversity within its Board of Directors through policies such as forced rotations and a limit on the number of representatives from one organisation (Gnome Foundation, 2023; Zanotti & Vélez, 2020). Projects under the Apache Software Foundation adhere to a sophisticated voting system for consensus-building on critical decisions, including code modifications and project releases (Wang et al., 2023). Meanwhile, despite being significantly influenced by corporate interests, OpenStack benefits from contributions by hundreds of companies, fostering varied collaborations that mitigate the impact of any single entity (Zhang et al., 2020, 2022).

Additionally, numerous case studies focusing on smaller FOSS ecosystems, including Matomo (Gamalielsson et al., 2021), PyPi (Valiev et al., 2018), Skyline (Pino et al., 2020), Ruby (Jansen et al., 2011), and Pearl (Hariharan, 2023) have touched on governance issues, underscoring the wide range of approaches to ecosystem governance within FOSS. This diversity is echoed by several authors (Kilamo et al., 2020; O'Neil et al., 2021), who all emphasise the variety of governance frameworks across FOSS ecosystems. Furthermore, recent case studies from various countries, such as Kenya (Hewapathirana, 2017), Finland (Bazarhanova et al., 2018), and Sweden (Linåker & Runeson, 2020), have shed light on the unique dynamics of FOSS ecosystems with a societal purpose in which public sector organisations play pivotal roles as keystone players or orchestrators. These ecosystems differ from their company-driven counterparts in terms of resourcing and objectives, and therefore, the former require different governance approaches (Mukhopadhyay & Bouwman, 2019).

Based on an extensive literature review, Alves et al. (2017) identified governance mechanisms that various players, mostly keystones and orchestrators, employ in open and closed software ecosystems. These mechanisms were grouped into three categories: value creation (e.g., innovation strategies, revenue models, and cost-sharing), the coordination of players (e.g., partnership models, conflict resolution, and risk management), and organisational openness and control (e.g., knowledge-sharing, architecture design, quality standards, and roadmaps). Many of these mechanisms have recently been explored in the context of FOSS ecosystems, including knowledge-sharing (Nimmagadda et al., 2022), architecture design (Amorim et al., 2023; Moon, 2021), and partner/contributor management (Kaur et al., 2022; O'Mahony & Karp, 2022). These studies build on the older research on FOSS community governance (see Section 3.1) but adopt an ecosystemic perspective.

3.4 Desired Outcomes of Governance

In the initial phases of FOSS governance research, scholars primarily aimed to understand the impact of governance mechanisms on the success or failure of FOSS projects (Capra et al., 2008; Markus, 2007; Sagers, 2004). Success was often quantified using metrics such as user and developer adoption rates, the volume and frequency of contributions, and stakeholder satisfaction levels (Lee et al., 2009; Sagers, 2004). Over time, the notion of project success expanded to incorporate a longer-term outlook, leading to an increased focus on sustainability in FOSS studies (Chengalur-Smith et al., 2010). Sustainability highlights the ability of FOSS projects to continue thriving over an extended period (Chengalur-Smith et al., 2010; Noni et al., 2011; Nyman & Lindman, 2013). According to Chengalur-Smith et al. (2010), "sustainability requires certain levels of activity to be maintained over a long period, whereas success can be measured at one particular point in time." (p. 660).

The shift from success to sustainability reflected the growing acknowledgement of the challenges faced by FOSS governance as communities grow and evolve (Chengalur-Smith et al., 2010; Noni et al., 2011; Nyman & Lindman, 2013). It also mirrored the increasing popularity of using ecological concepts derived from the natural sciences in organisational research (Chengalur-Smith et al., 2010). Central to this perspective is sustaining a positive feedback loop in which FOSS projects attract a community of users who gradually evolve into contributors (Butler, 2001; Howison & Crowston, 2014;

Kane & Ransbotham, 2016). These contributors, in turn, enhance the software, attracting even more users and reinforcing the cycle (ibid).

The FOSS sustainability debate has also become linked with the 'tragedy of the commons' (Hardin, 1968) debate. Initially, this concept described the issue of over-depleting unmanaged common-pool resources (Hardin, 1968; 2007). Digital commons, such as FOSS, however, are often regarded as different because digital resources are not depleted through use and can be shared indefinitely (Amherst et al., 2007). The debate was reframed in the FOSS context to refer to a situation in which many wish to use the resource but few are willing to contribute to its production (Amherst et al., 2007; English & Schweik, 2007).

In her work on environmental commons, Ostrom (1999; 2009) argued that the tragedy of 'commons' can be avoided and that careful community governance can sustain common pool resources. Her ideas have significantly influenced FOSS sustainability research, as Curto-Millet and Corsín Jiménez (2023) noted. Sustainability and, consequently, the ultimate purpose of FOSS governance have often been viewed, in this context, as a way to sustain community participation and inflow of contributions (Chengalur-Smith et al., 2010; Dennehy et al., 2023; Fang & Neufeld, 2009; Linåker et al., 2018b).

Curto-Millet and Corsín Jiménez (2023) argue that despite the change in terminology, the understanding of sustainability in FOSS studies has remained centred on success. They contend that FOSS studies have suffered from the binary framing of sustainability as a "stabilised success or failure" (p. 1). They attribute this partially to the influential position of Ostrom's (1999; 2009) work on environmental commons, in which systems are either sustainable or not based on their capacity to manage resources effectively without depletion. They continue to argue for a multidimensional view of FOSS sustainability and propose a typology of FOSS sustainability consisting of resource-based, interactional, and infrastructural sustainability (see Section 1.3.1 for definitions).

According to Curto-Millet and Corsín Jiménez (2023), the three types of sustainability—resource-based, interactional, and infrastructural—interact in ways that significantly influence the overall sustainability of FOSS projects. These interrelationships may result in synergies, in which enhancements of one sustainability type positively affect the others, amplifying beneficial outcomes across the project. Conversely, they may involve trade-offs, in which progress in one area can inadvertently undermine that in another. Additionally, there are instances where these sustainability types are independent; changes in one do not necessarily impact the others. Understanding these dynamics is crucial for effective governance, allowing project leaders to align management strategies with the specific sustainability needs and interactions within their FOSS projects.

In addition to sustainability, health emerged early as another crucial aspect of the well-being of FOSS projects. In the context of FOSS, health was first mentioned by Wyn (2007), who described it as vigour (activity level and productivity), resilience (the ability to adapt and recover), and organisation (the efficiency of the governance structure). Wahyudin et al. (2007) introduced the notion of health indicators, which serve as a method for assessing and quantifying attributes that can potentially forecast the viability of individual open-source projects. Together, they build a foundation for a large body of research on health assessment for FOSS projects (Linåker et al., 2022).

Much of this research, which was recently reviewed by Linåker et al. (2022), focuses on quantitative measurements and mining them from collaborative platforms such as GitHub. Link and Germonprez (2018) adopted a less common qualitative approach, interviewing practitioners to assess open-source project health. They identified three key areas—community, code, and resources—in which the health of open-source projects becomes evident and significant. Interestingly, this parallels the sustainability types identified by Curto-Millet and Corsín Jiménez (2023). Although FOSS health assessment shares similarities with FOSS sustainability, its primary objective has focused on assisting organisations in selecting FOSS components rather than serving as a governance goal like sustainability.

However, this changed when the body of research on FOSS project health was (re[5])integrated with existing theories of business ecosystem health (Iansiti & Levien, 2004) and software ecosystem health (Hyrynsalmi et al., 2015; Manikas & Hansen, 2013). For example, Jansen (2014) compiled an inventory of metrics mentioned in the prior FOSS literature into a comprehensive framework and extended them to cover both FOSS projects and networks. She organised these metrics in alignment with the business ecosystem health dimensions proposed by Iansiti and Levien (2004): productivity, resilience, and niche creation. Unlike traditional research on FOSS project health, ecosystem research has placed significant emphasis on the interplay between ecosystem health and governance (Mukhopadhyay & Bouwman, 2019; Oliveira & Alves, 2021).

Ecosystem governance and orchestrating activities are perceived as crucial in sustaining and improving the health of ecosystems (Hyrynsalmi et al., 2015; Iansiti & Levien, 2004; Manikas & Hansen, 2013). Several articles, as comprehensively reviewed by Alves et al. (2017), delve into specific orchestration activities, such as conflict management procedures, assessing their influence on software ecosystem health. Indeed, health is often posited as the goal of ecosystem governance. For example, Alves et al. (2017) define governance mechanisms as "managerial tools of participants in software ecosystems …. that have the goal of influencing an ecosystem's health" (p. 2).

While much research on enhancing ecosystem health through governance mechanisms has focused on keystone players, Carst and Hu (2023) underscore the significant role of complementors in shaping the well-being and overall trajectory of software ecosystems. This is particularly pertinent in the context of FOSS, in which numerous studies highlight the influential role of peripheral ecosystem participants (Bazarhanova et al., 2018; Dhungana, 2013; Tiwana, 2015).

[5] One could talk about 'reintegration' because originally, the influential article on FOSS health by Wyn (2007) was, in turn, influenced by the work of Iansity and Levien (2004).

3.5 Summary of Literature Review

This chapter thoroughly reviewed how research explored FOSS governance at three levels: community, organisation, and ecosystem. Studies in FOSS community governance evolved from acknowledging foundational governance structures to more nuanced explorations of governance diversity. The growing involvement of corporations in FOSS gave rise to a new research focus on corporate FOSS governance. As the concept of ecosystems gained prominence in software research, attention shifted towards FOSS governance and orchestration at the ecosystem level. The well-being of FOSS has long been seen as a key objective of governance, with the term 'sustainability' being increasingly used to describe this goal since the 2010s.

However, FOSS governance research remains fragmented across these levels, with few exceptions, such as Jansen (2014), who considered the 'project' and 'network' levels in parallel. Most studies concentrate on specific governance activities and their impact on FOSS sustainability at a single level, such as conflict resolution or sponsor attraction, often neglecting the interconnectedness of governance across these levels. This fragmentation has created a gap in understanding the governance challenges relevant at each level and how they interrelate, ultimately affecting the overall sustainability of FOSS.

4 RESEARCH DESIGN AND METHODOLOGY

This chapter presents the research design and methodology employed in this dissertation. Section 4.1 outlines the overall research approach and philosophical foundations. Section 4.2 explains case study designs, providing justifications for the selection of case organisations and the number of cases. Sections 4.3 and 4.4 then detail the methods used for data collection and analysis, respectively.

4.1 Research Approach and Philosophical Stance

The case study is an empirical inquiry that investigates a phenomenon within its real-life context (Runeson & Höst, 2009; Simons, 2014; Yin, 1998). It focuses on understanding how the phenomenon interacts with its environment through an in-depth analysis of specific cases rather than using controlled settings or statistically representative samples (Runeson & Höst, 2009). In this dissertation, the case study was selected as the primary research approach because it aligns well with the nature of the phenomenon being studied, the research objectives, and the types of research questions posed, as detailed below.

As a phenomenon, FOSS governance involves complex interactions among diverse stakeholders. Case studies are particularly suited to exploring these interactions in depth, offering a nuanced understanding of governance processes. Given that FOSS communities and ecosystems vary significantly in size, scope, purpose, and culture (Carillo & Bernard, 2015), the case study method effectively captures these contextual influences on governance practices and outcomes.

Much of my research has a strong practical orientation. Case studies are arguably well suited to delivering practical contributions because they can capture rich, contextual information, which allows practitioners to assess the results' applicability to their situations (Chetty, 1996). Furthermore, my research questions are 'how '-oriented, necessitating a deep understanding of operational issues instead of reporting incidents or frequencies. Case studies are often regarded as particularly effective in analysing 'why' and 'how' questions due to their explanatory nature (Rowley, 2002; Yin, 2009).

The various ontological and epistemological positions in IS research shape the approaches to case study research. Positivist case studies aim to identify general laws or principles that can explain behaviour in specific contexts. These may be formulated deductively, through hypothesis testing, as highlighted by Yin (1998; 2009) or inductively, through theory generation, as advocated by Eisenhardt (1989). In either case, the emphasis is on the researcher's objectivity and the ability to replicate the findings across contexts. In contrast, the interpretivist approach to case study research, as described by Flyvbjerg (2004) and Walsham (1995), emphasises understanding the meaning that participants attribute to their experiences and the context in which they operate. They underline the depth and breadth of insights and rich contextual descriptions, often placing little importance on generalisation. Research within this paradigm is seen as fundamentally subjective, with researchers being encouraged to reflect on their own biases and subjectivities (Goldkuhl, 2012; Walsham, 1995).

The philosophical position of this dissertation places it between positivism and interpretivism. It could best be described as falling within a broad category of perspectives described as critical realism (Groff, 2004; Sayer, 2010). Critical realism combines ontological realism, a belief that some mind-independent reality is 'out there', with epistemological relativism, a position that our knowledge of reality is always conceptually mediated and thus approximate or probabilistic at best (ibid). Unlike positivist approaches, which prioritise observable phenomena, and unlike interpretivism, which emphasises subjective experiences, critical realism focuses on uncovering the underlying structures and mechanisms that shape both observable phenomena and subjective experiences within a broader context (Easton, 2010).

Critical realism has influenced the case studies presented in this dissertation in terms of the objective of describing and explaining empirical realities beyond the subjective perceptions of the research participants (ontological realism). Additionally, it has shaped my attempts to critically assess how both my subjectivities and those of the informants have influenced the research process (epistemological relativism). Although the research questions primarily focus on the 'how', they implicitly delve into the 'why' by examining the factors contributing to the sustainability of FOSS ecosystems. From my perspective, many FOSS governance mechanisms are neither directly measurable nor entirely subjective. Furthermore, according to some scholars, such as Easton (2010), critical realism aligns well with the case study approach, as it supports our understanding of causalities within specific contexts.

While critical realism forms the foundation of my thinking, the practical orientation of my research also has a great deal in common with pragmatism (e.g., Mead, 1938; Morgan, 2014). Pragmatism emphasises the practical application of ideas and the usefulness of theories in solving real-world problems (Goldkuhl, 2012). In my case studies, the aim is not only to understand the underlying mechanisms of FOSS governance but also to provide actionable recommendations that practitioners can implement. However, I also believe that understanding underlying structures and mechanisms is essential for a comprehensive analysis of any phenomenon. Consequently, I feel comfortable aligning myself with critical realism in the philosophical sense despite the practical focus of my work. I also believe critical realism is not at odds with a practical orientation. While the existence of a mind-independent reality is unverifiable, assuming its presence surely facilitates practical decision-making in professional contexts.

Both critical realism and pragmatism have been associated with favouring mixed-method approaches (Easton, 2010; Goldkuhl, 2012).However, this dissertation primarily relies on qualitative methods, although some case studies incorporate small quantitative elements. Qualitative methods provide a deep, detailed analysis of complex issues involving human behaviour, and many consider them especially suited to grasping organisational dynamics (Cassell et al., 2004). Furthermore, authors such as Chetty (1996) encourage using qualitative methods in case studies aimed at making practical contributions. He (ibid) argues that quantitative approaches, by reducing organisational characteristics to a limited number of variables, often strip away the contextual richness necessary for the findings to be genuinely helpful for practitioners. It is also noteworthy that, partially due to the extensive data mining opportunities offered by FOSS platforms such as SourceForge and GitHub (Almarzouq et al., 2022; Kalliamvakou et al., 2016), quantitative research has become very popular in FOSS research (see Crowston et al., 2012; Linåker et al., 2022 for reviews). Therefore, I perceived a greater need for qualitative research to study the aspects of FOSS governance that cannot be derived solely from measurable variables.

4.2 Case Study Designs

This dissertation consists of studies employing both single-case and multi-case designs. The choice of analytic unit—a company, community, or ecosystem—varies depending on the research objectives. Brief introductions to the case contexts, objectives, and research questions were provided in Section 1.2. This section focuses on other decisions in case study design, particularly the number of cases, case study approaches, and criteria for case selection.

To describe the diverse study designs employed, Stake's (1995, 2005) typology of case study approaches and Flyvbjerg's (2006, 2011) case selection strategies are referenced herein. Stake's typology focuses on the purpose and scope of the case study. It distinguishes between intrinsic case studies, conducted out of specific interest in a unique case; instrumental case studies, which provide broader insights through an in-depth focus on a single case; and collective case studies, which examine multiple cases to identify patterns across contexts (Stake 1995, 2005) Flyvbjerg (2006, 2011) outlines several case selection strategies that align with these purposes, including critical cases, which are strategically important in relation to the problem being studied; deviant cases, which are unusual or special in some way; and maximum variation cases, which capture a wide range of perspectives and experiences. These categories are not rigid classifications but serve as concepts to describe the case studies in this dissertation to some extent.

Table 2 summarizes the case study designs across the articles, detailing the number of cases, the type of organization studied, and their relationship with Stake's (1995, 2005) case study types and Flyvbjerg's (2006, 2011) case selection strategies. The following paragraphs elaborate on these choices.

TABLE 2 Summary of case study designs across articles

Article	**Number of cases**	**Case name(s)**	**Organisation type**	**Case selection strategy (Flyvberg 2004, 2011)**	**Case study type (Stake, 1997, 2005)**
Articles I and II	Single case	Stylebase for Eclipse	FOSS Community	Critical case	Instrumental case study
Article III	Single case	Mahiti Infotech	Company	Deviant case	Intrinsic case study
Article IV	Multi-case study, six cases	Unnamed SMEs	Company	Maximum variation cases	Collective case study
Article V	Single case	Oskari Platform	Community	Critical case	Instrumental case study
Article VI	Multi-case study, three cases	Decidim, Oskari, Plone	FOSS Ecosystem	Maximum variation cases	Collective case study

Single studies allow for a deep, comprehensive, and detailed examination and are particularly well suited to studying cases deemed 'deviant' or 'critical.' (Flyvbjerg, 2004). For instance, the case company in India studied in Article III was notably active in the FOSS scene, contrasting sharply with many other Indian software companies at the time, which reportedly lacked the capabilities to utilize FOSS components, let alone contribute to FOSS communities. This aligns with Flyvbjerg's (2004, 1011) concept of a deviant case. This case study can also be described as primarily intrinsic, driven by a specific interest in understanding the unique characteristics of the company6. In contrast, Article I and IV present more instrumental case studies and, in terms of case selection criteria, can be seen as examples of 'critical' cases. According to Flyvbjerg (2011), a critical case allows for logical deductions such as, 'If this is (not) valid for this case, then it applies to all (no) cases.' For example, the Oskari platform, which is featured in Article IV, was a flagship public sector FOSS initiative in Finland. The Finnish Ministry of

6 While the case study presented in Article III was largely intrinsic in the beginning, some broader lessons could also be drawn, especially due to similar patterns later identified in other cases. Stake (1997) warns against categorizing case studies strictly into three 'boxes', reminding that is rarely possible. The case studies in this dissertation could be seen as aligning with these categories to a degree rather than being pure representations.

Finance aimed to use it as a model for open-source practices that could be replicated in other public sector organizations.

The use of multiple case studies allows for a comprehensive exploration of differences and similarities across contexts (Bryman, 2008; Eisenhardt, 1989). Deriving recommendations for policy or practice can also become easier via the use of multiple cases, given the reduced risk of idiosyncratic findings (ibid). I adopted a multi-case design for two studies to obtain a comprehensive view of a research topic. The study of six FOSS-engaged software companies in Article IV revealed a broad spectrum of managerial attitudes towards FOSS across firms. In Article VI, consistent patterns emerged in three public sector FOSS ecosystems despite variations in terms of national origin, size, and domain. In Stake's (1997, 2005) terms, these could be categorized as collective case studies. The case selection criteria were centered on achieving 'maximum variation' (Flyvbjerg, 2004, 2011), aiming to identify cases that, despite diverse variations, have shared patterns that cut across these variations. Setting the number of cases between three and six was an attempt to strike a balance between maximising data richness and maintaining a manageable scope while also aiming to achieve a reasonable level of thematic saturation (see Runeson et al., 2012).

In addition to the mentioned rationales for case selection, convenience of access also played a role. Access to organizations or communities was often facilitated by personal contacts, which helped in building rapport with informants. In some instances, my involvement with the organisations or communities consisted of a full-time professional role, allowing for long-term participatory observation. This deep engagement complemented interview findings and enriched crucial background knowledge. One of the case studies (Article V, Oskari platform) was methodologically framed as an 'action case' (Vidgen & Braa, 1997) to reflect its dual objective: revealing the case context and facilitating actionable improvements. While the interventionist elements were not as pronounced as in full-fledged action research, they surpassed what could be justified by participatory observation alone.

4.3 Data Collection Methods

Triangulating multiple data sources is one of the key strengths of case study research (Runeson et al., 2012), as it allows for a nuanced and comprehensive understanding of complex phenomena. This dissertation employed various data collection methods across the included articles to capture diverse perspectives and insights. Table 3 summarises the data sources and the volume of the data analysed for each article, highlighting the use of interviews, document analysis, observation, and software analysis.

TABLE 3 Summary of data collection methods

Article	Interviews	Documents	Observation	Software
Article I	N/A*	63 pages	6-month work period	10 250 LOC*
Article II	N/A	N/A	N/A	3 applications (87 FP*)
Article III	4 h 28 min	50+ pages	10+ pages of notes	N/A
Article IV	8+ h*	50+ pages	N/A	N/A
Article V	5+ h	N/A	6-month work period	N/A
Article VI	13 h 40 min	223 pages	21 pages of notes	N/A

*Explanations: N/A = not applicable, LOC = lines of code, FP = function points, h = hours

The data collection methods used are as follows:

Interviews: This dissertation employs semi-structured interviews, with the level of structuring varying across the studies. The interviews for Articles III and IV were more structured, reflecting the choice of a theory-driven approach for these early phases of the research. In contrast, later interviews, as in Articles V and VI, allowed more space for spontaneous discussions and, thus, exploring emerging themes. All interviews were recorded and transcribed, with a few exceptions due to technical failures or instances in which informants did not permit recording; in such cases, hand-written notes were taken. Informant selection was influenced by the research question posed in each article, and I made a general attempt to include individuals from diverse professional roles and positions within the case organisation to gather a broad range of perspectives.

Documents: The documents analysed fell into three categories: unpublished and potentially confidential business documents received from informants, such as balance sheets and strategy documents; publicly available online documents, including FOSS community descriptions and governance rules; and ongoing discussions on FOSS forums, such as on digital platforms, IRC (Internet Relay Chat) channels, and mailing lists. Unpublished documents and tools were stored in a dedicated case study database or folder, with links being saved for publicly accessible documents. Only excerpts that were relevant to the research questions were retained for lively discussion forums.

Observation: Observation methods ranged across the spectrum of participant observation. As a researcher, I assumed a dual role: that of an observer who systematically noted behaviours and interactions within the setting and that of a participant who engaged to varying degrees depending on access and the objectives of the case context. Levels of participation ranged from moderate, such as mingling informally at community events or in company cafeterias, to full involvement, such as contributing to an open-source software project and leading design workshops in a professional capacity. Observational data were primarily recorded through notetaking. Furthermore, some of the documents described above were accessed during participant observation.

Software: In Article II, a black box testing approach was adopted, in which three open-source software tools were installed and used without accessing or analysing the underlying code. This methodology was crucial in evaluating the tools from an end-user's perspective, focusing on external functionalities and usability. Conversely, the research in Article I involved a thorough analysis of the existing Eclipse architecture, which is essential for integrating a new plugin seamlessly. This analysis enhanced the plugin’s technical utility and deepened our understanding of modular development within software ecosystems. Thus, software served as a data source in Article I. In Articles V and VI, although knowledge of the platform software was vital for context, aiding in the discussions with informants and the interpretation of the findings, it did not constitute an independent data source.

4.4 Data Analysis Methods

This section offers an overview of the data analysis methods employed. It begins by detailing the primary analysis method, which is qualitative content analysis, and then briefly describes the additional analytical methods used in the included articles.

4.4.1 Qualitative Content Analysis

Qualitative content analysis was applied to interview transcripts, selected documents, and observation notes. Both directed and conventional approaches were used. In directed content analysis (Hsieh & Shannon, 2005), initial coding categories are derived from existing theory, facilitating theory-driven data exploration. Conversely, in conventional content analysis, categories are developed directly from the data, allowing themes to emerge naturally, without theoretical preconceptions (Hsieh & Shannon, 2005).

Template analysis (King, 1998; 2012) was utilised throughout the dissertation and provided a structured yet flexible approach to thematic coding. Central to this method is a coding ‘template’—a hierarchical organisation of themes deemed essential for analysing the dataset. This template guides the initial stages of analysis and is also iteratively refined as the analysis progresses. Themes may be refined, combined, split, or discarded, and new themes may emerge, necessitating revising the template. This iterative process is continued until a stable template that accurately reflects the data’s themes is achieved.

There were several pragmatic reasons for choosing template analysis. The method is adaptable to the specific needs and contexts of various research projects and adept at managing relatively large volumes of data. It effectively balances a priori theoretical concerns with emergent themes derived from the data, supporting both conventional and directed content analysis. An additional advantage of template analysis is its flexibility in terms of not being tied to any theoretical framework or epistemological stance.

The initial study containing interviews (Article III) adopted a more theory-driven approach, using directed content analysis with prior codes derived from open innovation theory and FOSS business research. In contrast, the later studies (Articles III-VI) adopted a more open-ended coding approach that was informed by my experiences with the limitations of a strictly theory-driven coding strategy. Article III can be categorised as employing directed content analysis, but it used only a few initial codes related to open innovation theory. Article V utilised purely conventional content analysis, with all codes derived directly from the data. Article VI employed a combination of directed and conventional content analysis, emphasising the latter.

All studies that implemented formal thematic coding had interview transcripts as their primary data source. Documents and observations were crucial in supplementing and corroborating the data collected from the interviews. While some core documents and observational notes underwent systematic coding identical to that the interview transcripts were subjected to, most of these supplementary data were integrated into the content analysis differently. Alongside the thematic coding of the interview data, documents and observation notes were scrutinised for examples that either corroborated or contested the themes derived from the interviews. This pragmatic approach allowed for the inclusion of much larger sets of supplementary data. The role of triangulation was emphasised: data from interviews were often supported, but sometimes contested, by evidence derived from documents or observations, prompting the further examination of conflicts within the data. This led to credible and robust research findings.

4.4.2 Specialised Analytical Methods

In addition to the qualitative content analysis employed across several research articles, this dissertation integrates the following specialised analysis methods to address specific aspects of the research questions in some articles:

Quality-Driven Architecture Design and Analysis (QADA): Utilised in Article I, QADA (Matinlassi et al., 2002; Niemelä & Ihme, 2001; Ovaska et al., 2010) is an academically grounded software engineering methodology that ensures software architectures have critical quality attributes, such as performance, security, maintainability, and scalability. This method underpinned the development of an open-source Eclipse plugin and supported its integration within the Eclipse ecosystem.

Normative Information Model-Based Systems Analysis and Design (NIMSAD): Applied in Article II, the NIMSAD (Jayaratna, 1994; Simister, 1996) framework was used to evaluate and assess the functionality and usability of three open-source tools. Its structured approach offers a robust framework for systematic analysis, enhancing our understanding of information systems.

Value Network Analysis (VNA): VNA (Allee, 2003; 2008) is a practice-oriented method of mapping relationships and analysing value exchanges among stakeholders. Used in Article III, alongside qualitative content analysis, it helped delineate the company's strategic position within FOSS-related value networks and quantify associated revenue streams.

Software Configuration Management (SCM) Plan: Software configuration management planning (Leon, 2015) leverages strategic planning to refine software processes and lifecycle activities. Due to its practical relevance and adaptability, it was selected to guide the iterative improvement of management practices within a public sector FOSS community, as reported in Article V.

These methods are primarily qualitative but also incorporate quantitative elements, creating a mixed methods approach. They blend academic theories with practical IS applications and, in my view, have been particularly helpful in enhancing the practical relevance of the research findings.

5 OVERVIEW OF THE INCLUDED ARTICLES

This chapter provides a concise overview of the six articles that form the core of this dissertation. Each article is summarised in a structured format, covering the objective, methodology, and key results.

5.1 Article I: Contributing to Eclipse – A Case Study

Objective

The objective of this study was to understand the requirements for building a sustainable FOSS community, particularly within an existing platform ecosystem such as Eclipse. Equal consideration was given to the social, technical, and legal aspects of community development. This study explored both the academic literature and practitioner accounts of FOSS community building. Based on these resources, it aimed to make informed governance decisions concerning issues such as licensing, hosting infrastructure, and initial communications. These choices are designed to contribute to the long-term sustainability of the community. The article documents these experiences and their implications.

Methodology

The study documents and analyses the lessons learned from a practical trial involving the development of an Eclipse plugin called Stylebase for Eclipse and the building of a new FOSS community to continue this development. The paper was written in a manner typical of 'experience reporting' in software engineering at the time and did not discuss methodology in detail. The work can be seen as a form of participatory observation in which the researcher fully engages in plugin development and community building. The QADA methodology (Matinlassi et al., 2002; Niemelä & Ihme, 2001) was instrumental in the technical and functional design of the plugin, as explained by Ovaska et al. (2010).

Results

The study highlighted the complexities and rewards of contributing to FOSS ecosystems, providing insights into various practical issues, including project hosting, choosing and complying with licences as well as strategies for building user and contributor bases. Developing the plugin proved relatively straightforward; however, integrating FOSS components raised technical and legal issues, necessitating careful consideration and, sometimes, the replacement of components. The report also discussed some previously unreported practical hurdles, such as the lengthy acceptance processes of FOSS hosting services at that time. Reflecting its era, the report also analysed the steps taken to avoid ideological disapproval on the part of certain FOSS communities, such as replacing Sun Java libraries and ensuring full Linux support, even though Linux users were a marginal or nonexistent user base. In addition to this article, more detailed results have been published in a research report (Henttonen, 2007).

5.2 Article II: Open Source-Based Tools for Sharing and Reuse of Software Architectural Knowledge

Objective

The article is an extension of Article I, but it shifts in focus from FOSS community building to the purpose of the software artefact itself. The article was intended to evaluate how well the Stylebase for Eclipse plugin meets its intended purpose, which is to boost the sharing and reuse of software architectural knowledge (SHARK) within FOSS projects and other geographically distributed development teams. It was believed that the effective dissemination of architectural knowledge could improve the sustainability of FOSS projects by facilitating better decision-making and integration practices. Furthermore, the long-term sustainability of the plugin community itself was naturally dependent on it creating value for users.

Methodology

The methodology employed involves constructing an evaluation framework specifically designed to assess SHARK tools. This evaluation framework was built on the NIMSAD meta-framework (Jayaratna, 1994; Simister, 1996), and its criteria were derived from the academic literature. The framework was applied to assess Style for Eclipse and two other open-source tools for the same purpose: Web of Patterns and PAKME (Process-based Architecture Knowledge Management Environment). The study utilised publicly available materials, hands-on tool installation, and usage in certain test scenarios to derive insights into each tool's effectiveness.

Results

The evaluation revealed distinct strengths and areas for improvement in 'Stylebase for Eclipse' as compared to the two other tools. The strengths of 'Stylebase for Eclipse' included its flexibility, easy integration with Eclipse tools, and illustrative presentation of design knowledge. 'Web of Patterns' excelled at facilitating the search and retrieval of design patterns directly from source code, making it highly suitable for FOSS and other agile projects. In turn, PAKME, the most comprehensive tool evaluated, offered robust functionalities for managing extensive architectural knowledge at the cost of being complex to deploy. The results pinpointed areas for further enhancement regarding the 'Stylebase for Eclipse' plugin, including usability improvements and the addition of collaborative features. They highlighted the fact that the long-term success of the 'Stylebase for Eclipse' community would be tied to its ability to deliver value through both robust features and ease of integration into developers' daily activities. Although it was not the focus of this work, the results also illustrated the potential of SHARK tools to support the sustainable governance of FOSS projects broadly by enabling efficient architectural knowledge management.

5.3 Article III: Libre Software as an Innovation Enabler in India – Experiences of a Bangalorean Software SME

Objective

The study explores the role of organisational FOSS governance in fostering innovation and economic development within a small to medium-sized enterprise (SME) in Bangalore, India. The primary objective is to assess how the SME governs its FOSS efforts to enhance innovative capabilities and value chain positioning. These issues are related to open-source governance, particularly how the SME manages its engagement with FOSS communities and integrates open-source tools into its business model to sustain competitive advantages.

Methods

The approach adopted is a primarily qualitative case study of Mahiti Infotech, a Bangalore-based company that integrates FOSS into its business model. Data were collected through semi-structured interviews with the company's directors and senior developers and supplemented by document analysis and observations of employee interactions in the workspace and on relevant IRC channels. The dataset is hereafter referred to as the 'Dataset of Article III'. The interviews were recorded, transcribed, and thematically analysed using the template analysis (King, 1998) method. The initial coding was based on concepts derived from open innovation theory (Chesbrough, 2004; West & Gallagher, 2006), but it was adjusted to accommodate the themes drawn from the data. The role of other data collection methods was primarily to corroborate or contrast the evidence gathered through interviews. In addition, the documents provided data in support of value network analysis (Allee, 2003; 2008), which was used as a complementary analytical method. This approach helped elucidate the business model by tracing monetary and non-monetary value flows.

Results

The study found that the case company had developed mature organisational FOSS governance practices and that leveraging FOSS significantly boosted its innovation capabilities and progression in the software value chain. By incorporating FOSS into products, the company cut R&D costs, hastened time-to-market, and bolstered global competitiveness. Despite resource limitations, the company prioritised contributing to FOSS communities. Being active developers, rather than mere users of FOSS solutions, improved their brand image, provided marketing benefits, and created opportunities for inter-organisational learning. The company ensured the compatibility of its business models with FOSS licenses and integrated FOSS tools and practices into internal processes, such as employee training. Challenges were also reported, including adapting FOSS solutions to local market needs and financing some FOSS efforts. In addition to this article, I published a longer and more detailed report on the results in (Henttonen, 2011).

5.4 Article IV: Managerial Perspective of Open Collaboration and Networked Innovation

Objectives

The study explored the diverse managerial views regarding FOSS usage and networked innovation across six software companies. The primary objectives were to understand the differences in managerial attitudes towards FOSS, the expected benefits, and the challenges related to its adoption and integration into business practices. Specifically, the study sought to examine how different levels of engagement with FOSS communities influence corporate governance. While the focus was on organizational/corporate FOSS governance, its influence on the sustainability of FOSS communities and ecosystems was also addressed.

Methods

The research employed a multiple case study approach, analyzing six software companies that utilize FOSS extensively but vary in their level of engagement with FLOSS communities. The method included semi-structured interviews with company managers and a review of documentation such managerial guidelines and saved chat conversations. The interviews were recorded and transcribed. Documents and interviews were thematically analysed using the template analysis (King, 1998) method. The initial coding was based on concepts adapted from open innovation theory, but was flexibly adjusted to incorporate themes drawn from the data. Interviews were the primary data collection method, and document analysis was used to corroborate or contest the evidence gathered through interviews.

Results

The study highlighted the pivotal role of managerial attitudes in shaping corporate FOSS governance. Although this was only implied in the original text, it is worth noting that these attitudes often seemed more rooted in gut feelings and ideology than in informed business analysis. The attitudes led to stark differences in how companies engaged with FOSS, consequently impacting the benefits they reaped and the challenges they encountered. Companies categorised as 'external innovators' tended to perceive FOSS primarily as a means of cutting costs and accelerating time-to-market. Conversely, 'open innovators' viewed FOSS as integral to their value creation processes, actively participating in and contributing to the community to foster networked innovation. Despite the significant time and resources required for FOSS contributions, managers from the later companies perceived their reciprocal relationships with FOSS communities as a sustainability strategy. They recognised that their contributions help sustain the ecosystems they depend on, ensuring the longevity of their business models.

5.5 Article V: Life-Cycle Management in Government-Driven Open-Source Projects – Practical Framework

Objective

The article addressed a gap in the existing literature concerning practical tools for managing the evolution of FOSS products in the public sector, emphasising the stages from deployment to end-of-life. The study was intended to develop a practical framework to support adopting a collaborative lifecycle management model in the governance of public sector FOSS communities. This framework served as a practical application of the model introduced by Kääriäinen et al. (2012).The framework was tested in a real-life case study to evaluate its effectiveness in enhancing FOSS sustainability. The focus was on determining whether and how a structured lifecycle management approach could address previously observed sustainability challenges in public sector FOSS projects, such as insufficient work coordination and poor quality.

Methodology

The research adopts an action-case approach (Vidgen & Braa 1997), combining explanatory and interventionist elements. Developing a lifecycle management framework involved iterative focus group discussions with public sector information systems experts and email interviews with SME-sized software companies familiar with public sector collaboration. Software configuration management planning (Leon, 2015) was selected as a practice-oriented method to guide framework development. The framework was then applied and tested in the Oskari project, a collaborative open-source geospatial toolkit. The assessment data were collected from participant observations and semi-structured interviews with key project contributors. Interview transcripts were analysed using the template analysis (King, 1998; 2012) method.

Results

The paper introduced a CO-SLM (Collaborative Software Lifecycle Management) framework, which provided clear guidelines for designing governance structures, responsibilities, and processes within public sector FOSS projects. Figure 1 depicts the four main elements of the product management framework. The practical framework then elaborates on each element, describing the issues that must be considered and documented when a consortium of public sector organisations seeks to co-develop a FOSS product.

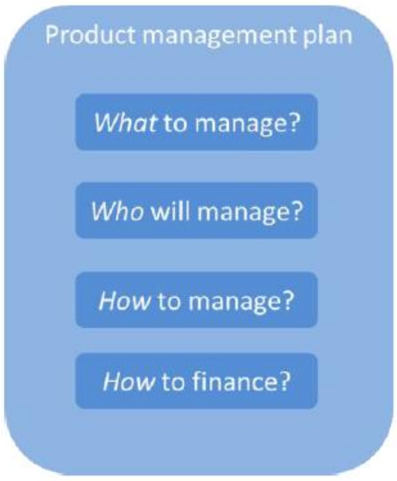


FIGURE 1 Main elements of the CO-SLM framework

The application of the CO-SLM framework in the Oskari project demonstrated several benefits, such as increased stakeholder trust in the community, increased inter-organisational collaboration and enhanced software quality (e.g., due to shared architectural principles). These contributed to the sustainability of the project. However, the study also acknowledged sustainability challenges the framework did not fully address, such as ensuring the project's independence from any single coordinating entity. Overall, the findings underscored the need for robust, community-driven governance models that can adapt to the unique challenges of open-source software development in government contexts. Apart from this article, the results were also disseminated by Matinmikko et al. (2017) in an article aimed at practitioners that emphasised actionable insights.

5.6 Article VI: Health and Orchestration of Public-Sector Open-Source Software Ecosystems: Roles, Rules, and Tools

Objectives

The study explores how public-sector organisations function as orchestrators and keystones within free and open-source software (FOSS) ecosystems, with a particular focus on governance activities that contribute to the long-term well-being of these ecosystems. To define this well-being, the research draws on the concept of 'ecosystem health'. The objectives were twofold. First, the study sought to identify the key dimensions of FOSS ecosystem health in the public sector, and second, it sought to understand how orchestrators and keystones can influence this health through governance.

Methods

The research employs a qualitative multi-case study approach, examining three FOSS ecosystems orchestrated by public sector organisations. This was not a comparative study design, so the cases were not selected for comparison but, rather, to maximise the diversity of perspectives and allow for the identification of commonalities across cases. Data were collected through interviews, participant observation, and an analysis of online discussion forums. More detailed information on the dataset, which is hereafter referred to as the 'Dataset of Article VI', is available at the Jyväskylä University Digital Repository (Henttonen, 2020; 2024). The selection of interviewees was strategically oriented toward capturing diverse perspectives within each case ecosystem to provide a balanced representation across ecosystem roles and occupational positions. The study used a combination of directed and conventional content analysis (Hsieh & Shannon, 2005) to approach the interview data. Template analysis (King, 1998, 2012; King & Brooks, 2017) was employed to analyse the transcripts thematically. In parallel to the coding of the interview transcripts, the supplementary documents, such as text taken from online discussion forums and notes from participative observation, were scrutinised for examples that corroborated or contested the evidence derived from the interviews. Reflecting our analytical focus on commonalities, the results were written in a manner that integrates the insights derived into a unified narrative rather than examining them on a case-by-case basis.

Results

This study introduced a model for health-sustaining activities in public sector FOSS ecosystems and underscored the importance of their purposeful governance. The model described seven health-sustaining activities and elucidated their relationship with ecosystem health, as illustrated by Figure 2.

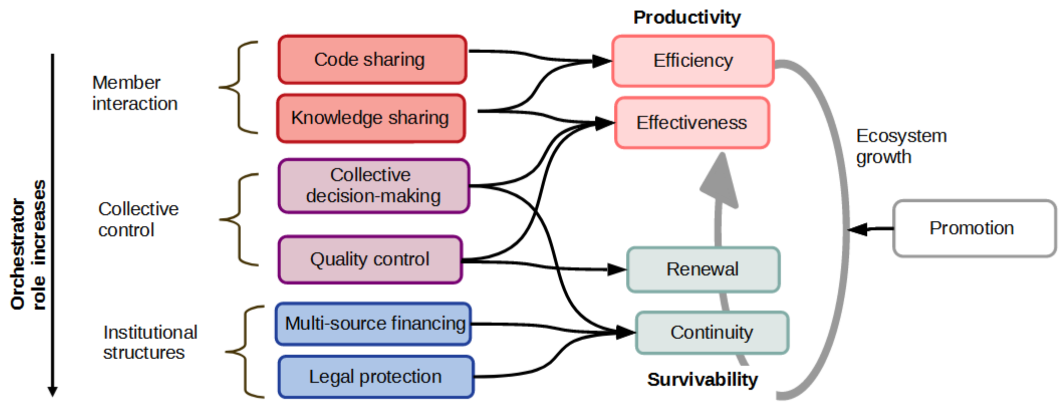


FIGURE 2 Health-sustaining activities in public sector FOSS ecosystems

The model also described how public-sector orchestrators and keystones engage in these health-sustaining activities through strategic role creation, rule establishment, and tool provision. Figure 3 illustrates this triad, which hints at the potential explanatory potential of activity theory. Through rulemaking and role creation, the orchestrator shapes organisational relationships within the ecosystem. Digital tools support both prescriptive tools, such as automatic quality assurance (QA) tools, and collaborative tools, such as wikis and CVS, which help ensure compliance with the rules, while collaborative tools, such as wikis and CVS, facilitate the division of labour.

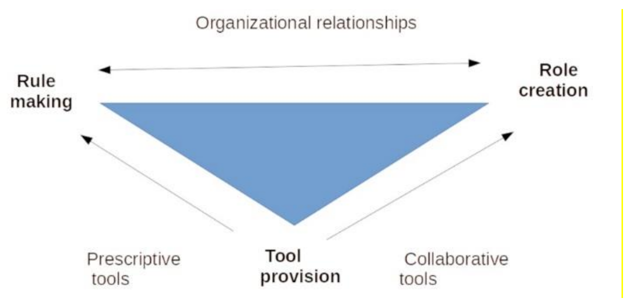


FIGURE 3 Relationships between roles, rules, and tools

The study also identified significant challenges public sector orchestrators face, such as conflicting policy frameworks and institutional misalignment with private software contractors. Challenging the myth of complete self-sufficiency, the findings emphasised that fostering a sustainable open-source ecosystem in the public sector requires substantial dedication, expertise, and investment.

6 RESULTS

This chapter synthesises the findings derived from the articles. It aims to answer the following overall research question:

RQ How do governance approaches across organizational, community, and ecosystem levels influence the sustainability of FOSS?

The chapter is organised into subsections based on the sustainability types described by Curto-Millet and Corsín Jiménez (2023) and the three levels of governance.

6.1 FOSS Governance for Infrastructural Sustainability

Infrastructural sustainability concerns the capacity to sustain technical and legal infrastructure on a long-term basis. This section discusses the findings regarding how governance at various levels can support it.

6.1.1 Community Governance and Infrastructural Sustainability

One important way governance enhances FOSS communities' sustainability is by establishing and maintaining a robust legal and technical infrastructure. Several case studies in this dissertation involved some interaction with foundations, including the Eclipse Foundation (Article I), the Plone Foundation (Articles III and VI), and the Decidim Free Software Association (Article VI). The role of foundations varies depending on the nature of the ecosystem—one could not draw many comparisons between the Eclipse Foundation of the 2000s, which was heavily dominated by IBM, and the very democratic and collectivist Decidim Foundation run by public sector organisation and civil society actors in the 2020s. What they share, however, is a key role in sustaining the legal infrastructure of the ecosystem. This was seen as crucial for the long-term sustainability of the ecosystem, as highlighted by an informant:

I would like to emphasise the Plone Foundation as a success story. We have a governance body responsible for ensuring that the wheel keeps rolling, that there is a legal basis for everything, that contributor license agreements remain in force, and so on. We also have a trademark and a logo that must be protected against misuse. This is very dull administrative work but extremely important in the long run. (Article IV, p. 13)

Of all legal agreements, licensing terms have particularly strong influences on the governance and sustainability of a project, as detailed in the literature review (2.2). The topic has surfaced—often unprompted—throughout the case studies in this dissertation. The narrative surrounding licensing has evolved markedly. Earlier conversations with informants, particularly those in Articles I and IV, were dominated by somewhat ideologically charged debates about the merits and limitations of copyleft licences and their impact on community sustainability. In contrast, the discussions in later articles adopt a more pragmatic approach, focusing on aligning licensing choices with business goals and ensuring compatibility with licenses that are widely adopted within the broader ecosystem.

The collaborative development environments have also undergone significant transformations during the dissertation period, from platforms such Savannah and SourceForge in the early 2000s, as discussed in Article I, to the contemporary predominance of GitHub, which was used in the case studies reported in Articles V and VI. However, despite changes in technology, the key tools remain similar: version control, issue tracking, and discussion forums. Additionally, the most recent article (VI) highlights the role of automatic and semi-automatic quality control tools. These critical tools include linting tools (e.g., Flake8 and RuboCop) that analyse code for potential errors and style violations and continuous integration (CI) tools (e.g., Jenkins and GitHub Actions) that automate build and test processes[7]. These tools can be seen as forms of architectural knowledge sharing (like the earlier tools reviewed in Article II) because they embody architectural principles, best practices, and patterns. However, they do not explicitly document architectural decisions. Discussion forums are often seen to suffice for this purpose in FOSS communities.

6.1.2 Ecosystem Governance and Infrastructural Sustainability

At the level of ecosystem governance, infrastructural sustainability is mainly related to ensuring technical and legal infrastructure compatibility. The license compatibility with the ecosystem was considered more important than an individual community's ideological or pragmatic preferences. One informant stated, "Right now, the most important thing is to pick the already strong licence in each ecosystem so that it is compatible with the other pieces" (dataset of Article VI). The uniformity of the tooling environment within a broader ecosystem was also discussed. For example, some informants mentioned the benefits of

[7] The tools are not mentioned by name in the articles but are present in the underlying dataset of Article IV. These and other linting and CI tools were pioneered within the FOSS scene and have since been widely adopted in proprietary development environments.

standardising QA processes by adopting uniform tools throughout the ecosystem.

6.1.3 Corporate Governance and Infrastructural Sustainability

Organisational/corporate governance strategies for legal and technical infrastructure vary widely. The results of the studies reported in Articles III and IV suggest that companies that establish strong partnerships with FOSS communities integrate their tooling environments with those of the FOSS communities. This leads to adopting FOSS-based collaborative development tools within their internal software development processes, as exemplified by the inner source platform deployed by Mahiti Infotech. Additionally, these companies adapt their business models to align with FOSS licence terms and often view 'open-source piracy'— using FOSS-licensed software in a manner not permitted by its licensing terms—as a business threat. In contrast, companies that only utilise FOSS components typically maintain their existing tools and processes, using them to assess and integrate FOSS code. They also opt for FOSS components that are compatible with their existing business models, rather than changing to accommodate licensing terms. As discussed in Article IV, these companies view copyleft licenses negatively and may be more prone to violating their terms if they believe they will not be caught.

While the above description may sound like two distinct approaches, corporate FOSS governance strategies are situated along a spectrum between these two, as highlighted by Article IV. Evidently, the sustainability of FOSS communities is influenced by how participating commercial entities adhere to the standards embodied in their legal and technical infrastructure.

6.2 FOSS Governance for Resource-Based Sustainability

Free and open-source software governance can enhance resource-based sustainability by effectively managing and nurturing all available resources within the ecosystem. This includes both input resources—mostly human but also financial—and output resources, with the source code being the most critical. Given the differences in the governance practices for input and output resources, this section is divided into two subsections accordingly. In line with the sustainability types (Curto-Millet & Corsín Jiménez, 2023), infrastructure is not treated herein as a resource but, rather, as a separate category (see Section 6.1).

6.2.1 Governing Input Resources

This section summarises governance of input resources—financial and human—from the perspectives of community, organisation, and ecosystem.

6.2.1.1 Community Governance of Input Resources

We know from the literature and practice (see Section 3.2) that large, multinational companies can maintain large FOSS communities alone if that serves their strategic interests. In this dissertation, I also report the example of a resource-constrained Indian SME kickstarting and maintaining a FOSS project alone (Article III). In this type of community, resource-based sustainability largely depends on the 'depth of the pockets' and revenue model of the leading company. However, in all other cases explored in this dissertation, the studied organisations have entered the FOSS scene with a strong desire to benefit from resource pooling. Thus, resource-based sustainability depends on mechanisms encouraging the inflow of human and financial resources from various sources.

The inflow of human resources can be sustained by promoting the growth of the user base and deepening user engagement, as discussed throughout this dissertation. At the community level, this involves attracting users and providing them with a rewarding pathway to becoming contributors. The case studies reported in Articles III, IV, and VI suggest that commercial organisations and broader ecosystems play crucial roles in expanding the user base. Community governance mechanisms, such as clear contribution guidelines, published roadmaps, and mentoring programs, are vital in encouraging and enabling independent and organisational users to become more actively involved and transition into becoming contributors.

Nevertheless, the results reported in Article IV highlight the fact that a balance must be struck regarding how much time core teams invest in accommodating new users and contributors. While the growth of communities or ecosystems is generally considered beneficial for sustainability, rapid expansion can overburden central organisations. The challenges associated with growth are particularly evident in public sector organisations but were also mentioned during interviews with companies. The results suggest that rapid growth poses challenges unless sufficient resources are available to "grow the core team proportionally to the growth of the user base", as the informant expressed it (dataset of Article VI). Managing growth is, therefore, a crucial aspect of ecosystem and community governance because it helps to ensure resource-based sustainability.

Apart from attracting human resources, community governance must also effectively manage their allocation. This includes addressing collective action dilemmas related to less glamorous tasks, such as maintaining core components. While they are crucial for everyone, these tasks often do not offer new business opportunities for companies, recognition for public sector managers, nor intellectual stimulation for individual developers. Effective governance mechanisms are essential to ensure that these critical tasks are not neglected. One potential mechanism identified in the dissertation involves collecting community membership fees to contract out unwanted tasks. This approach emerged from the public sector context, where the importance of financial inflows and formal co-financing contracts is emphasised. Co-financing contracts in public sector FOSS projects addresses legal and practical constraints, such as procurement laws and a lack of technical expertise, which limit the ability of public sector organisations to contribute directly to development work. They can also be used to assign tasks that no organisation would spontaneously undertake.

6.2.1.2 Ecosystem Governance of Input Resources

At the ecosystem level, resource-based sustainability is fundamentally linked to the resourcing of communities that develop the core platform and other foundational components of the ecosystem. The process of gradual engagement also applies at the ecosystem level. Initially, the ecosystem must grow by involving organisations in peripheral roles, such as users or developers of third-party extensions. These peripheral participants can then be offered opportunities and incentives to contribute to developing the core platform and other strategically important projects. However, managing growth is also a challenge in an ecosystemic context. This is because new extension developers often require support and attention from the core platform development community but typically do not contribute resources in the short term.

Platform development can also grapple with long-term collective action dilemmas. Investing in the core platform may appeal less to stakeholders than developing proprietary, value-adding extensions. One informant likened platform development to a "janitor's job" (Article IV, p. 6)—a task left for others to handle. Without effective ecosystem governance to incentivise and facilitate participation in platform development, too many participants might adopt a freeriding position regarding platform development.

In addition to attracting contributions to platform development, the role of ecosystem-level governance is to promote resource pooling among all ecosystem projects. One interviewee explained how this kind of resource pooling between projects works in the context of the Oskari case:

> The first level is that each organisation manages their own projects [with an ecosystem] but follows some commonly agreed-upon principles. Meanwhile, others are waiting to get their hands on it. This is the most common way because it is fast and easy if you have money. The second level involves collaborative financing; it is much more complex and requires trust. One organisation is chosen as a leader, and then, the leader organisation makes a consortium agreement with other organisations to co-finance and co-develop something together. (Article V, p. 17)

Inter-project collaboration within the ecosystem encourages consolidating efforts and resources to develop extensions, thereby preventing the proliferation of redundant extensions and enhancing overall efficiency.

6.2.1.3 Corporate Governance of Input Resources

Although governance mechanisms at the community and ecosystem levels can influence the amount and distribution of human and financial resources, ultimately, investment decisions are made at the level of corporate FOSS governance. The findings reveal stark differences in the willingness of commercial organisations to allocate resources to FOSS activities that may not directly generate revenue. Although they were perhaps less pronounced, differences were also noted in the willingness of public sector organisations to invest in long-term shared development efforts that go beyond addressing immediate administrative responsibilities.

Some private sector companies perceived FOSS contributions as giving money to a charity, that is, a benevolent but ultimately wasteful activity that a company could not afford in the long run, at least apart from minor PR investments. One interviewed software company manager answered a question about FOSS contributions as follows:

> Why would anybody contribute? […] You are not giving gold for free to anybody. It is like a joke that, you know, it is open-sourced. Everybody contributes. […] ha-hah, nobody contributes, especially [not] the big companies focusing on asset protection. (Article IV, p. 6).

An executive at another software company took a different approach, highlighting the importance of contributing to FOSS sustainability:

> If you are part of the ecosystem, you must do things to sustain that ecosystem. If you are just a consumer, then that ecosystem will sooner or later die… in order to make the open-source ecosystem stable, you [a company] have to start looking at other aspects than just being a consumer... to contribute in different ways and make sure that the ecosystem stays alive. (Article IV, p. 8).

Some findings from the case studies reported in Articles III, IV and VI suggest that the centrality of FOSS to a company's business model may partially explain some differences in corporate FOSS strategies. Logically, companies for which business depends on the continuity of a specific FOSS community or ecosystem are more likely to contribute financial or human resources. However, the findings also highlighted ideological reasons for contributing, which will be discussed later in the context of interactional sustainability.

6.2.2 Governing Output Resources

This section summarises the governance of the most important output resource, the source code, at the community, organisational, and ecosystem levels.

6.2.2.1 Community Governance of Output Resources

If a FOSS community can secure a sustained flow of financial and human resources, it sets favourable conditions for nurturing output resources, such as the source code. The importance of robust quality control was emphasised in communities that developed core platforms serving a large ecosystem (see Articles IV and VI). For platform communities, maintaining both the technical quality (e.g., ensuring the architecture remains maintainable) and the functional focus of the product is critical. Regardless of the QA methods selected, governance must establish clear guidelines and processes for quality control and communicate them effectively. The good communication also optimises the use of human resources by minimising last-minute debates over whether a contribution should be accepted, as one informant described:

> Contributor guidelines must be available, and the criteria must [be] clear about what is required from contributions. Because if […] those contributions get rejected on the last line, it is very frustrating, a complete waste of resources. […] People will stop trying to contribute if that happens again and again. (Article VI, p. 15)

The stringent QA approach also helps to maintain an approachable architecture, lowering entry barriers for new contributors and supporting human resource management. This development was seen with Plone, for example, for which a relatively relaxed approach to QA has led to an architecture that an informant compared to the "monster of Frankenstein" (dataset of Article VI), noting that such an architecture is difficult to learn, limiting the influx of new contributors.

6.2.2.2 Ecosystem Governance of Output Resources

Direct involvement in the QA processes of individual projects was often seen by informants (in the datasets of Article V and VI) as impractical and undesirable, as it would conflict with the principle of project autonomy. At the ecosystem governance level, QA work promoted integrability and interoperability between projects. The case studies in this dissertation exemplify various strategies with which to accomplish this[8], such as by endorsing open standards, as Oskari does with relevant Open Geospatial Consortium (OGC) standards, by providing extensive APIs like Decidim, and by offering common frameworks for extension developers like the Eclipse Modelling Framework (EMF).

Interoperability and integrability are essential at the ecosystem level because they make resource pooling easier between projects and increase the value of output resources. For example, a large public sector organisation in Finland contracted the development of Decidim extensions, which, due to the smooth integration with the Decidim core platform and easy installation through the Ruby Gems platform, have come to be used internationally and add value to others. This work also included integration into Finnish national systems - such as those offered by the Digital and Population Data Services Agency – which enhanced the value of Decidim to other Finnish organisations.

6.2.2.3 Corporate Governance of Output Resources

Corporate FOSS governance also plays a crucial role in determining how much attention is given to the quality of FOSS contributions. The findings from the last study (Article IV) revealed the challenges of QA in environments in which many participants make FOSS contributions merely to comply with external demands. This includes public sector organisations that, in the words of one informant, "throw stuff into GitHub" (Article VI, p. 16) merely to comply with policy recommendations or software companies that use open-source methods only because that is required by the organisations contracting the work.

If the contributing organisations have little intrinsic interest in the long-term sustainability of the ecosystem or project, this affects the quality of

[8] Although not explicitly mentioned in the included articles, the examples provided here are derived from the datasets of Article VI (Oskari and Decidim) and the experience of working with EMF while developing an Eclipse plugin, as described in Article I.

contributions. For example, new functionalities have been placed in illogical parts of the code structure, and interfaces are used in a non-standard manner. Some contributing companies were reluctant to comply with the basic architectural requirements of the core platform and insisted on making non-reusable contributions. One informant explained this situation as follows:

> We have architectural principles that should be followed in the development work; one must pay attention to generalisability in software design. However, it is practically always cheaper for a company to develop a specific solution with minimum time and effort. However, then, it cannot be reused as such. Some generalisation would be needed, which is a big part of the work. Suppose companies are not contractually obligated to do it. In that case, they will not do it because they think money and time is money for them… it annoys me when the companies say, 'Yeah, yeah, this is reusable' when it is not reusable. (dataset of Article VI; see Article IV, p. 16, for a shorter version of the citation)

In the public sector context (Articles V and VI), it was evident that some participants' lack of interest in adhering to quality principles placed significant pressure on QA processes and personnel. Resources were wasted on assessing substandard contributions and/or arguing about their rejection. Fortunately, this was not a universal issue, and some companies played key roles in maintaining the quality of the codebase. For example, some SME-sized companies in the Decidim platform community make high-quality contributions and significantly advise public sector participants on QA issues. Meanwhile, some larger companies received substantial revenues but paid little attention to quality.

6.3 FOSS Governance for Interactional Sustainability

If resource-based sustainability highlights the continued inflow and QA of contributions, interactional sustainability highlights interactions that ensure that the incoming contributions are aligned with shared values and expectations.

6.3.1 Community Governance and Interactional Sustainability

At the community level, the research done for Articles III, V, and IV strongly supports the notion that inclusive and transparent decision-making processes are pivotal in encouraging positive community dynamics. Contributions can only be aligned with community expectations if members understand how decisions are made and feel that their perspectives are valued. As one informant put it (dataset of Article VI), "You want to be treated as a partner and not as a free-or-charge resource thrown with unwanted tasks".

The importance of knowledge sharing in sustaining both input and output resources has already been touched upon (see Section 6.2). Additionally, knowledge sharing is crucial for collective decision-making. For example, one informant noted (Article VI) that the Project Steering Committee (PSC) meetings were particularly challenging because many participants lacked sufficient background knowledge to engage in the discussions. He expressed reluctance to dominate the conversation but found himself being the only one speaking, as others were there ″just to listen and learn″. It was acknowledged that participating in PSC meetings without the necessary background knowledge to discuss the matters being decided was not the most efficient way to learn. Because the problem could not be solved at the organisational or community level, those in charge of ecosystem governance planned to invest further in inter-organizational knowledge sharing, for example, by organising training events.

6.3.2 Ecosystem Governance and Interactional Sustainability

As discussed in the context of resource-based sustainability and as mentioned in several articles (III, IV, V, and VI), there are various approaches to inter-project resource pooling: some stakeholders engage in the passive reuse of source code from other projects, while others pursue opportunities for co-developing extensions together. While technical knowledge sharing (e.g., well-documented source code and interfaces) may be sufficient to enable the former, the latter model requires organisations to understand one another's requirements and goals in depth. As it typically involves the creation of new communities or merging existing ones within the ecosystem, the role of interpersonal trust and face-to-face meetings was highlighted. One informant (the dataset of Article V) argued, “If you need to put money on the table and finance something together, it does require trust, and trust does not happen unless you meet in person.”

Collaborative decision-making at the ecosystem level is often perceived as primarily focused on integration issues, such as interface standards. However, the close intertwining of ecosystem governance with the community governance of the core platform makes the picture more complex, as observed in Articles V and VI. Most of the time, the platform community governance determines which issues are important enough to escalate to ecosystem-level decision-making. Due to the highly heterogeneous and loosely coupled participants, ecosystem-wide negotiations are often particularly challenging, which may create a temptation to make decisions in a smaller circle. This interactional deficit can cause platform remakes that are not interesting for most ecosystem participants. An informant recalled a historical example:

> The remember in the 2000s when the Zope component architecture [was adopted]… In the Python ecosystem, it was the most advanced way to build applications from new components. However, because Python is mainly used for small applications, it did not interest most of the Python community, so it remained a niche area. (dataset of Article VI)

The same informant continued to explain that this major design overhaul—which remained underutilised—caused later challenges.

6.3.3 Corporate Governance and Interactional Sustainability

There are limitations to what governance at the ecosystem or community level can do to achieve interactional sustainability. Several factors independent of community or ecosystem governance influence corporate FOSS governance, leading some companies to align their contributions carefully with the expectations of the community and others to disrespect community norms, as discussed above in the context of infrastructural and resourced-based sustainability.

The findings suggest that FOSS companies often balance profit maximisation and collective FOSS community interests. Ideological factors played a role in shaping these assessments, particularly in small companies in which organisational values are tightly connected with the personal values of the entrepreneur(s). For example, when asked about the time spent engaging with the FOSS community, one informant downplayed business reasons and highlighted the ideological underpinnings of the collaborative approach:

> The more prominent software houses... they just look straight at how many billable hours there are; it is up to the consultants themselves if they are interested, but seldom are they interested in their free time... For me, it is mainly for ideological reasons: if you participate in a community, you must commit enough to understand what that community is about. The core idea of this [platform] is that it does not fragment into ten different software pieces; that is important for me in terms of the software's design… that features which should be in the core do make it there, at least on some timeline. (dataset of Article VI)

Examining all the material collected during this dissertation, primarily from Articles III–VI, reveals that ideology and values frequently arise when discussing the nature and quality of corporate interactions with FOSS. In a fully private sector context, the discussion focused on whether to contribute or interact or not. In contrast, in the public sector context (Articles III and IV), the discussion extends to the quality of interactions. This may be due to the transactional relationships that some public sector contractors have with FOSS.

The ideological orientations are also related to the overarching ethos of a company—its cultural alignment with open collaboration and sharing. As discussed in Article IV, some companies intensely focused on protecting intellectual property and revealed a culture of risk aversion and exclusivity, which stemmed from their fundamental business ideologies. These companies typically exhibit significant concerns about participating in FOSS forums, citing potentially exaggerated risks. For example, one informant stated (Article IV, p. 140): "If we started hanging out on open-source forums, they [customers] may think that we will tell their secrets to the world... [ ] it would cast a shadow of doubt". In contrast, companies that embodied FOSS principles in their business models demonstrated a culture and ideology that value openness and inter-organisational collaboration, viewing knowledge sharing as a strategic benefit.

6.4 Summary of FOSS Governance and Sustainability

This section presents a summary of the results. Subsection 6.4.1 examines how various levels of governance contribute to sustainability, highlighting the synergistic and conflicting dynamics between them. Subsection 6.4.2 examines the role of each governance level in contributing to a FOSS community's virtuous growth cycle. Lastly, subsection 6.4.3 notes some differences between the public and private sectors regarding FOSS governance.

6.4.1 Levels of FOSS Governance and Their Inter-dynamics

Table 4 summarises the role of governance in supporting FOSS and contextualises it within sustainability types. It illustrates how ecosystem, community, and corporate FOSS governance enhance interactional, resource-based, and infrastructural sustainability.

TABLE 4 FOSS governance contributions to each sustainability type

Sustainability Type	**FOSS Ecosystem Governance**	**FOSS Community Governance**	**Corporate FOSS Governance**
Infrastructural sustainability	Promotes licence compatibility and uniform tooling infrastructure (Articles I & VI)	Maintains robust digital and legal infrastructure (Articles I, II, V, & VI)	Ensures licence compliance and integration with own tooling environments (Articles III & IV)
Resource-based sustainability - input	Facilitates project discovery, promotes resource pooling between projects, and attracts platform contributions (Articles V & VI)	Boost (social) media visibility, encourages and rewards engagement, provides clear pathways to new contributors and sponsors (Articles I, III, V & VI)	Promotes growth through marketing, invests financial and human resources, ensures sustainable revenue streams (Articles III, IV & VI)
Resource-based sustainability - output	Advocates open standards and project interoperability (Datasets of Articles I & VI)	Implements QA guidelines and checks, co-ordinates development work (Dataset of Article I, Articles V & VI)	Integrates with a company's own software products and QA processes (Articles III & IV)
Interactional sustainability	Enhances knowledge-sharing opportunities across projects (Articles III, Article V & VI)	Enables collective decision-making, encourages knowledge sharing, builds reciprocal culture (Articles III, V, & VI)	Aligns with corporate culture and values (Articles III & IV)

The interaction between different governance levels reveals a complex web of synergies and conflicts. Resource pooling, both among communities within the ecosystem and between stakeholders within a project, was often seen by informants as the most significant synergistic benefit of FOSS collaboration. However, the results show that input resources are also a key source of conflict between governance levels. Organisations often face financial and human capital constraints and are often tempted to channel these resources towards internal software development projects, rather than collaborative efforts in the FOSS community. Similarly, FOSS communities, which operate with limited resources, may prioritise investing in value-adding community products rather than contributing to ecosystem platforms or other shared software assets. This competition for input resources can lead to deterioration in the quality and value of shared software assets (i.e., the output resources).

Governance between ecosystems and communities tends to be more synergistic in interactional and infrastructural sustainability matters. However, in governance between organisations and communities, conflicts over resource-based sustainability are often intertwined with those over interactional and infrastructural sustainability. Stark value differences exist in terms of how FOSS-involved organisations perceive the importance of contributing to shared resources. These value differences were also reflected in debates over licencing choices. Although the debates on licensing issues have diminished somewhat over the years, licenses still represent different value frameworks and mandate distinct levels of reciprocity. Moreover, differences in the attitudes towards contributing were also related to infrastructural sustainability. While non-contributing organisations had equal access to the source code, they may not have had equal access to the knowledge and tools needed to ensure integration and interoperability. Poor integration can further decrease the value of input resources throughout the ecosystem.

6.4.2 Governance Supporting a Positive Feedback Loop

The interplay between governance levels could be further summarised by considering their role in a positive feedback loop, a fundamental concept in FOSS research (see Section 3.4). Informants described this idea, which is very familiar to practitioners, as a "virtuous cycle" or "community-driven growth" (datasets of Article III and VI). **Figure 4** illustrates and extends the loop to demonstrate how governance levels reinforce and shape one another, emphasising how community, corporate, and ecosystem governance is integral to each phase.

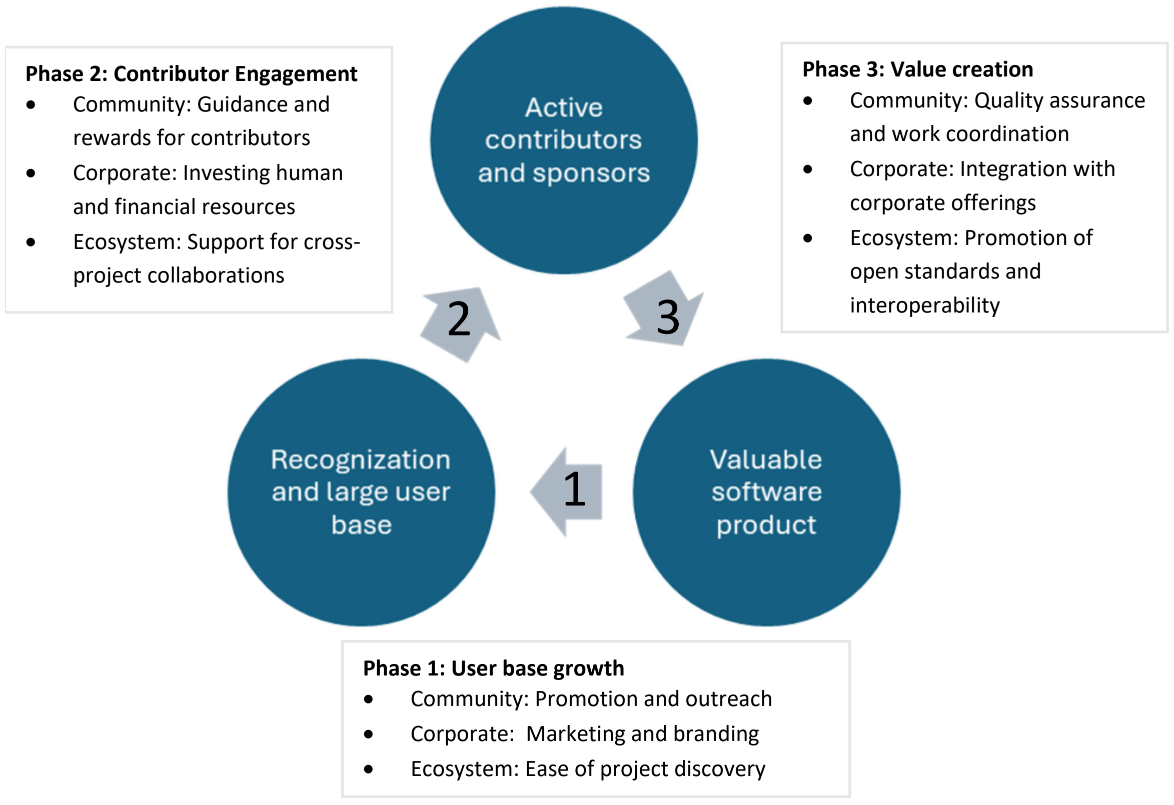

Phase 2: Contributor Engagement
Community: Guidance and rewards for contributors
Corporate: Investing human and financial resources
Ecosystem: Support for cross-project collaborations
Active contributors and sponsors
Phase 3: Value creation
Community: Quality assurance and work coordination
Corporate: Integration with corporate offerings
Ecosystem: Promotion of open standards and interoperability
2
3
Recognization and large user base
1
Valuable software product
Phase 1: User base growth
Community: Promotion and outreach
Corporate: Marketing and branding
Ecosystem: Ease of project discovery

FIGURE 4 Virtuous growth cycle and FOSS governance levels

The first phase involves attracting a user base to a valuable software product. This growth rarely happens solely due to technical merit, as discussed in Articles III and VI, although it is possible. Community governance may involve promotional activities, such as social media engagement, though their importance has been debated. Stakeholder companies contribute through direct marketing, while ecosystem-level governance enhances a project's discoverability by providing the necessary infrastructure. While not directly sustaining the project, these actions set the stage for future resource-based sustainability, especially if the growth is well managed.

The second phase involves converting a large user base into a robust pool of contributors and sponsors. In this phase, community governance is critical in creating an environment that encourages user contributions and equips users with the tools and knowledge they need to contribute effectively. This involves establishing clear contribution pathways, fostering knowledge-sharing mechanisms, and implementing systems that reward contributions. At the ecosystem level, governance must facilitate collaboration across projects, enabling contributors to engage with multiple initiatives, thereby broadening their impact. Corporate governance is also crucial in this phase, as it allocates resources and expertise to shared projects and aligns corporate contributions with collective goals. These activities primarily support interactional sustainability and the input aspect of resource-based sustainability.

The third phase of the cycle focuses on transforming contributions and sponsorships into an increasingly valuable product. This stage highlights the importance of governance in resource allocation, work coordination, and QA at the community level. Ecosystem governance enhances this process by promoting interoperability and integration with other software components, amplifying the product's value. Corporate governance further reinforces this transition by aligning the project with business models, ensuring that the software's value can be augmented through service provision, support, and other value-added offerings. These activities primarily address the output aspect of resource-based sustainability, while infrastructural sustainability is a critical enabler.

6.4.3 Differences Between Public and Private Sector FOSS Governance

The results have also highlighted key differences in the governance of FOSS in public and private sector contexts. At the level of ecosystem governance, securing financing for a fast-growing community is particularly challenging in the public sector due to budgetary constraints and bureaucratic hurdles. Unlike the private sector, where rapid growth is more unambiguously seen as a positive indicator of success, public sector FOSS communities risk fragmentation and declining quality if growth is not carefully managed. This influences resource-based sustainability, especially the quality of output resources.

Public procurement laws pose unique challenges at the community and ecosystem governance levels. Rigid procurement processes can lead to a transactional atmosphere, unlike the relational and trust-based dynamics often found in commercial or volunteer-based FOSS projects. Furthermore, as these laws limit the ability to choose partners freely, they can make it harder to reward reciprocity and goodwill.

At the level of organisational governance, stark differences between the public and private sectors were not as apparent, although the data on this topic was limited. Across both the public and private sectors, there is considerable variation in the commitment of individual organisations to collaboration within FOSS projects. Much like private corporations, public sector organisations display differing levels of willingness and ability to contribute to shared initiatives, influencing resource-based sustainability.

7 DISCUSSION

This section discusses the research process and results. Section 7.1 discusses the governance levels and their inter-dynamics in the context of prior studies. Section 7.2 reflects on the types of sustainability and their applicability to empirical studies. Section 7.3 discusses the methodological choices and quality criteria.

7.1 FOSS Governance Levels

Overall, the results have underscored the importance of purpose-driven leadership and coordination for the sustainability of FOSS initiatives. Some researchers, potentially influenced by utopian underpinnings, as Carillo and Bernard (2015) suggest, have overly emphasised universal openness, spontaneous emergence, and self-sustainability in open-source software development. However, the broader body of empirical evidence (Carillo & Bernard, 2015; Noni et al., 2011; Schreiber, 2023; Shaikh & Henfridsson, 2017) highlights the necessity of targeted leadership efforts to initiate and sustain these communities and ecosystems. Nevertheless, this does not diminish the significance of broad and organic participation. My observations at both the ecosystem and organisational levels of FOSS governance indicate that compared to autocratic models, governance structures that emphasise broad participation and democratic decision-making demand more significant effort. This additional effort manifests in the need for more complex governance systems and greater time investment. These observations are consistent with earlier research on FOSS governance, highlighting the critical role of governance frameworks in enabling and managing widespread participation within FOSS communities (e.g., Di Tullio & Staples, 2013; O'Mahony, 2007).

As reviewed in Chapter 3, the literature on FOSS governance models has evolved significantly, progressing from recognising forms of control (e.g. Ljungberg, 2000; Markus et al., 2000) to analysing multidimensional governance configurations (Di Tullio & Staples, 2013; Markus, 2007; Noni et al., 2011) and towards an even more nuanced understanding of governance diversity (e.g., O'Mahony & Karp, 2022; Shaikh & Henfridsson, 2017; Wang et al., 2023). However, as discussed in Section 3.5, much of this research remains fragmented, often examining organisational, community, and ecosystem governance practices in isolation. Moreover, as noted by Franco-Bedoya et al. (2017), the term 'ecosystem' is often used without a shared definition and sometimes interchangeably with 'community', adding to the complexity of the discourse. This dissertation has contributed to the ongoing discussion by conceptualising FOSS governance as a complex, interwoven framework of activities across these three levels, emphasising the interactions between them and their combined impact on the sustainability of FOSS initiatives.

While analytically distinct, it is essential to note that the three levels frequently overlap in practice. For example, our findings demonstrate that the governance of core platform communities often intertwines with the broader governance of the entire ecosystem. The fluid boundaries between the governance levels have been noted in prior studies, which describe how corporate dominance can blur the lines between organisational and community governance (e.g., O'Mahony, 2007; O'Mahony & Bechky, 2008; O'Neil et al., 2021). Moreover, large FOSS communities may be governed in an ecosystem-like manner, with autonomous subprojects, a practice Shaikh and Henfridsson (2017) refer to as 'federative self-governance'. The case study material also points out that further fluidity can arise from the dynamic nature of ecosystems: a niche extension at the periphery of the ecosystem may quickly become part of the core platform, but de facto governance often lags behind formal governance changes. The three levels remain a valuable analytical tool despite these overlaps and blurred boundaries.

The results section (see Section 6.4.2) emphasised the interplay between levels of governance and their collective role in driving a virtuous growth cycle. This concept, which was initially popularised by Raymond (1997) and later formalised (see Section 3.4), is crucial to FOSS sustainability studies, though often more implied than explicitly discussed. The first phase of this loop—user base growth—has received little attention in FOSS governance research, which typically focuses on external factors such as market dynamics or technology policy (see Sánchez et al. 2020 for a review). The second and third phases revolve around FOSS governance challenges. Markus (2007) identified the 'social action dilemma' (how to motivate contributions) and the 'work coordination dilemma' (how to organise and manage collective work efficiently). Both will be briefly discussed below.

The results (see Section 6.4.1) show that the primary points of conflict between organisational, community, and ecosystem governance stem from the inherent tension between collaboration and competition for resources, particularly the time and expertise of skilled developers. This tension is central to the 'social action dilemma'—the challenge of motivating stakeholders to contribute to a collective effort despite competing demands on their time—and remains one of the most studied themes in FOSS research (see Section 3.1.3.1).

The results reaffirm the finding from prior research that the 'underproduction' of shared components (Amherst et al., 2007; English & Schweik, 2007) constantly threatens FOSS sustainability, and addressing this challenge is a crucial function of governance at all levels. They also highlight the fact that some trade-offs between organisational and community-level governance appear to be inevitable because self-maximising behaviour by individual participants—whether individuals or organisations—rarely leads to optimal sustainability outcomes for collective FOSS efforts. This observation is consistent with findings from common pool resource management studies (e.g., Ostrom, 1999), which emphasise the challenges of balancing individual incentives with collective sustainability.

Within this complex interplay, an exciting nuance emerged: ideologically inclined motivation's notable influence on FOSS organisational participation. While ideologies have been widely recognised as motivational factors for individual contributors (Bonaccorsi & Rossi, 2005; Stewart & Gosain, 2006; Von Krogh et al., 2012), they have been less prominent in organisational FOSS research. Interestingly, despite not being the primary focus, ideological differences emerged as a significant theme, particularly in Articles III, IV, and VI. It was observed that some SMEs approached FOSS as a shared resource requiring careful stewardship, whereas others treated it purely transactionally, aiming to maximise benefits while minimising contributions (see Section 6.2.1.3). These differing mindsets were poorly explained by market position and the other 'rational' factors typically emphasised in traditional economic theory but often seemed to stem from the world view of the company owners and key managers. Thus, ideological conflicts existed not only across the volunteer–company divide, as traditionally suggested in FOSS research (Bonaccorsi & Rossi, 2005; O'Mahony, 2007; Riehle et al., 2014), but also between companies. An improved understanding of this divide could help enhance the sustainability of FOSS. As suggested in Article IV, institutional logic (Thornton & Ocasio, 2008) may provide a valuable framework by exploring the underlying norms that guide each company's FOSS participation. Studies in this field (e.g., Khan et al., 2018) have noted that not all companies operate solely on a commercial logic.

This dissertation has also highlighted the vast array of work coordination mechanisms in FOSS governance. Shaik and Henriksson's (2017) insightful examination of the Linux Kernel community and its coordination activities related to version control identified four essential coordination practices within that context. However, when we broaden our perspective beyond community-level version control, we encounter an even more diverse spectrum of coordination mechanisms operating across different levels. Given the variety of these mechanisms, organising them along new dimensions may be more analytically fruitful than merely listing or classifying them.

Building on the categories emerging from Article VI, one could organise these mechanisms into a four-field along a spectrum from emergent to assigned roles and another spectrum from suggestive to prescriptive rules. This speculative framework is depicted in Figure 5. It organises coordination mechanisms into four distinct quadrants: ‘Commanding’, in which roles and rules are highly assigned and prescriptive; ‘Legislating’, which involves predefined rules but allows emergent role-taking; ‘Facilitating’, which is characterised by emergent roles and suggestive guidelines; and ‘Delegating’, in which authority is delegated to assigned actors who operate with independence. Although further research is needed, the ecosystem-level coordination mechanism may predominantly operate within the facilitating quadrant, which is characterised by suggestive rules and emergent roles. In contrast, community-level coordination may span multiple quadrants, reflecting its more varied nature, while work coordination inside a single organisation typically involves the quadrants associated with assigned roles.

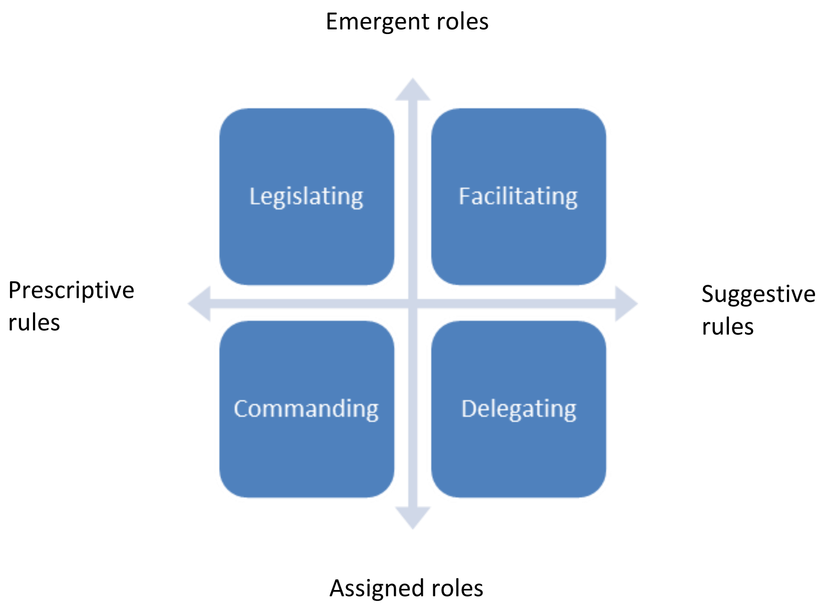


FIGURE 5 Matrix of FOSS Co-ordination Mechanisms

Two of the four coordination activities identified by Shaik and Henriksson (2017), Autocratic Clearing and Oligarchic Recursion, could be easily situated in the Commanding quadrant of the matrix. Meritocratic Idea Testing would fall within the Facilitating quadrant, while federated self-governance falls within the Delegating quadrant. More coordination mechanisms are mentioned in Article IV and the related dataset that informed the research. For example, mechanisms in the Legislating quadrant include deliberation and voting procedures based on predefined rules, as seen in platforms such as Decidim and Oskari, and quality control methods in which all contributions that adhere to specific rules are accepted, as in Plone. Additionally, Oskari’s practice of delegating social media activism to outsourced professionals could be an example within the ‘Delegating’ quadrant. These examples illustrate the diversity of coordination mechanisms within FOSS governance, underscoring the need for a nuanced understanding of how these processes interact and evolve across governance levels and contexts.

7.2 FOSS Sustainability Types

This study extends the understanding of sustainability in FOSS governance by empirically applying the sustainability types introduced by Curto-Millet and Jimenez (2023). While these types were initially conceptualised as theoretical constructs, this research demonstrates their practical utility and confirms their relevance across community, organisational, and ecosystem governance levels. Minor challenges arose when applying these sustainability types to governance issues, particularly the need to split resource-based sustainability into two subcategories, as governing input resources, such as funding and manpower, differ significantly from governing output resources, such as source code.

Synergies among the sustainability types were easy to identify. For example, infrastructural sustainability was identified as a foundational element, supporting and enabling human and software sustainability. When analysed through the lens of practical governance tasks, these synergies sometimes appear more as overlaps and boundaries between various types of sustainability, which can be rather fluid. Positive interactional dynamics, for example, are crucial in sustaining human resources and source code, making it challenging to draw clear distinctions between resource-based and interactional sustainability.

Interestingly, the findings indicate minimal trade-offs between the sustainability types themselves. Instead, the more significant trade-offs and conflicts arise between actors and between levels of governance. While the theoretical discussion (Curto-Millet & Jimenez, 2023) has suggested potential trade-offs, such as a single actor's dominance enhancing resource sustainability at the expense of interactional sustainability, the case studies in this dissertation present this issue differently. For example, reliance on a single financing organisation undermined resource-based and interactional sustainability in the long term, highlighting a strong connection between sustainability types. Overall, the results suggest that the relationships between the different types of sustainability in FOSS governance may be predominantly synergistic. The lack of practical examples involving significant trade-offs between sustainability types underscores this point. However, this observation should not be considered conclusive, as trade-offs could arise under conditions that are not present in the cases examined in this dissertation.

While the sustainability topology created by Curto-Millet and Corsín Jiménez (2023) offers a new conceptual framework, its alignment with conventional FOSS approaches is expected given that it is based on a literature review. While broader sustainability frameworks, such as the triple bottom line (Elkington, 1998), have long been embraced in IS research, they have yet to be integrated into the discourse on FOSS sustainability, as observed by Curto-Millet and Jiménez (2023). It is possible for a FOSS project to have excellent infrastructure, resource management, and communication but still be unsustainable in terms of environmental and social impact. For example, the extensive energy consumption associated with Bitcoin mining has raised ecological concerns (Yang, 2022), while the censorship practices on the Chinese FOSS platform Gitee have implications for fundamental rights (Yang, 2022). While expanding the definition of sustainability within the FOSS context is important, this study did not undertake that task and, instead, built on existing definitions.

7.3 Limitations and Methodological Reflections

This dissertation was shaped by an evolutionary study design, rather than a pre-planned structure. This approach, while organic and responsive to emerging needs, led to certain critical elements, such as open standards within FOSS governance, being partially overlooked in the scope of the study. Additionally, the selection of case studies, though diverse, exhibited some imbalances. For example, studies primarily focused on community and ecosystem perspectives (see Table 1 in Section 1.2) were conducted in the public or non-profit sectors and within strongly platform-centric ecosystems. This influenced the findings; for example, the emphasis on centralised quality control likely stemmed from the specific characteristics of these ecosystems. Due to these factors resulting from the organic study design, constructing a fully comprehensive theoretical framework for FOSS governance and sustainability was not feasible. Nevertheless, the research generated an in-depth understanding of FOSS governance levels and their inter-dynamics.

Although the case selection was less than optimal, the case study methodology proved largely effective overall. My 'triple role' within the FOSS scene—as a hobbyist, professional, and researcher—provided me with access to various organisations and facilitated a strong rapport with informants. This multi-faceted involvement allowed for extensive and rich data collection over prolonged periods, not only through interviews but also through document analysis and participant observations. The triangulation of multiple data sources within each case contributed to the credibility of the findings. The necessity to consolidate insights from sometimes conflicting sources contributed to a deep, nuanced analysis. Critically examining the reasons for differing viewpoints and acknowledging that informants may have agendas that are not entirely aligned with the research objectives was an essential part of this process.

On the other hand, the vast data collected in most case studies, coupled with the broad nature of the research questions and my relative inexperience with open coding, occasionally led to a lack of focused exploration. Many dimensions relevant to the research questions were touched upon only superficially, rather than being explored in significant depth. This issue was compounded by the constraints of standard article lengths in the IS field, which often limit the space available for reporting qualitative research. Consequently, substantial reductions in article length were frequently necessary, contributing to a perceived lack of depth in the findings. In hindsight, the use of more narrowly defined research questions in some articles could have resulted in a more in-depth inquiry.

However, these broad research questions and the use of open coding also provided notable advantages. They enabled a holistic understanding of specific subtopics and facilitated the integration of multiple disciplines—legal, economic, and technical—that are critical in the study of FOSS governance. Moreover, open coding allowed for the emergence of unexpected themes and findings from the data. For example, in Article IV, relationships between private and public sector actors became a prominent topic of discussion, even though this was not an original focus of the research. The flexibility of the research approach permitted the incorporation of these insights, which ultimately emerged as some of the most valuable findings of the study. Often, the most insightful revelations are those that informants share unprompted.

Throughout the research process, I also periodically questioned the suitability of my realist, albeit critical-realist, approach. In some of the case studies (e.g., Article IV), the subjective attitudes and ideologies of interviewees came to the forefront, and finding a logical structural explanation for these perspectives was not feasible with the available information, perhaps not even sensible. Reflecting on this, I believe that an interpretivist approach might have led to a more nuanced analysis and yielded more valuable results during some parts of the dissertation work. On the other hand, the realist approach facilitated the development of practical tools and frameworks, which were successfully implemented in real organisations (as demonstrated, e.g., in Article V).

In keeping with the critical realist approach, my dissertation adopts a writing style that limits the extent of self-reflection. However, this does not mean that self-reflective practice was absent from the research process. Especially in cases in which I was deeply involved in a professional capacity, maintaining a research-oriented mindset was crucial. This required considerable effort to critically analyse and manage the biases inherent in my specific roles within these organisations. It was essential to ensure that perspectives resonating with my own experiences did not receive undue emphasis or overshadow the equally important viewpoints of individuals with different roles. Achieving this balance was vital to the integrity of the findings, enabling a comprehensive understanding of each case.

8 CONCLUSION

This section summarises the contributions for both researchers and practitioners and then discusses potential future research directions. In terms of contributions to researchers, the focus is on the integrated insights, while the section on practitioner contributions mentions each article separately. This approach is taken because practitioner insights are context specific, with each article addressing certain scenarios and environments.

8.1 Contributions to Researchers

This dissertation advances our understanding of FOSS governance by providing a comprehensive analysis of how governance mechanisms, at the ecosystem, community, and corporate levels, contribute to the sustainability of FOSS projects. Through a detailed examination of various governance approaches and their interactions, this research project offers the following contributions to the field.

First, while the dissertation does not develop a complete framework for governance across all levels, it provides inroads that facilitate the further exploration of the roles of and conflicts and synergies between these levels. By emphasising the interplay between ecosystem, community, and corporate governance, this study reveals how these levels interact to create synergies and conflicts that significantly impact the sustainability of FOSS projects. This approach addresses a critical gap in the literature (see section 3.5), in which governance practices are often studied in isolation, providing a more holistic view that underscores the complexity of sustaining FOSS initiatives.

Second, the research contributes to the discourse on sustainability in FOSS (see section 3.4) by empirically applying and validating the sustainability types proposed by Curto-Millet and Jimenez (2023). The findings demonstrate the practical utility of these types—interactional, resource-based, and infrastructural sustainability—across governance levels, showing how they are interconnected and mutually reinforcing. The study notes that while synergies between these sustainability types are common, examples of trade-offs are difficult to find. More significant trade-offs occur between actors and between governance levels than between the sustainability types. This insight adds nuance to the existing theoretical discussions of sustainability in open-source ecosystems.

Third, the study explores the often-overlooked differences (see section 3.2) between the public and private sectors in FOSS governance. Identifying distinct challenges and dynamics in public sector governance, such as the impact of budgetary constraints and procurement laws on resource-based and interactional sustainability, provides a critical perspective that broadens the understanding of how various institutional contexts influence FOSS governance and sustainability. The most recent article (VI) introduces a model for sustaining public-sector FOSS ecosystems, detailing key activities and the critical roles of orchestrators and keystones in driving those activities.

Fourth, this research sheds light on the ideological differences between companies involved in FOSS. While ideological motivations have been widely recognised at the individual contributor level (see section 3.1.3.1), this study identifies similar ideological divides at the organisational level, particularly in terms of how companies approach collaboration and resource stewardship within FOSS projects. Because FOSS communities and ecosystems often lack mechanisms via which to enforce reciprocity, these ideological commitments are crucial to sustainability. While this issue is particularly pronounced in the public sector, it is also evident across the broader FOSS landscape. By uncovering these ideological dynamics, this research provides valuable insights into how organisational values influence corporate FOSS governance and shape the long-term sustainability of FOSS.

8.2 Contributions to Practitioners

For practitioners, this study emphasises the critical role of governance in ensuring the sustainability of FOSS, highlighting the need for adequate resourcing and strategic planning. It identifies the key governance mechanisms and pivotal dilemmas faced by FOSS-producing organisations, communities, and ecosystems. This is particularly important given the historical challenges involved in understanding FOSS governance, perhaps reflecting the persistent myth that FOSS projects and ecosystems are self-sustaining.

The most pertinent insights for practitioners are context-specific and detailed in the overview of the articles (see Chapter 5). Article I outlines the social, technical, and legal tasks involved in launching a new FOSS community within a large ecosystem and shares practical experiences related to each aspect of this process. Article II offers guidelines for selecting an architectural knowledge-sharing tool that benefits FOSS or any virtual development team. Article III showcases a practical example of how a resource-constrained SME leveraged FOSS governance decisions to ascend the value chain. Article IV examines the advantages and disadvantages of managerial approaches to FOSS, providing entrepreneurs with insights that can be used to weigh the pros and cons of these approaches and position themselves strategically. Article V introduces a framework designed to assist a consortium of public sector organisations in the collaborative governance of FOSS communities and shares the experiences of its application in a real-world organisation. Extending these insights, Article VI provides a model and a comprehensive description of the responsibilities of public sector organisations acting as orchestrators and keystones in FOSS ecosystem governance. Collectively, these articles equip practitioners with tools to navigate the complexities of FOSS governance across different contexts, ultimately contributing to the sustainability of FOSS.

8.3 Future Research Directions

This dissertation has laid the necessary groundwork for understanding the governance and sustainability of FOSS across the organisational, community, and ecosystem levels. However, several essential areas remain underexplored, as discussed previously. One promising direction for future research is developing a comprehensive governance framework that integrates practices across all levels—organisational, community, and ecosystem. While this study provided valuable insights into how these levels interact, a systematic literature review (SLR) could further consolidate existing knowledge and identify gaps. Following this, empirical validation through practitioner interviews or case studies would be essential to ensure the framework's relevance and effectiveness in diverse real-world settings.

Another important avenue for future research involves expanding the definition of sustainability within the FOSS context. The current study is built on the traditional FOSS literature, with a narrow, intra-organisational definition of sustainability. However, broader frameworks, such as the 'triple bottom line', which includes economic, environmental, and social dimensions, would offer a more comprehensive understanding of sustainability. Future research could explore how these broader sustainability considerations apply to FOSS projects, addressing apparent gaps in the current approach. For example, while a FOSS project may be well-managed regarding resources and infrastructure, it may still raise concerns regarding environmental impact or social equity. Expanding the sustainability discourse to include these dimensions could provide a more complete picture of what it means for a FOSS project to be genuinely sustainable. One potential path for future research is conducting case studies to examine how different FOSS projects address these broader sustainability challenges.

Furthermore, this research has highlighted the need to explore the role of ideological motivations on organisational participation in FOSS. Small and medium-sized enterprises often play a crucial role in sustaining FOSS projects because these enterprises are driven by a strong ideological commitment to open-source principles. However, the way companies engage with FOSS varies significantly, with some viewing it as a collaborative endeavour and others taking a more transactional approach. Future studies could apply institutional theory to better understand how these underlying values and organisational cultures shape FOSS participation. By examining the norms and beliefs that drive different types of engagement, research can provide deeper insights into how these factors influence the sustainability of FOSS ecosystems. This line of inquiry could also explore how large organisations might support or collaborate with SMEs to enhance the overall sustainability of FOSS initiatives.

An additional valuable area for future research is the role of governance at the community, corporate, and ecosystem levels in boosting the growth of the FOSS user base. In studies on FOSS growth, the role of governance has received less attention compared to external factors like market dynamics and technology policy. In my research, many informants considered promotional and marketing activities to be very critical during this phase, although not all of them shared this notion. Investigating how governance strategies across these three levels influence these activities and their impact on the long-term sustainability of FOSS projects could provide essential insights into how communities attract and retain users, ultimately enhancing the overall sustainability of FOSS initiatives.

YHTEENVETO (SUMMARY IN FINNISH)

Tämä väitöskirja tutkii vapaiden ja avoimen lähdekoodin ohjelmistojen (VALO) kestävyyttä ja hallintoa kolmella eri tasolla: yhteisö-, organisaatio- ja ekosysteemitasolla. Tutkimuksen tavoitteena oli ymmärtää, miten nämä hallintotasot vuorovaikuttavat keskenään ja miten ne yhdessä edistävät VALO-projektien pitkäaikaista kestävyyttä. Työ pohjautuu tapaustutkimuksiin, jotka on toteutettu avoimen lähdekoodin ohjelmistoja tuottavissa yhteisöissä, organisaatioissa ja ekosysteemeissä. Niissä on hyödynnetty monipuolisia tiedonkeruutapoja, kuten haastatteluita, dokumenttianalyysia ja havainnointia.

Tämän tutkimuksen yhteydessä hallinto määriteltiin laajasti, kattaen sekä strategisen päätöksenteon että jokapäiväisen toiminnanohjauksen. Yhteisöhallinto viittaa toimijoiden muodostaman VALO-yhteisön hallintoon yksittäisissä projekteissa. Organisaatiohallinto keskittyy siihen, miten yksittäiset organisaatiot hallinnoivat omaa osallistumistaan VALO-projekteihin. Ekosysteemihallinto puolestaan ohjaa laajempaa organisaatioiden ja VALO-projektien joukkoa, jotka toimivat yhteisessä teknologiaympäristössä.

Kestävyys sen sijaan on määritelty tässä yhteydessä melko suppeasti, tarkoittaen kykyä jatkaa tuottavaa toimintaa pitkällä aikavälillä. VALO-kestävyys on jaettu Curto-Milletin ja Jiménezin (2023) esittämällä tavalla kolmeen eri tyyppiin: resurssipohjaiseen, vuorovaikutukselliseen ja infrastruktuuriseen kestävyyteen. Resurssipohjainen kestävyys viittaa kykyyn hankkia ja hallita erilaisia resursseja, kuten ohjelmisto-omaisuutta, henkilöstöä ja taloudellisia varoja. Vuorovaikutuksellinen kestävyys tarkoittaa positiivisen ja tehokkaan yhteisödynamiikan ylläpitämistä. Infrastruktuurinen kestävyys viittaa kykyyn ylläpitää järjestelmiä, jotka tarjoavat tukea ja vakautta päätoiminnoille kuten työkaluympäristöjä ja oikeudellisia kehyksiä.

Tutkimus korostaa hallinnon roolia ja alleviivaa, että VALO-yhteisöt ja ekosysteemit eivät ole täysin itseohjautuvia, vaan niiden pitkäaikainen kestävyys edellyttää tarkoituksellista ohjausta ja koordinointia. Tutkimuksessa tarkastellaan, miten hallinnolliset toimet eri tasoilla voivat edistää kutakin kestävyystyyppiä. Taulukko 5 tiivistää tulokset tältä osin.

TABLE 5 VALO-hallinnon vaikutukset kestävyyden eri tyyppeihin

Kestävyyden tyypit	**Ekosysteemihallinto**	**Yhteisöhallinto**	**Organisaatiohallinto**
Vuorovaikutuksellinen	Edistää tiedonjakamista projektien välillä	Mahdollistaa yhteisöllisen päätöksenteon, edistää tiedon jakamista, rakentaa vastavuoroista kulttuuria	Sovittaa yhteen yrityksen ja yhteisön kulttuurin ja arvot
Resurssipohjainen - Panokset (työvoima ja raha)	Tarjoaa yhteisöille näkyvyyttä, edistää resurssien jakamista projektien välillä ja hankkii resursseja yhteiseen kehitystyöhön	Edistää ja palkitsee käyttäjien osallistumista, tarjoaa selkeitä polkuja uusille osallistujille ja sponsoreille	Edistää kasvua markkinoinnin kautta, investoi taloudellisiin ja henkilöstöresursseihin, varmistaa kestävät tulonlähteet
Resurssipohjainen - Tuotokset (ohjelmakoodi)	Kannustaa avoimiin standardeihin ja projektien yhteentoimivuuteen	Toteuttaa laadunvarmistusta ja tukee järkevää työnjakoa	Integroi omiin ohjelmistotuotteisiin ja tukee laadunvarmistusprosesseja
Infrastruktuurinen	Edistää lisenssien yhteensopivuutta ja yhtenäisiä työkaluympäristöjä	Ylläpitää digitaalisia työkaluympäristöjä ja oikeudellisia kehyksiä	Varmistaa lisenssien noudattamisen ja integroi työkaluympäristöt

Tutkimuksessa havaittiin, että VALO-hallinnon eri tasot voivat vahvasti tukea toinen toisiaan, mutta niiden välillä esiintyy myös konflikteja. Ekosysteemien ja yhteisöjen välillä vallitsee usein synergistinen suhde erityisesti vuorovaikutuksellisen ja infrastruktuurisen kestävyyden osalta. Toisaalta, vaikka resurssien yhdistäminen onkin usein tärkeä motiivi VALO-kehitykseen osallistumiselle, eri hallintotasojen välinen kilpailu resursseista voi aiheuttaa konflikteja ja haasteita resurssikestävyyden näkökulmasta. Organisaatiolle saattaa olla houkuttelevaa ohjata kaikki taloudelliset ja henkilöstöresurssit omiin projekteihin ja samoin yhteisöt saattavat priorisoida omat projektinsa ekosysteemin yhteisten kehitystarpeiden kustannuksella, mikä voi heikentää jaettujen ohjelmisto-omaisuuksien arvoa pitkällä aikavälillä. Etenkin organisaatio- ja yhteisöhallinnon välillä resurssikonfliktit voivat vaikeuttaa myös rakentavan vuorovaikutuksen ylläpitämistä sekä etenkin oikeudellisten kehysten kuten lisenssien osalta yhteisestä infrastruktuurista sopimista.

Parhaimmillaan hallintotasojen vuorovaikutus tukee VALO-yhteisöjen kasvua ja kestävyyttä positiivisen palautesilmukan kautta. Tämä prosessi alkaa käyttäjäkunnan kasvattamisesta, jota tukevat yhteisön ja organisaatioiden markkinointitoimenpiteet sekä ekosysteemin tarjoama näkyvyys. Seuraavassa vaiheessa laajentunut käyttäjäkunta muuttuu osallistujiksi ja sponsoreiksi. Tämä edellyttää sitä, että yhteisöhallinto onnistuu luomaan ympäristön, jossa osallistuminen on helppoa ja tehokasta ja organisaatiohallinnot tekevät päätöksiä resurssien investoimisesta. Ekosysteemihallinto voi myös osaltaan auttaa tukemalla henkilöresurssien liikkumista projektien välillä. Kolmannessa vaiheessa osallistumiset ja sponsoroinnit muutetaan laadukkaiksi ohjelmistotuotteiksi. Yhteisöhallinto voi tukea arvonluontia huolehtimalla asianmukaisesta laadunvalvonnasta ja tehtävien koordinoinnista. Ekosysteemihallinto tukee teknisten ratkaisujen yhteentoimivuutta, joka kasvattaa ohjelmiston arvoa, samoin kuin yritysten tarjoamat lisäpalvelut. On kuitenkin huomattava, että liian nopea ja huonosti hallittu kasvu voi olla kuormittava tekijä kaikille osapuolille ja heikentää kestävyyttä.

Tutkimuksessa nousi myös esiin merkittäviä eroja julkisen ja yksityisen sektorin VALO-hallinnossa. Julkisen sektorin organisaatioilla on usein haasteita rahoituksen ja resurssien kanssa, mikä vaikuttaa negatiivisesti VALO-projektien kestävyyteen erityisesti nopean kasvun tilanteissa. Julkisten hankintalakien asettamat rajoitukset voivat lisäksi vaikeuttaa luottamukseen perustuvien suhteiden muodostumista ja ylläpitämistä yhteisöissä ja ekosysteemeissä. Organisaatiotason hallinnossa julkisen ja yksityisen sektorin välillä ei havaittu yhtä selkeitä eroja. Molempien sektorien sisällä organisaatioiden sitoutumisessa VALO-kehitykseen oli huomattavia eroja, mikä vaikutti merkittävästi etenkin resurssipohjaiseen ja osin myös vuorovaikutukselliseen kestävyyteen.

Kokonaisuudessaan väitöskirja syventää ymmärrystä VALO-hallinnosta ja sen vaikutuksista kestävyyteen. Lopuksi esitetään useita suuntaviivoja tulevalle tutkimukselle. Näitä ovat muun muassa kattavan kehyksen kehittäminen, joka yhdistäisi käytännöt organisaatio-, yhteisö- ja ekosysteemitasoilla. Lisäksi tutkimuksessa ehdotetaan kestävyyskäsitteen laajentamista VALO-kontekstissa siten, että se kattaisi myös taloudelliset, ympäristölliset ja sosiaaliset ulottuvuudet. Ideologisten motiivien ja institutionaalisten arvojen vaikutukset organisaatioiden osallistumiseen VALO-hankkeisiin tunnistettiin myös kiinnostavaksi tutkimusalueeksi.

REFERENCES

Aksulu, A., & Wade, M. (2010). A comprehensive review and synthesis of open source research. *Journal of the Association for Information Systems*, *11*(11), 576–656. https://doi.org/10.17705/1jais.00245

Alamer, G., & Alyahya, S. (2017). Open source software hosting platforms: A collaborative perspective's review. *Journal of Software*, *12*(4), 274–291. https://doi.org/10.17706/jsw.12.4.274-291

Alami, A., Cohn, M. L., & Waisowski, A. (2020). How do FOSS communities decide to accept pull requests? *The 24th International Conference on Evaluation and Assessment in Software Engineering (EASE '20)*, 220–229. https://doi.org/10.1145/3383219.3383242

Alarcon, G. M., Walter, C., Gibson, A. M., Gamble, R. F., Capiola, A., Jessup, S. A., & Ryan, T. J. (2020). Would you fix this code for me? Effects of repair source and commenting on trust in code repair. *Systems, 8*(1), 8. https://doi.org/10.3390/SYSTEMS8010008

Alexy, O., West, J., Klapper, H., & Reitzig, M. (2018). Surrendering control to gain advantage: Reconciling openness and the resource-based view of the firm. *Strategic Management Journal*, *39*(6), 1704–1727. https://doi.org/10.1002/SMJ.270

Allee, V. (2003). *The future of knowledge: Increasing prosperity through value networks*. Butterworth-Heinemann. https://doi.org/10.1002/SMJ.2706

Allee, V. (2008). Value network analysis and value conversion of tangible and intangible assets. *Journal of Intellectual Capital*, *9*(1), 5–24. https://doi.org/10.1108/14691930810845777

Almarzouq, M., Alzaidan, A., & Al Dallal, J. (2022). The Relevance of SourceForge Data in the Age of GitHub. *ACM SIGMIS Database: The DATABASE for Advances in Information Systems*, *53*(4), 83–93. https://doi.org/10.1145/3571823.3571830

Alspaugh, T. A., & Scacchi, W. (2010). Software licenses in context: The challenge of heterogeneously-licensed systems. *Journal of the Association for Information Systems*, *11*(11), 2. https://doi.org/10.17705/1jais.00241

Alves, C., Oliveira, J., & Jansen, S. (2017). Software ecosystems governance a systematic literature review and research agenda. *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems*, *3*, 215–226. https://doi.org/10.5220/0006269402150226

Amherst, S., Schweik, C. M., & English, R. (2007). Tragedy of the FOSS commons? Investigating the institutional designs of free/libre and open source software projects. *First Monday*, *2*(12). https://doi.org/10.5210/fm.v12i2.1619

Amorim, S., McGregor, J. D., Almeida, E. S. de, & Chavez, C. von F. G. (2023). Software architectural practices: Influences on the open source ecosystem health. *Journal of Software Engineering Research and Development*, *11*(1), 9–23. https://doi.org/10.5753/JSERD.2023.967

Antikainen, M., Aaltonen, T., & Väisänen, J. (2007). In J. Feller, B. Fitzgerald, W. Scacchi, & A. Sillitti (Eds.), *Open source development, adoption and innovation*. OSS 2007. IFIP Advances in Information and Communication Technology, 234. 223–228. Springer. https://doi.org/10.1007/978-0-387-72486-7_19

Armenise, V. (2015). Continuous delivery with Jenkins: Jenkins solutions to implement continuous delivery. *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, 24–27. https://doi.org/10.1109/RELENG.2015.19

Bazarhanova, A., Yli-Huumo, J., & Smolander, K. (2018). Love and hate relationships in a platform ecosystem: A case of Finnish electronic identity management. *The 51st Hawaii International Conference on System Sciences 2018 (HICSS-51)*, 1493–1052.

Belo, Í., & Alves, C. (2021). How to create a software ecosystem? A partnership meta-model and strategic patterns. *Information, 12*(6), 240. https://doi.org/10.3390/INFO12060240

Benkler, Y. (2002). Coase's Penguin, or, Linux and the nature of the firm. *The Yale Law Journal*, *112*(3), 369–446.

Benkler, Y. (2006). *The wealth of networks: How social production transforms markets and freedom*. Yale University Press.

Bergquist, M., Ljungberg, J., & Rolandsson, B. (2011). A historical account of the value of free and open source software: From software commune to commercial commons. In S. A. Hissam, B. Russo, M. G. de Mendonça Neto, & F. Kon (Eds.), *Open source systems: Grounding research*. OSS 2011. IFIP Advances in Information and Communication Technology, 365, 196–207. https://doi.org/10.1007/978-3-642-24418-6_13

Bettenburg, N., Hassan, A. E., Adams, B., & German, D. M. (2015). Management of community contributions: A case study on the Android and Linux software ecosystems. *Empirical Software Engineering*, *20*(1), 252–289. https://doi.org/10.1007/S10664-013-9284-6

Blind, K., Böhm, M., Grzegorzewska, P., Katz, A., Muto, S., Pätsch, S., & Schubert, T. (2021). *The impact of Open Source Software and Hardware on technological independence, competitiveness and innovation in the EU economy. Final Study Report.* Publications Office of the European Union.

Bonaccorsi, A., & Rossi, C. (2005). Altruistic individuals, selfish firms? The structure of motivation in open source software. *First Monday*, *10*(SPEC. ISS. 2). https://doi.org/10.2139/SSRN.433620

Bryman, A. (2008). *Social Research Methods*. Oxford University Press. ISBN 0199202958, 9780199202959

Buchner, S., & Riehle, D. (2023). The business impact of inner source and how to quantify it. *ACM Computing Surveys*, *56*(2). https://doi.org/10.1145/3611648

Butler, B. S. (2001). Membership size, communication activity, and sustainability: A resource-based model of online social structures. *Information Systems Research*, *12*(4), 346–362. https://doi.org/10.1287/ISRE.12.4.346.9703

Butler, S., Gamalielsson, J., Lundell, B., Brax, C., Mattsson, A., Gustavsson, T., Feist, J., Kvarnström, B., & Lönroth, E. (2022). Considerations and challenges for the adoption of open source components in software-intensive businesses. *Journal of Systems and Software*, *186*, 111152. https://doi.org/10.1016/J.JSS.2021.111152

Capra, E., Francalanci, C., & Merlo, F. (2008). An empirical study on the relationship between software design quality, development effort, and governance in open source projects. *IEEE Transactions on Software Engineering*, *34*(6), 765–782. https://doi.org/10.1109/TSE.2008.68

Capra, E., Francalanci, C., Merlo, F., & Rossi-Lamastra, C. (2011). Firms' involvement in Open Source projects: A trade-off between software structural quality and popularity. *Journal of Systems and Software*, *84*(1), 144–161. https://doi.org/10.1016/j.jss.2010.09.004

Capraro, M., & Riehle, D. (2016). Inner source definition, benefits, and challenges. *ACM Computing Surveys (CSUR)*, *49*(4). https://doi.org/10.1145/2856821

Carillo, K., & Bernard, J.-G. (2015). How many penguins can hide under an umbrella? An examination of how lay conceptions conceal the contexts of free/open source software. *ICIS 2015 Proceedings*, 16. https://aisel.aisnet.org/icis2015/proceedings/ManagingIS/16

Carosone, Michelle. (2017). FOSS governance programs: Why do you need one? *International In-House Counsel Journal*, *41*(11).

Carst, A. E., & Hu, Y. (2023). Complementors as ecosystem actors: A systematic review. *Management Review Quarterly*. https://doi.org/10.1007/s11301-023-00368-y

Cassell, C., Symon, G., Humphrey, C., & Lee, B. (2004). *Qualitative methods in organizational research*. Elsevier BV.

Ceruzzi, P. (2003). *A history of modern computing*. MIT Press.

Chengalur-Smith, I., Sidorova, A., & Daniel, S. (2010). Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems*, *11*(11), 5. https://doi.org/10.17705/1jais.00244

Chesbrough, H. (2003). The logic of open innovation. *California Management Review*, *45*(3), 33–58. https://doi.org/10.1177/000812560304500301

Chesbrough, H. (2004). Managing open innovation. *Research-Technology Management*, *47*(1), 23–26. https://doi.org/10.1080/08956308.2004.11671604

Chesbrough, H. (2017). The future of open innovation. *Research-Technology Management*, *60*(1), 35–38. https://doi.org/10.1080/08956308.2017.1255054

Chetty, S. (1996). The case study method for research in small-and medium-sized firms. *International Small Business Journal*, *15*(1), 73–85. https://doi.org/10.1177/0266242696151005

Ciffolilli, A. (2004). The economics of open source hijacking and the declining quality of digital information resources: A case for copyleft. *First Monday*, *9*(9). https://doi.org/10.5210/FM.V9I9.1173

Colazo, J., & Fang, Y. (2009). Impact of license choice on Open Source Software development activity. *Journal of the American Society for Information Science and Technology*, *60*(5), 997–1011. https://doi.org/10.1002/ASI.21039

Cowan, C. (2023). Open and closed software security redux. *IEEE Security and Privacy*, *21*(2), 18–23. https://doi.org/10.1109/MSEC.2022.3227819

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys*, *44*(2). https://doi.org/10.1145/2089125.2089127

Crowston, K., Wei, K., Li, Q., Eseryel, U. Y., & Howison, J. (2005). Coordination of free/libre open-source software development. *International Conference on Information Systems (ICIS 2005)*,18–26. https://doi.org/10.1145/1029997.1030003

Curto-Millet, D., & Corsín Jiménez, A. (2023). The sustainability of open source commons. *European Journal of Information Systems*, *32*(5), 763–781. https://doi.org/10.1080/0960085X.2022.2046516

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, 1277–1286. https://doi.org/10.1145/2145204.2145396

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

Dahlander, L., & Magnusson, M. G. (2006). Business models and community relationships of open source software firms. *The Economics of Open Source Software Development*, 111–130. https://doi.org/10.1016/B978-044452769-1/50005-6

Dahlander, L., & O'Mahony, S. (2010). Progressing to the center: Coordinating project work. *Organization Science*, *22*(4), 961–979. https://doi.org/10.1287/ORSC.1100.0571

Daniel, S. L., Maruping, L. M., Cataldo, M., & Herbsleb, J. (2018). The impact of ideology misfit on open source software communities and companies. *MIS Quarterly*, *42*(4), 1069–1096. https://doi.org/10.25300/MISQ/2018/14242

De Laat, P. B. (2007). Governance of open source software: State of the art. *Journal of Management and Governance*, *11*(2), 165–177. https://doi.org/10.1007/S10997-007-9022-9

Demil, B., & Lecocq, X. (2006). Neither market nor hierarchy nor network: the emergence of bazaar governance. *Organization Studies*, 27(10), 1447–1466. https://doi.org/10.1177/0170840606067250

den Besten, M., Dalle, J. M., & Galia, F. (2008). The allocation of collaborative efforts in open-source software. *Information Economics and Policy*, *20*(4), 316–322. https://doi.org/10.1016/J.INFOECOPOL.2008.06.003

Dennehy, D., Conboy, K., Ferreira, J., & Babu, J. (2023). Sustaining open source communities by understanding the influence of discursive manifestations on sentiment. *Information Systems Frontiers*, *25*(1), 241–257. https://doi.org/10.1007/s10796-020-10059-8

Dhungana, D. , Groher. I. , Schludermann E. , & Biffl, S. (2013). Guiding principles of natural ecosystems and their applicability to software ecosystems. In S. Jansen, M. Cusumano, & S. Brinkkemper (Eds.), *Software ecosystems: analysing and managing business networks in the software industry* (pp. 43–58). Edward Elgar Publishing,.

Di Tullio, D., & Staples, D. S. (2013). The governance and control of open source software projects. *Journal of Management Information Systems*, *30*(3), 49–80. https://doi.org/10.2753/MIS0742-1222300303

Digkas, G., Lungu, M., Avgeriou, P., Chatzigeorgiou, A., & Ampatzoglou, A. (2018). How do developers fix issues and pay back technical debt in the Apache ecosystem? *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering*, 153–163. https://doi.org/10.1109/SANER.2018.8330205

Dinh-Trong, T. T., & Bieman, J. M. (2005). The FreeBSB project: A replication case study of open source development. *IEEE Transactions on Software Engineering*, *31*(6), 481–494. https://doi.org/10.1109/TSE.2005.73

Duparc, E., Möller, F., Jussen, I., Stachon, M., Algac, S., & Otto, B. (2022). Archetypes of open-source business models. *Electronic Markets*, *32*(2), 727–745. https://doi.org/10.1007/S12525-022-00557-9

Easton, G. (2010). Critical realism in case study research. *Industrial Marketing Management*, *39*(1), 118–128. https://doi.org/10.1016/J.INDMARMAN.2008.06.004

Eisenhardt, K. M. (1989). Building theories from case study research. *The Academy of Management Review*, *14*(4), 532–550. https://doi.org/10.2307/258557

Elkington, J. (1998). Partnerships from cannibals with forks: The triple bottom line of 21st-century business. *Environmental Quality Management*, *8*(1), 37–51. https://doi.org/10.1002/TQEM.3310080106

Elliott, M. S., & Scacchi, W. (2008). Mobilization of software developers: The free software movement. *Information Technology & People*, *21*(1), 4–33. https://doi.org/10.1108/09593840810860315/FULL/XML

English, R., & Schweik, C. M. (2007). Identifying success and tragedy of FLOSS commons: A preliminary classification of Sourceforge.net projects. *First International Workshop on Emerging Trends in FLOSS Research and Development, FLOSS'07*, 11–15. https://doi.org/10.1109/FLOSS.2007.9

Fang, Y., & Neufeld, D. (2009). Understanding sustained participation in open source software projects. *Journal of Management Information Systems*, *25*(4), 9–50. https://doi.org/10.2753/MIS0742-1222250401

Faridian, P. (2023). Leading open innovation: The role of strategic entrepreneurial leadership in orchestration of value creation and capture in GitHub open source communities. *Technovation*, *119*, 102546. https://doi.org/10.1016/J.TECHNOVATION.2022.102546

Favario, L. (2023). Toward a free and open source-driven public sector: An Italian journey. *IEEE Software*, *40*(4), 55–61. https://doi.org/10.1109/MS.2023.3266706

Feldman, S. S., & Horan, T. A. (2011). Collaboration in electronic medical evidence development: A case study of the social security administration's MEGAHIT system. *International Journal of Medical Informatics*, *80*(8), e127–e140. https://doi.org/10.1016/J.IJMEDINF.2011.01.012

Fendt, O., & Jaeger, M. C. (2019). Open source for open source license compliance. In F. Bordeleau, A. Sillitti, P. Meirelles, & V. Lenarduzzi (Eds.), *Open source systems*. OSS 2019. IFIP advances in information and communication technology, 556, 133–138. https://doi.org/10.1007/978-3-030-20883-7_12

Fielding, R. T. (1999). Shared leadership in the Apache project. *Communications of the ACM*, *42*(4), 42–34.

Finney, G. (2009). The evolution of GPLV3 and contributor agreements in open-source software. *Journal of Technology Law & Policy*, *14*.

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly*, *30*(3), 587–598. https://doi.org/10.2307/25148740

Flyvberg, B. (2004). Five misunderstandings about case-study research. *Sosiologisk Tidsskrift*, *12*(2), 117–142. https://doi.org/10.18261/ISSN1504-2928-2004-02-02

Flyvberg, B. (2011). Case study. In N. K. Denzin & Y. S. Lincoln (Eds.), *The Sage Handbook of Qualitative Research* (pp. 301-316). Sage.

Fortunato, L., & Galassi, M. (2021). The case for free and open source software in research and scholarship. *Philosophical Transactions of the Royal Society A*, *379*(2197). https://doi.org/10.1098/RSTA.2020.0079

Franck, E., & Jungwirth, C. (2003). Reconciling rent-seekers and donators - The governance structure of open source. *Journal of Management and Governance*, *7*(4), 401–421. https://doi.org/10.1023/A:1026261005092

Franco-Bedoya, O., Ameller, D., Costal, D., & Franch, X. (2017). Open source software ecosystems: A Systematic mapping. *Information and Software Technology*, *91*, 160–185. https://doi.org/10.1016/J.INFSOF.2017.07.007

Free Software Foundation. (2024). *Various Licenses and Comments about Them*. https://www.gnu.org/licenses/license-list.html

Gallivan, M. J. (2001). Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies. *Information Systems Journal*, *11*(4), 277–304. https://doi.org/10.1046/J.1365-2575.2001.00108.X

Gamalielsson, J., Lundell, B., Butler, S., Brax, C. T., Persson, T., Mattsson, A., Gustavsson, T., Feist, J., & Lönroth, E. (2021). Towards open government through open source software for web analytics: The case of Matomo. *Journal of E-Democracy and Open Government*, *13*(2), 133. https://doi.org/10.29379/JEDEM.V13I2.650

Gangadharan, G. R., D'Andrea, V., De Paoli, S., & Weiss, M. (2012). Managing license compliance in free and open source software development. *Information Systems Frontiers*, *14*(2), 143–154. https://doi.org/10.1007/S10796-009-9180-1

Gasson, S., & Purcelle, M. (2018). A participation architecture to support user peripheral participation in a hybrid FOSS community. *ACM Transactions on Social Computing*, *1*(4), 1–46. https://doi.org/10.1145/3290837

Germonprez, M., Kendall, K., Kendall, J., Young, B., & Warner, B. (2013). The domestication of open source. *Diffusion Interest Group In Information Technology (DIGIT) Proceedings*, *8*. http://aisel.aisnet.org/digit2013/8

Ghapanchi, A. H., Wohlin, C., & Aurum, A. (2014). Resources contributing to gaining competitive advantage for open source software projects: An application of resource-based theory. *International Journal of Project Management*, *32*(1), 139–152. https://doi.org/10.1016/J.IJPROMAN.2013.03.002

Gharehyazie, M., Posnett, D., Vasilescu, B., & Filkov, V. (2015). Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. *Empirical Software Engineering*, *20*(5), 1318–1353. https://doi.org/10.1007/S10664-014-9332-X

Gnome Foundation. (2023, June 23). *GNOME Foundation Board of Directors Elections 2023*. GNOME Discourse / Elections and Referendums. https://discourse.gnome.org/t/gnome-foundation-board-of-directors-elections-2023/15657

Goldkuhl, G. (2012). Pragmatism vs interpretivism in qualitative information systems research. *European Journal of Information Systems*, *21*(2), 135–146. https://doi.org/10.1057/EJIS.2011.54

Groff, R. (2004). *Critical realism, post-positivism, and the possibility of knowledge*. Routledge.

Hardin, G. (1968). The tragedy of the commons. *Science*, *162*(3859), 1243–1248. https://doi.org/10.1126/SCIENCE.162.3859.1243

Hardin, G. (2007). The tragedy of the unmanaged commons. In D. Pen & I. Mysterud (Eds.), *Evolutionary Perspectives on Environmental Problems* (pp. 106–108). Routledge. https://doi.org/10.4324/9780203792650

Hariharan, A. (2023). *Evaluation of Volunteering Capabilities in an Open-Source Software Community* [Master's thesis]. University of Calgary.

Hars, A., & Ou, S. (2002). Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, *6*(3), 25–39. https://doi.org/10.1080/10864415.2002.11044241

Harutyunyan, N. (2019). *Corporate open source governance of software supply chains* [Phd Dissertation]. Friedrich-Alexander-Universität Erlangen-Nürnberg.

Harutyunyan, N. (2020). Managing your open source supply chain- why and how? *Computer*, *53*(6), 77–81. https://doi.org/10.1109/MC.2020.2983530

Harutyunyan, N. (2022). Open source software governance: Distilling and applying industry best practices. In Felderer, M., et al. (Edss). *Ernst Denert Award for Software Engineering* (pp. 73–90). Springer. https://doi.org/10.1007/978-3-030-83128-8_5

Harutyunyan, N., & Riehle, D. (2019). Industry best practices for open source governance and component reuse. In *Proceedings of the 24th European*

Conference on Pattern Languages of Programs (EuroPLop '19), Association for Computing Machinery, https://doi.org/10.1145/3361149.3361170

Harutyunyan, N., & Riehle, D. (2021). Getting started with corporate open source governance: A case study evaluation of industry best practices. *The 54th Hawaii International Conference on System Sciences 2021 (HICSS-54)*. https://aisel.aisnet.org/hicss-54/os/practice-based_research/3

Hecker, F. (1999). Setting up shop: The business of open-source software. *IEEE Software*, *16*(1), 45–51. https://doi.org/10.1109/52.744568

Heimburg, V., & Wiesche, M. (2022). Relations between actors in digital platform ecosystems: A literature review. *ECIS 2022 Research Papers*, *93*, https://aisel.aisnet.org/ecis2022_rp/93

Hein, A., Schreieck, M., Riasanow, T., Setzke, D. S., Wiesche, M., Böhm, M., & Krcmar, H. (2020). Digital platform ecosystems. *Electronic Markets*, *30(1)*, 87–98. https://doi.org/10.1007/S12525-019-00377-4

Hemetsberger, A., & Reinhardt, C. (2009). Collective development in open-source communities: An activity theoretical perspective on successful online collaboration. *Organization Studies*, *30*(9), 987–1008. https://doi.org/10.1177/0170840609339241

Henkel, J. (2006). Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy*, *35*(7), 953–969. https://doi.org/10.1016/J.RESPOL.2006.04.010

Henttonen, K. (2007). *Stylebase for Eclipse: An open-source tool to support the modelling of quality-driven software architecture.* VTT Tiedotteita - Meddelanden - Research Notes No. 2387. VTT Technical Research Centre of Finland. ISBN 978-951-38-6925-0

Henttonen, K. (2011). *Open source as an innovation enabler: Case study of an Indian SME* [M.Sc. Thesis]. University of Manchester.

Henttonen, K. (2020). *Interviews and observation notes from a multi-case study on three open-source software ecosystems (Decidim.org, Oskari.org and Plone.org)*, University of Jyväskylä. https://doi.org/10.17011/JYX/DATASET/72939

Henttonen, K. (2024). *Additional documents and observation notes from a multi-case study on three open-source software ecosystems (Decidim.org, Oskari.org and Plone.org)*. University of Jyväskylä. https://doi.org/10.17011/JYX/DATASET/96515

Hewapathirana, R. (2017). *FOSS as a platform ecosystem: Understanding governance of open source HIS implementation in a low and middle income country context* [PhD Dissertation], University of Oslo.

Himanen, P. (2004). The hacker ethic as the culture of the information age. In M. Castells (Ed.), *The Network Society: A Cross-cultural Perspective*. Edward Elgar Publishing.

Howison, J., & Crowston, K. (2014). Collaboration through open superposition. *Mis Quarterly*, *38*(1), 29–50. https://www.jstor.org/stable/26554867

Hsieh, H., &., Shannon, S. (2005). Three approaches to qualitative content analysis. *Qualitative Health Research*, *15*(9), 1277–1288. https://doi.org/10.1177/1049732305276687

Hyrynsalmi, S., Seppänen, M., Nokkala, T., Suominen, A., & Järvi, A. (2015). Wealthy, healthy and/or happy —what does 'ecosystem health' stand for? In J. Fernandes, R. Machado, & K. Wnuk (Eds.), *Software business. ICSOB 2015*. Lecture Notes in Business Information Processing, 210, 272–287. Springer. https://doi.org/10.1007/978-3-319-19593-3_24

Iansiti, M., & Levien, R. (2004). Strategy as ecology. *Harvard Business Review*, *82*(3), 68–78, 126.

Izquierdo, J. L. C., & Cabot, J. (2018). The role of foundations in open source projects. *In Proceedings of 2018 ACM/IEEE 40th International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS 2018*, 3–12. https://doi.org/10.1145/3183428.3183438

Jalali, S., & Wohlin, C. (2012). Systematic literature studies: Database searches vs. backward snowballing. *The 6th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 29-38. https://doi.org/10.1145/2372251.2372257

Jansen, S. (2014). Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, *56*(11), 1508–1519. https://doi.org/10.1016/J.INFSOF.2014.04.00

Jansen, S. (2020). A focus area maturity model for software ecosystem governance. *Information and Software Technology*, *118*, 106219. https://doi.org/10.1016/J.INFSOF.2019.106219

Jansen, S., Brinkkemper, S., Souer, J., & Luinenburg, L. (2012). Shades of gray: Opening up a software producing organization with the open software enterprise model. *Journal of Systems and Software*, *85*(7), 1495–1510. https://doi.org/10.1016/J.JSS.2011.12.007

Jayaratna, N. (1994). *Understanding and evaluating methodologies: NIMSAD, a systemic framework*. McGraw-Hill.

Jensen, C., & Scacchi, W. (2005). Collaboration, leadership, control, and conflict negotiation and the Netbeans.org open source software development community. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 196. https://doi.org/10.1109/HICSS.2005.147

Jensen, C., & Scacchi, W. (2007). Role migration and advancement processes in OSSD projects: A comparative case study. *Proceedings - International Conference on Software Engineering*, 364–373. https://doi.org/10.1109/ICSE.2007.74

Jiang, J., Mohamed, A., & Zhang, L. (2019). What are the characteristics of reopened pull requests? A case study on open source projects in GitHub. *IEEE Access*, *7*, 102751–102761. https://doi.org/10.1109/ACCESS.2019.2928566

Joo, C., Kang, H., & Lee, H. (2012). Anatomy of open source software projects: Evolving dynamics of innovation landscape in open source software ecology. *The 5th International Conference on Communications, Computers and Applications (MIC-CCA2012)*, 96–100.

Kääriäinen, J., Pussinen, P., Matinmikko, T., & Oikarinen, T. (2012). Lifecycle management of open-source software in the public sector: A model for community-based application evolution. *ARPN Journal of Systems and Software*, *2*(11), 279–288.

Kabbedijk, J., & Jansen, S. (2011). Steering insight: An exploration of the ruby software ecosystem. In B. Regnell, I. van de Weerd, & O. De Troyer (Eds.), *Software Business. ICSOB 2011*. Lecture Notes in Business Information Processing, 80. Springer. https://doi.org/10.1007/978-3-642-21544-5_5

Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L., & German, D. M. (2015). Open source-style collaborative development practices in commercial projects using GitHub. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, *1*, 574–585. https://doi.org/10.1109/ICSE.2015.74

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2016). An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering*, *21*(5), 2035–2071. https://doi.org/10.1007/S10664-015-9393-5

Kane, G. C., & Ransbotham, S. (2016). Content as community regulator: The recursive relationship between consumption and contribution in open collaboration communities. *Organization Science*, *27*(5), 1258–1274. https://doi.org/10.1287/ORSC.2016.1075

Karger, T. (2023). Bounded ownership: Lessons learned from online platforms in creating inclusive goods. *Social Media + Society*, *9*(2). https://doi.org/10.1177/20563051231175624

Kastrinou Theodoropoulou, A. M. A. (2008). The Gift of the Code: The culture of an operating system. *Durham Anthropology Journal*, *15*(1). ISSN: 1742-2930

Kaur, R., Kaur Chahal, K., & Saini, M. (2022). Understanding community participation and engagement in open source software Projects: A systematic mapping study. *Journal of King Saud University - Computer and Information Sciences*, *34*(7), 4607–4625. https://doi.org/10.1016/J.JKSUCI.2020.10.020

Khan, S., Lacity, M., & Carmel, E. (2018). Entrepreneurial impact sourcing: A conceptual framework of social and commercial institutional logics. *Information Systems Journal*, *28*(3), 538–562. https://doi.org/10.1111/ISJ.12134

Kilamo, T., Hammouda, I., Mikkonen, T., & Aaltonen, T. (2012). From proprietary to open source - Growing an open source ecosystem. *Journal of Systems and Software*, *85*(7), 1467–1478. https://doi.org/10.1016/j.jss.2011.06.071

Kilamo, T., Lenarduzzi, V., Ahoniemi, T., Jaaksi, A., Rahikkala, J., & Mikkonen, T. (2020). How the cathedral embraced the bazaar, and the bazaar became a cathedral. In V. Ivanov, A. Kruglov, S. Masyagin, A. Sillitti & G. Succi, (Eds.), *Open Source Systems. OSS 2020*. IFIP Advances in Information and Communication Technology, 582, 141–147. https://doi.org/10.1007/978-3-030-47240-5_14

King, N. (1998). Template analysis. In G. Symon & C. Cassell (Eds.), *Qualitative methods and analysis in organizational research: A practical guide,* 118–134. Sage Publications Ltd.

King, N. (2012). Doing template analysis. In C. Cassell & G. Symon (Eds.), *Qualitative Organizational Research: Core Methods and Current Challenges* (pp. 426–450). Sage Publications Ltd. https://doi.org/10.4135/9781526435620.N24

King, N., & Brooks, J. M. (2017). *Template analysis for business and management students*. Sage Publications Ltd. https://doi.org/10.4135/9781473983304

Krishnamurthy, S. (2005). An analysis of open source business models. In B. Fitzgerald, S. Hissam, & K. Lakhani (Eds.), *Making sense of the bazaar: Perspectives on open source and free software*. MIT Press.

Krishnamurthy, S., Ou, S., & Tripathi, A. K. (2014). Acceptance of monetary rewards in open-source software development. *Research Policy*, *43*(4), 632–644. https://doi.org/10.1016/J.RESPOL.2013.10.007

Lakhani, K. R., & Wolf, R. G. (2003). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. *SSRN Electronic Journal*. https://doi.org/10.2139/SSRN.443040

Lane, M., Vyver, G., Basnet, P., & Howard, S. (2004). Interpretative insights into interpersonal trust and effectiveness of virtual communities of open source software (OSS) developers. *ACIS 2004 Proceedings, 64*. https://aisel.aisnet.org/acis2004/67

Langlois, R. N., & Garzarelli, G. (2008). Of hackers and hairdressers: Modularity and the organizational economics of open-source collaboration. *Industry and Innovation*, *15*(2), 125–143. https://doi.org/10.4324/9781315873503-2

Lattemann, C., & Stieglitz, S. (2005). Framework for governance in open source communities. *The 28th Annual Hawaii International Conference on System Sciences* (HICSS'28), 192. https://doi.org/10.1109/HICSS.2005.278

Lee, S. Y. T., Kim, H. W., & Gupta, S. (2009). Measuring open source software success. *Omega*, *37*(2), 426–438. https://doi.org/10.1016/J.OMEGA.2007.05.005

Leon, A. (2015). *Software configuration management handbook* (Third Edition). Artech House.

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *The Journal of Industrial Economics*, *50*(2), 197–234. https://doi.org/10.1111/1467-6451.00174

Lerner, J., & Tirole, J. (2005). The scope of open source licensing. *Journal of Law, Economics, and Organization*, *21*.

Levy, S. (1984). *Hackers: Heroes of the computer revolution*. Anchor Press/Doubleday.

Li, P., Maruping, L. M., & Mathiassen, L. (2020). Developing and managing open source enterprise systems through open superposition: A digital options and technical debt perspective. *AMCIS 2020 Proceedings, 2*.

Liebenau, J., & Smithson, S. (1994). Banning organizational secrecy can threaten research too. *European Journal of Information Systems, 3*(2), 83–86. https://doi.org/10.1057/EJIS.1994.9

Linåker, J., Munir, H., Wnuk, K., & Mols, C. E. (2018). Motivating the contributions: An Open Innovation perspective on what to share as Open Source Software. *Journal of Systems and Software*, *135*, 17–36. https://doi.org/10.1016/J.JSS.2017.09.032

Linåker, J., Papatheocharous, E., & Olsson, T. (2022). How to characterize the health of an Open Source Software project? A snowball literature review of an emerging practice. *The 18th International Symposium on Open Collaboration (OpenSym'22),* Association for Computing Machinery. https://doi.org/10.1145/3555051.3555067

Linåker, J., Robles, G., Bryant, D., & Muto, S. (2023). Open source software in the public sector: 25 years and still in its infancy. *IEEE Software*, *40*(4), 39–44. https://doi.org/10.1109/MS.2023.3266105

Linåker, J., & Runeson, P. (2020). Public sector platforms going open: Creating and growing an ecosystem with open collaborative development. *16th International Symposium on Open Collaboration (OpenSym'20),* Association for Computing Machinery. https://doi.org/10.1145/3412569.3412572

Ljungberg, J. (2000). Open source movements as a model for organising. *European Journal of Information Systems*, *9*(4), 208–216. https://doi.org/10.1057/PALGRAVE.EJIS.3000373

Luis, J., Anovas Izquierdo, C., & Cabot, J. (2020, May 20). A survey of software foundations in open source. *ArXiv E-Prints by Cornell University*. https://arxiv.org/abs/2005.10063v1

Lundell, B., Gamalielsson, J., Butler, S., Brax, C., Persson, T., Mattsson, A., Gustavsson, T., Feist, J., & Öberg, J.(2021). Enabling OSS usage through procurement projects: How can lock-in effects be avoided? In D. Taibi, V. Lenarduzzi, T. Kilamo & S. Zacchiroli, (Eds.). *Open Source Systems*. OSS 2021. IFIP Advances in Information and Communication Technology, 624, Springer. https://doi.org/10.1007/978-3-030-75251-4_2

Lundell, B., Gamalielsson, J., Tengblad, S., Hooshyar Yousefi, B., Fischer, T., Johansson, G., Rodung, B., Mattsson, A., Oppmark, J., Gustavsson, T., Feist, J., Landemoo, S., & Lönroth, E. (2017). Addressing lock-in, interoperability, and long-term maintenance challenges through open source: How can companies strategically use open source? *Open Source Systems.* IFIP Advances in Information and Communication Technology*, 496*, 80–88. Springer. https://doi.org/10.1007/978-3-319-57735-7_9

MacDonald, M. (2013). Open source licensing in the networked era. *Masaryk University Journal of Law and Technology*, *7*(2), 229–239.

Mäenpää, H. (2020). *Organizing and Managing Contributor Involvement in Hybrid Open Source Software Development Communities* [Phd Dissertation, University of Helsinki].

Mäenpää, H., Kilamo, T., & Männistö, T. (2016). In-between open and closed – Drawing the fine line in hybrid communities. In K. Crowston, I.

Hammouda, B. Lundell, G. Robles, J. Gamalielsson, & J. Lindman (Eds.), Open source systems: Integrating communities. OSS 2016. *IFIP Advances in Information and Communication Technology*, *472*, 134–146. https://doi.org/10.1007/978-3-319-39225-7_11

Manikas, K. (2016). Supporting the evolution of research in software ecosystems: Reviewing the empirical literature. In A. Maglyas & A. L. Lamprecht (Eds.), *Software Business*. ICSOB 2016. Lecture Notes in Business Information Processing, 240, 63–78. Springer. https://doi.org/10.1007/978-3-319-40515-5_5

Manikas, K., Hämäläinen, M., & Tyrväinen, P. (2016). Designing, developing, and implementing software ecosystems: Towards a step-wise guide. In S. Jansen, C. Alves, & J. Bosch (Eds.), *Proceedings of the 8th International Workshop on Software Ecosystems* (IWSECO 2016).

Manikas, K., & Hansen, K. M. (2013). Software ecosystems – A systematic literature review. *Journal of Systems and Software*, *86*(5), 1294–1306. https://doi.org/10.1016/J.JSS.2012.12.026

Manikas, K., & Hansen, K. (2013). Reviewing the health of software ecosystems–a conceptual framework proposal. In C. Alves, G. Hansen, & J. Bosch (Eds.), *The Proceedings* of *5th International Workshop on Software Ecosystems (IWSECO),* 33–44.

Markus, L., Manvilleand, B., & Agres, C. E. (2000). What makes a virtual organization work: Lessons from the open-source world. *MIT Sloan Management Review*, *October 2000*.

Markus, M. L. (2007). The governance of free/open source software projects: Monolithic, multidimensional, or configurational? *Journal of Management and Governance*, *11*(2), 151–163. https://doi.org/10.1007/s10997-007-9021-x

Marsan, J., Paré, G., & Wybo, M. D. (2012). Has open source software been institutionalized in organizations or not? *Information and Software Technology*, *54*(12), 1308–1316. https://doi.org/10.1016/J.INFSOF.2012.07.001

Maruping, L. M., Daniel, S. L., & Cataldo, M. (2019). Developer centrality and the impact of value congruence and incongruence on commitment and code contribution activity in open source software communities. *MIS Quarterly*, *43*(3), 951–976. https://doi.org/10.25300/MISQ/2019/13928

Maruping, L. M., & Matook, S. (2020). The evolution of software development orchestration: current state and an agenda for future research. *European Journal of Information Systems*, *29*(5), 443–457. https://doi.org/10.1080/0960085X.2020.1831834

Matinlassi, M., Ovaska, E., & Dobrica, L. (2002). *Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture*. VTT Publications 456. https://publications.vtt.fi/pdf/publications/2002/P456.pdf

Matinmikko, T., Kääriäinen, J., Kylmäaho, J., & Henttonen, K. (2017). Yhteispelillä kohti edullisempia ja laadukkaampia ohjelmistoja. *Kuntalehti*, *26*(6), 5050.

Maxwell, E. (2006). Open standards, open source, and open innovation: Harnessing the benefits of openness. *Innovations: Technology, Governance, Globalization*, *1*(3), 119–176. https://doi.org/10.1162/ITGG.2006.1.3.119

Mead, G. H. (1938). *Philosophy of the act*. University of Chicago Press.

Medappa, P. K., & Srivastava, S. C. (2019). Does superposition influence the success of floss projects? An examination of open-source software development by organizations and individuals. *Information Systems Research*, *30*(3), 764–786. https://doi.org/10.1287/isre.2018.0829

Medappa, P. K., & Srivastava, S. C. (2020). Ideological shifts in open source orchestration: Examining the influence of licence choice and organisational participation on open source project outcomes. *European Journal of Information Systems*, *29*(5), 500–520. https://doi.org/10.1080/0960085X.2020.1756003

Mergel, I. (2015). Open collaboration in the public sector: The case of social coding on GitHub. *Government Information Quarterly*, *32*(4), 464–472. https://doi.org/10.1016/J.GIQ.2015.09.004

Miller, C., Cohen, S., Klug, D., Vasilescu, B., & Kastner, C. (2022). "Did you miss my comment or what?" Understanding toxicity in open source discussions. *International Conference on Information Systems (ICIS 2022)*, *2022-May*, 710–722. https://doi.org/10.1145/3510003.3510111

Mizushima, K., & Ikawa, Y. (2011). A structure of co-creation in an open source software ecosystem: A case study of the eclipse community. *Proceedings of PICMET '11: Technology Management in the Energy Smart World (PICMET)*, 1–8. https://ieeexplore.ieee.org/abstract/document/6017787

Mockus, A., Fielding, R. T., & Herbsleb, J. (2000). A case study of open source software development. *Proceedings of the 22nd International Conference on Software Engineering - ICSE '00*, 263–272. https://doi.org/10.1145/337180.337209

Moon, E. (2021). Episodic peripheral contributors and technical dependencies in open source software (OSS) rcosystems. *Communications of the Association for Information Systems*, *49*(1), 8. https://doi.org/10.17705/1CAIS.04908

Moreira Soares, D., de Lima Júnior, M. L., Murta, L., & Plastino, A. (2021). What factors influence the lifetime of pull requests? *Software: Practice and Experience*, *51*(6), 1173–1193. https://doi.org/10.1002/SPE.2946

Morgan, D. L. (2014). Pragmatism as a paradigm for social research. *Qualitative Inquiry*, *20*(8), 1045–1053. https://doi.org/10.1177/1077800413513733

Mukhopadhyay, S., & Bouwman, H. (2019). Orchestration and governance in digital platform ecosystems: A literature review and trends. *Digital Policy, Regulation and Governance*, *21*(4), 329–351. https://doi.org/10.1108/DPRG-11-2018-0067

Müller, M., Diegmann, P., & Rosenkranz, C. (2019). Evolution of platform-based open source ecosystems: Uncovering socio-technical dynamics using digital traces. *International Conference on Interaction Sciences*.

Munir, H., & Mols, C. E. (2021). The rise of open source program office. *IT Professional*, *23*(1), 27–33. https://doi.org/10.1109/MITP.2020.3019961

Nevo, S., & Chengalur-Smith, I. S. (2017). Examining organizations' continued use of open source technologies: An institutional perspective. *Information Technology and People*, *30*(1), 24–46. https://doi.org/10.1108/ITP-09-2014-0204

Niemelä, E., & Ihme, T. (2001). Product line software engineering of embedded systems. *ACM SIGSOFT Software Engineering Notes*, *26*(3), 118–125. https://doi.org/10.1145/379377.375271

Nimmagadda, S., Reiners, T., Wood, L., & Mani, N. (2022). On developing sustainable digital ecosystems and their spatial-temporal knowledge management. *ACIS 2022 Proceedings*. https://aisel.aisnet.org/acis2022/20

Noni, I. De, Ganzaroli, A., & Orsi, L. (2011). The governance of open source software communities. *Journal of Law and Governance*, *6*(1), 1-18. https://doi.org/10.15209/JBSGE.V6I1.195

Nyman, L., & Lindman, J. (2013). Code forking, governance, and sustainability in open source software. *Technology Innovation Management Review*, January 2013, 7–12.

Oliveira, J., & Alves, C. (2021). Software ecosystems governance - An analysis of SAP and GNOME platforms. *47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2021)*, 296–299. https://doi.org/10.1109/SEAA53835.2021.00045

O'Mahony, S. (2007). The governance of open source initiatives: What does it mean to be community managed? *Journal of Management and Governance*, *11*(2), 139–150. https://doi.org/10.1007/S10997-007-9024-7

O'Mahony, S., & Bechky, B. A. (2008). Boundary organizations: Enabling collaboration among unexpected allies. *Administrative Science Quarterly*, 53(3), 422-459. https://doi.org/10.2189/ASQU.53.3.422

O'Mahony, S., & Ferraro, F. (2007). The emergence of governance in an open source community. *Academy of Management Journal*, *50*(5), 1079–1106. https://doi.org/10.5465/AMJ.2007.27169153

O'Mahony, S., & Karp, R. (2022). From proprietary to collective governance: How do platform participation strategies evolve? *Strategic Management Journal*, *43*(3), 530–562. https://doi.org/10.1002/SMJ.3150

Omar, I. (2005). View of the penguin in peril: SCO's legal threats to Linux. *First Monday*, *10*(1). https://doi.org/10.5210/fm.v10i1.1203

O'Neil, M., Muselli, L., Raissi, M., & Zacchiroli, S. (2021). 'Open source has won and lost the war': Legitimising commercial–communal hybridisation in a FOSS project. *New Media and Society*, *23*(5), 1157–1180. https://doi.org/10.1177/1461444820907022

Open Source Initiative. (2024). *Licenses – Open Source Initiative*. https://opensource.org/licenses/

O'Reilly, T. (1999). Lessons from open-source software development. *Communications of the ACM*, *42*(4), 32–37. https://doi.org/10.1145/299157.299164

Orsila, H., Geldenhuys, J., Ruokonen, A., & Hammouda, I. (2009). Trust issues in open source software development. *Proceedings of the Warm Up*

Workshop for ACM/IEEE ICSE 2010, WUP'09, 9–12. https://doi.org/10.1145/1527033.1527037

Ostrom, E. (1999). Coping with tragedies of the commons. *Annual Review of Political Science*, *2*(1999), 493–535.

Ostrom, E. (2009). A general framework for analyzing sustainability of social-ecological systems. *Science*, *325*(5939), 419–422.

Ovaska, E., Evesti, A., Henttonen, K., Palviainen, M., & Aho, P. (2010). Knowledge based quality-driven architecture design and evaluation. *Information and Software Technology*, *52*(6), 577–601 https://doi.org/10.1016/J.INFSOF.2009.11.008

Perens, B. (1999). The open source definition. In *Open sources: voices from the open source revolution* (pp. 171–188). O'Reilly Media.

Petersen, K., Badampudi, D., Shah, S. M. A., Wnuk, K., Gorschek, T., Papatheocharous, E., Axelsson, J., Sentilles, S., Crnkovic, I., & Cicchetti, A. (2018). Choosing component origins for software intensive systems: In-house, COTS, OSS or outsourcing? - A case survey. *IEEE Transactions on Software Engineering*, *44*(3), 237–261. https://doi.org/10.1109/TSE.2017.2677909

Pino, L. K., Searle, B. C., Bollinger, J. G., Nunn, B., MacLean, B., & MacCoss, M. J. (2020). The Skyline ecosystem: Informatics for quantitative mass spectrometry proteomics. *Mass Spectrometry Reviews*, *39*(3), 229–244. https://doi.org/10.1002/MAS.21540

Poo-Caamaño, G., Knauss, E., Singer, L., & German, D. M. (2017). Herding cats in a FOSS ecosystem: A tale of communication and coordination for release management. *Journal of Internet Services and Applications*, *8*(1), 1–24. https://doi.org/10.1186/S13174-017-0063-2

Raymond, E. (1997, May 27). The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary. *Linux Kongress* .

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, *12*(3), 23–49. https://doi.org/10.1007/s12130-999-1026-0

Riehle, D., & Berschneider, S. (2012). A model of open source developer foundations. In I. Hammouda, B. Lundell, T. Mikkonen, & W. Scacchi (Eds.), *Open source systems: Long-term sustainability*. OSS 2012. IFIP advances in information and communication technology, 378, 15–28 Springer. https://doi.org/10.1007/978-3-642-33442-9_2/

Riehle, D., Riemer, P., Kolassa, C., & Schmidt, M. (2014). Paid vs. volunteer work in open source. *The 47th Annual Hawaii International Conference on System Sciences (HICSS 2014)*, 3286–3295. https://doi.org/10.1109/HICSS.2014.407

Riembauer, S., Hornung, O., & Smolnik, S. (2020). Knowledge unchained or strategically overseen? Knowledge management in open source software projects. *The 53rd Hawaii International Conference on System Sciences (HICSS 2020)*, 5003–5012.

Rolfstam, M. (2012). An institutional approach to research on public procurement of innovation. *Innovation: The European Journal of Social*

Science Research, *25*(3), 303–321. https://doi.org/10.1080/13511610.2012.717475

Rolland, K. H., & Herstad, J. (2000). The “critical case” in information systems research. *Proceedings of IRIS 23*.

Rowley, J. (2002). Using case studies in research. *Management Research News*, *25*(1), 16–27. https://doi.org/10.1108/01409170210782990/FULL/XML

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. https://doi.org/10.1007/S10664-008-9102-8

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012). *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley and Sons. https://doi.org/10.1002/9781118181034

Sagers, G. (2004). The influence of network governance factors on success in open source software development projects. *ICIS 2004 Proceedings*. https://aisel.aisnet.org/icis2004/34

Sajadi, A., Damevski, K., & Chatterjee, P. (2023). Interpersonal trust in OSS: exploring dimensions of trust in GitHub pull requests. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE-NIER 2023)*, 19–24. https://doi.org/10.1109/ICSE-NIER58687.2023.00010

Sánchez, V. R., Ayuso, P. N., Galindo, J. A., & Benavides, D. (2020). Open source adoption factors-a systematic literature review. *IEEE Access*, *8*, 94594–94609. https://doi.org/10.1109/ACCESS.2020.2993248

Sapkota, H., Murukannaiah, P. K., & Wang, Y. (2019). A network-centric approach for estimating trust between open source software developers. *PLOS ONE*, *14*(12), e0226281. https://doi.org/10.1371/JOURNAL.PONE.0226281

Sayer, R. Andrew. (2010). *Method in social science: A realist approach* (Second Edition). Routledge.

Scacchi, W. (2010). Collaboration practices and affordances in free/open source software development. In I. Mistrík, J. Grundy, A. Hoek, & J. Whitehead (Eds.), *Collaborative Software Engineering* (307–327). Springer. https://doi.org/10.1007/978-3-642-10294-3_15

Scacchi, W., & Alspaugh, T. A. (2012). Understanding the role of licenses and evolution in open architecture software ecosystems. *Journal of Systems and Software*, *85*(7), 1479–1494. https://doi.org/10.1016/J.JSS.2012.03.033

Schaarschmidt, M., Walsh, G., & von Kortzfleisch, H. F. O. (2015). How do firms influence open-source software communities? A framework and empirical analysis of different governance modes. *Information and Organization*, *25*(2), 99–114. https://doi.org/10.1016/J.INFOANDORG.2015.03.001

Schoder, D., Schlagwein, D., & Fischbach, K. (2019). Open resource-based view (ORBV): A theory of resource openness. *ICIS 2019 Proceedings*. https://aisel.aisnet.org/icis2019/research_methods/research_methods/9

Scholtes, I., Mavrodiev, P., & Schweitzer, F. (2016). From Aristotle to Ringelmann: A large-scale analysis of team productivity and coordination in open source software projects. *Empirical Software Engineering*, *21*(2), 642–683. https://doi.org/10.1007/S10664-015-9406-4/FIGURES/15

Schreiber, R. R. (2023). Organizational influencers in open-source software projects. *International Journal of Open Source Software and Processes*, *14*(1). https://doi.org/10.4018/IJOSSP.318400

Schreieck, M., Wiesche, M., & Krcmar, H. (2023). Governing innovation platforms in multi-business organisations. *European Journal of Information Systems*, *32*(4), 695–716. https://doi.org/10.1080/0960085X.2022.2041371

Scott, T., & Rung, A. (2016). *Federal source code policy: Achieving efficiency, transparency, and innovation through reusable and open source software*. Memorandum for the Heads of Departments and Agencies M-16-21, Executive Office of the President. Whitehouse Archives.

Sen, R., Subramaniam, C., & Nelson, M. L. (2008). Determinants of the choice of open source software license. *Journal of Management Information Systems*, *25*(3), 207–240. https://doi.org/10.2753/MIS0742-1222250306

Sen, R., Subramaniam, C., & Nelson, M. L. (2011). Open source software licenses: Strong-copyleft, non-copyleft, or somewhere in between? *Decision Support Systems*, *52*(1), 199–206. https://doi.org/10.1016/J.DSS.2011.07.004

Shah, S. K. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, *52*(7), 1000–1014. https://doi.org/10.1287/MNSC.1060.055

Shahrivar, S., Elahi, S., Hassanzadeh, A., & Montazer, G. (2018). A business model for commercial open source software: A systematic literature review. *Information and Software Technology*, *103*, 202–214. https://doi.org/10.1016/J.INFSOF.2018.06.018

Shaikh, M., & Henfridsson, O. (2017). Governing open source software through coordination processes. *Information and Organization*, *27*(2), 116–135. https://doi.org/10.1016/J.INFOANDORG.2017.04.00

Simister, S. (1996). Understanding and evaluating methodologies: NIMSAD, a systemic framework. *Journal of the Operational Research Society*, *47*(4), 594–595. https://doi.org/10.1057/JORS.1996.67

Simons, H. (2014). Case study research: In-depth understanding in context. *The Oxford Handbook of Qualitative Research*, 454–470. https://doi.org/10.1093/OXFORDHB/9780199811755.013.00

Sinclair, A. (2010). License Profile: BSD. *International Free and Open Source Software Law Review*, *2*.

Spinellis, D. (2019). How to Select Open Source Components. *Computer*, *52*(12), 103–106. https://doi.org/10.1109/MC.2019.2940809

Stake, R. E. (1995). The art of case study research. Sage.

Stake, R. E. (2005). Qualitative case studies. In N. K. Denzin & Y. S. Lincoln (Eds.), *The Sage Handbook of Qualitative Research* (pp. 443-466). Sage.

Stallman, R. (1985). *The GNU Manifesto*. Gnu Project. https://www.gnu.org/gnu/manifesto.en.html

Stevens, G., & Draxler, S. (2010). Appropriation of the eclipse ecosystem: Local integration of global network production. *Proceedings of COOP 2010*, 287–308. https://doi.org/10.1007/978-1-84996-211-7_16

Stewart, K. J., Ammeter, A. P., & Maruping, L. M. (2006). Impacts of license choice and organizational sponsorship on user interest and development activity in open source software projects. *Information Systems Research*, *17*(2), 126–144. https://doi.org/10.1287/ISRE.1060.0082

Stewart, K. J., & Gosain, S. (2006). The impact of ideology on effectiveness in open source software development teams. *MIS Quarterly*, *30*(2), 291–314. https://doi.org/10.2307/25148732

Tang, T. (Ya), Fisher, G. J., & Qualls, W. J. (2021). The effects of inbound open innovation, outbound open innovation, and team role diversity on open source software project performance. *Industrial Marketing Management*, *94*, 216–228. https://doi.org/10.1016/J.INDMARMAN.2021.02.013

Teixeira, J., Robles, G., & González-Barahona, J. M. (2015). Lessons learned from applying social network analysis on an industrial Free/Libre/Open Source Software ecosystem. *Journal of Internet Services and Applications*, *6*(1). https://doi.org/10.1186/S13174-015-0028-2

Terrell, J., Kofink, A., Middleton, J., Rainear, C., Murphy-Hill, E., Parnin, C., & Stallings, J. (2017). Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science*, *2017*(5), e111. https://doi.org/10.7717/PEERJ-CS.111/SUPP-2

Thomas, L., & Samuel, K. (2017). Characteristics of open source business models. *The XXVIII ISPIM Innovation Conference - Composing the Innovation Symphony*.

Thornton, P. H., & Ocasio, W. C. (2008). Institutional logics. *The SAGE Handbook of Organizational Institutionalism*, 99–129. https://doi.org/10.4135/9781849200387.N4

Tiwana, A. (2015). Platform desertion by app developers. *Journal of Management Information Systems*, *32*(4), 40–77. https://doi.org/10.1080/07421222.2015.1138365

Vainio, N., & Vadén, T. (2012). Free software philosophy and open source. *International Journal of Open Source Software and Processes*, *4*(4), 56–66. https://doi.org/10.4018/ijossp.2012100105

Valiev, M., Vasilescu, B., & Herbsleb, J. (2018). Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem. *ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 644–655. https://doi.org/10.1145/3236024.3236062

Välimäki, M. (2002). Dual licensing in open source software industry. *Systemes d'Information et Management*, 8(1), 63–75. https://doi.org/10.2139/ssrn.1261644

van Angeren, J. , Kabbedijk, K., Popp, K., & Jansen, S. (2013). Managing software ecosystems through partnering. In J. Slinger, M. Cusumano, & S.

Brinkkemper (Eds.), *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry* (pp. 85–102). Edward Elgar Publishing.

Vidgen, R., & Braa, K. (1997). Balancing interpretation and intervention in information system research: The action case approach. *Information Systems and Qualitative Research*, 524–541. https://doi.org/10.1007/978-0-387-35309-8_26

Viseur, R., & Jullien, N. (2023). CommunesPlone: An original open source model of resource pooling in the public sector. *IEEE Software*, *40*(4), 46–54. https://doi.org/10.1109/MS.2023.3268352

Von Krogh, G., Haefliger, S., Spaeth, S., Wallin, M. W., & Zurich, E. (2012). Carrots and rainbows: Motivation and social practice in open source software. *MIS Quarterly*, *36*(2), 649–676.

Walsham, G. (1995). Interpretive case studies in IS research: Nature and method. *European Journal of Information Systems*, *4*(2), 74–81. https://doi.org/10.1057/EJIS.1995.9

Wang, J., Bao, L., & Ni, C. (2023). An empirical study of the Apache voting process on open source community governance. *In Proceedings of the 14th Asia-Pacific Symposium on Internetware (Internetware '23)*, 101–111, ACM, https://doi.org/10.1145/3609437.3609454

Wang, W., Mahakala, K. R., Gupta, A., Hussein, N., & Wang, Y. (2019). Data on security requirements in open-source software projects. *Data in Brief*, *25*, 103590. https://doi.org/10.1016/J.DIB.2018.12.02

Weber, S. (2004). *The Success of Open Source*. Harvard University Press. https://doi.org/10.4159/9780674044999/

West, J. (2003). How open is open enough?: Melding proprietary and open source platform strategies. *Research Policy*, *32*(7), 1259–1285. https://doi.org/10.1016/S0048-7333(03)00052-0

West, J., & Gallagher, S. (2006). Challenges of open innovation: The paradox of firm investment in open-source software. *R&D Management*, *36*(3), 319–331. https://doi.org/10.1111/J.1467-9310.2006.00436.X

West, J., & O'Mahony, S. (2008). The role of participation architecture in growing sponsored open source communities. *Industry and Innovation*, *15*(2), 145–168. https://doi.org/10.1080/13662710801970142

West, J., & Wood, D. (2013). Evolving an open ecosystem: The rise and fall of the Symbian platform. *Advances in Strategic Management*, *30*, 27–67. https://doi.org/10.1108/S0742-3322(2013)0000030005

Wolfenbarger, V., & Smith, J. (2023). How the public shaped the internet: Open-source software development and implementation through the years. *SAIS 2023 Proceedings*. https://aisel.aisnet.org/sais2023/25

Yang, Z. (2022). How censoring China's open-source coders might backfire. *MIT Technology Review*, May 2022.

Ye, Y., & Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. *International Conference on Software Engineering (ICSE 2003)*, 419–429. https://doi.org/10.1109/ICSE.2003.1201220

Yin, R. (1998). The abridged version of case study research: Design and method. In L. Bickman & D. Rog (Eds.), *Handbook of applied social research methods.* 229–259. Sage Publications.

Yin, R. K. (2009). *Case study research: Design and methods*. SAGE Publications.

Yu, Y. (2020). Role of reciprocity in firms' open source strategies. *Baltic Journal of Management*, *15*(5), 797–815. https://doi.org/10.1108/BJM-12-2019-0408

Yu, Y., Wang, H., Yin, G., & Wang, T. (2016). Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology*, *74*, 204–218. https://doi.org/10.1016/J.INFSOF.2016.01.004

Zanotti, A., & Vélez, J. G. (2020). Floss development and peer governance: the case of the gnome desktop environment. *International Journal of Innovation, 8*(3), 438–465. https://doi.org/10.5585/iji.v8i3.17114

Zhang, Y., Stol, K. J., Liu, H., & Zhou, M. (2022). Corporate dominance in open source ecosystems: A case study of OpenStack. *ESEC/FSE 2022 - Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1048–1060. https://doi.org/10.1145/3540250.3549117

Zhang, Y., Zhou, M., Stol, K. J., Wu, J., & Jin, Z. (2020). How do companies collaborate in open source ecosystems an empirical study of openstack. *Proceedings - International Conference on Software Engineering*, 1196–1208. https://doi.org/10.1145/3377811.3380376

Zhou, M., Mockus, A., Ma, X., Zhang, L. U., & Mei, H. (2016). Inflow and retention in OSS communities with commercial involvement. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *25*(2). https://doi.org/10.1145/2876443

ORIGINAL PAPERS

I

CONTRIBUTING TO ECLIPSE: A CASE STUDY

by

Katja Henttonen & Mari Matinlassi, 2007

Software Engineering 2007 – Fachtagung des GI-Fachbereichs Softwaretechnik 27.–30.03.2007 in Hamburg, pp. 59–70

https://dl.gi.de/items/fdb0a380-211e-437d-85a3-c11e0729436f

Contributing to Eclipse:A Case Study

Henttonen Katja, Matinlassi Mari
VTT Technical Research Centre of Finland
P.O. Box 1100, 90571 Oulu, Finland
{Katja.Henttonen, Mari.Matinlassi}@vtt.fi

Abstract: Open source software has gained a lot of well-deserved attention during the last few years. Eclipse is one of the most successful open source communities providing an open development environment and an application lifecycle platform. The main aim of this paper is to describe a case study on contributing to the Eclipse open source community and report experiences. The most important experiences are related to building an architecture model repository tool as an Eclipse plug-in and startinga new community around it.

1 Introduction

Open source software (OSS) has been growing its popularity among software developers, communities and media over a decade and lately also the research community has been active in studying the subject. Among the most successful open source projects, Eclipse (www.eclipse.org) is an open development platform and application framework for building software. The Eclipse community has distinguished itself in productivity and creativity [Ki05] and has developed new features that have evolved Eclipse towards a platform that is integrating not only tools but also applications and services [Gr05]. The new features also make Eclipse a strong force in the embedded market [Er06] [Kl05] and make the community even more international [La05].

There are several ways of contributing to open source communities, e.g. fixing bugs and providing new features. In this paper, contributing to Eclipse is considered as a twofold issue. On the one hand, a contribution, i.e. a plug-in, needs to be developed, and on the other hand, it needs to get published to the audience: users and community [Be04]. A good plug-in is modular, because the only way an open source project can mature is by allowing a number of people to participate in the development [Fl01]. While developing modular plug-ins is relatively straightforward [Ga04], the acute question remains how to get the modular plug-in to the open source "market"? The contribution of this paper is to introduce a case study on contributing to Eclipse and to report the experiences on the subject. The rest of the paper is structured as follows. The next section summarizes the approach used for contributing to Eclipse. After that the case study is introduced and the experiences discussed. Conclusions close the paper.

2 Approach for contributing to Eclipse

Open source projects have been likened to a bazaar [Ra03]. An open source project gathers people, whose skills, motivation and time of involvement may vary significantly [GaA04], to work in a distributed environment. Any contributor is welcomed to add a new feature to "scratch an itch". In such a development model, the importance of maintainability and the ease of making modifications are highlighted. This goal is achieved, e.g., by designinga modular software architecture.

In order to create a successful open source community around an Eclipse plug-in, the contributor needs (1) to build a plug-in with well designed software architecture, (2) to enable others to extend the plug-in with further plug-ins, and (3) to publish the plug-in as an open source project. The main stakeholders of the Eclipse community [Be04] are summarized in Table 1. In our approach for contributing to Eclipse, and in this case study, we are acting in three different stakeholder roles: extender, publisher and enabler. In this hierarchical stakeholder stack, the stack is built from a technical point of view. That is, publishing an Eclipse extension does not require making it extendable. Although this is true, attracting an active open source community, as stated above, requires easily expandable architecture, which in this case is implemented by Eclipse extension mechanisms.

Table 1. Stakeholders in the Eclipse community.

Stakeholder	**Description**
User	Uses Eclipse as it is.
Configurer	A user who customizes his/her experience of Eclipse within the limits envisioned by the original programmer.
Extender	A programmer who makes extensions by plugging in functionality.
Publisher	An extender who makes extensions available to others for loading them.
Enabler	A publisher who has defined one or more extension points for a plug-in, thus enabling others to extend the contribution.
Committer	Modifies the Eclipse code and incorporates changes into the global Eclipse release. Requires trust of existing committer community.

An open source project can be announced when the code artifact has been implemented into a "minimal but working version" [Go05]. Mandatory requirements, which need to be implemented before announcing the project, were identified in the requirement specification phase. In addition to the code artifact, an open source project needs a standard set of tools for managing information, including at least a website, mailing lists, a code versioning system and a real-time chat room [Fo05]. Once the infrastructure is ready, the next goal is 1) to get publicity for the project, 2) to gain a large enough user community, and 3) to gradually increase the level of community involvement. To be successful, an open source project requires marketing just like any other product. [Go05]

3 Case study: Stylebase for Eclipse

3.1. Overview

Quality-driven architecture design is an approach for software architecture design which emphasizes the importance of quality attributes (e.g. performance, modularity or maintainability) during the development[1]. The approach relies on the assumption that architectural styles and patterns, and also design patterns, embody different quality attributes. When patterns are applied in the architecture, the quality characteristics of the selected patterns are reflected in the entire software architecture. Stylebase is an important part of the quality-driven software architecture design and analysis (QADA®) methodology. Stylebase is a knowledge base of architectural styles and patterns [Ni05], and design patterns. The aim of the stylebase is to assist the architect in selecting styles and patterns which promote desired quality attributes. On the other hand, the stylebase may also assist in evaluating software architectures [Do02].

The Stylebase for Eclipse community has developed an open source implementation of the stylebase based on previous work [Me05] [MN05], and it has also published as a tool for browsing and maintaining the stylebase. The tool can be used in three ways: 1) as an electronic library for patterns, 2) as a guide for evaluating quality-driven architecture and 3) as a guide for quality-driven architecture modeling. When used as an electronic library, the architect browses the stylebase as a pattern catalogue. When used for model evaluation, the architect detects which patterns have been used in an architecture model and then checks from the stylebase which quality-attributes are associated with these patterns. When constructing a new architecture model, the architect searches the knowledge base and selects patterns according to the desired quality characteristics.

3.2. Expandable plug-in architecture

In the Stylebase for Eclipse, modularity was implemented at two levels. (1) The internal architecture of the Stylebase for Eclipse was designed to be modular. This was achieved by following the well-known model-view-controller pattern (MVC), see e.g. [Bu96]. (2) The Stylebase for Eclipse provides functionality which lets users build custom extensions without touching the source code of the Stylebase for Eclipse. This has been achieved by (a) building *access points* by implementing and exporting API (Application Programming Interface) packages, and (b) defining *extension points* with the Eclipse extension point mechanism. The access and extension points are listed in Table 2 and Table 3 respectively.

[1] http://virtual.vtt.fi/qada/

The MVC architecture (Model, View and Controller) is a way of breaking an application into three parts: model, view and controller. The user input, the manipulation of data and the visual feedback to the user are separated and handled by controller, model and view objects respectively. The MVC architecture was selected for the following reasons:

1) MVC promotes extensibility by allowing multiple representations (views) of the same information (model). This makes it easy to update the graphical user interface, which is especially prone to change requests, and/or to customize views based on user profiles.

2) MVC promotes code reuse by allowing a single view to show data from different models. By adding a new model, it is possible to adjust the tool to manage entirely different types of data with minimum recoding effort.

3) MVC facilitates maintenance by allowing an individual developer to focus on one aspect of the application at a time. Multiple developers can simultaneously update the interface, logic or input of an application without affecting other parts of the source code.

5) MVC architecture suits well for Eclipse plug-in development because the view classes of Eclipse can receive any other class as an input object. The Eclipse platform itself has a model-view-controller architecture [Gr04].

In order to increase the level of modularity, the three main components communicate with each other via predefined interfaces. Figure 1 shows how the plug-in implements a model-view-controller pattern.

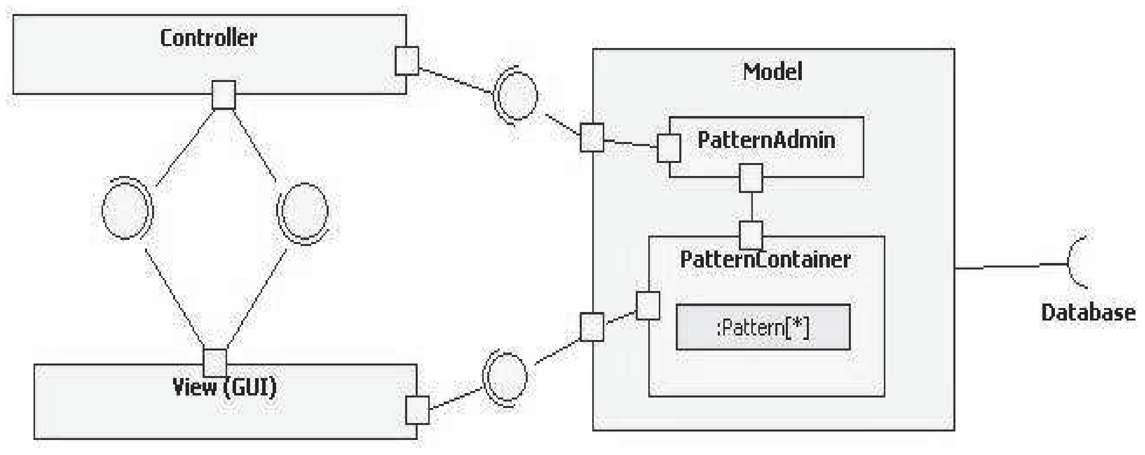


FIGURE 1: Stylebase plug-in composite structure

Tables 2 and 3 describe functionality that the Stylebase for Eclipse provides for downstream plug-ins.

Table 2. The access points provided by the Stylebase for Eclipse

Controller	The interface provides access to the control component. It lets users associate the functions of Stylebase for Eclipse with the GUI of another plug-in.
Model	The interface gives access to the model component. It provides a set of methods for retrieving and updating essential data in the Stylebase.
Database (SQL)	The interface provides SQL-level access to the underlying database. It helps in implementing specific functionality not provided by the model interface.

Table 3. The extension points provided by the Stylebase for Eclipse

Model Extension Point	The extension point provides the means of adding new models, units for storing and handling different types of data.
GUI Extension Point	The extension point provides the means of customizing the user interface of the Stylebase for Eclipse. It allows users to add their own views and/or menu items to the main view of the Stylebase for Eclipse. The controller component is extended respectively.

3.3. Founding a community

In the case of Stylebase for Eclipse, , the founding of the community was started right after implementing the mandatory functionality. Considering our limited resources and the laborious effort of self-hosting a project, it was decided to subscribe to one of the websites providing free hosting services for open source projects. Such services include web hosting, mailing lists, code archive, file hosting and bug tracking. For selecting hosting facilities, the most well-known, general-purpose hosting facilities – SourceForge.net, Savannah, BerliOS and Google code – were evaluated. They all run a collaborative software development management system originally called "SourceForge". SourceForge.net uses a closed-source proprietary version of this software, while the three other websites run free versions forked from the last open source version of SourceForge.

When registering a project to any such facility, the submitter has to state which license the project uses. We decided to use the GNU (**G**NU's **N**ot **U**nix) General Public License (GPL), which has been written by the Free Software Foundation. Many contributors are familiar with it as it is the most widely used open source license [Fo06]. GPL efficiently prevents creation of closed-source forks [Go05] and ensures that our software can be unambiguously used with MySQL client libraries which are also GPL licensed. The main drawback of GPL is that it may turn companies away from contributing to a project [Go05]. Another disadvantage is the incompatibility with EPL (Eclipse Public License) [Fr06] which, however, may be solved by GPL version 3 [Or06].

Savannah provided by the Free Software Foundation was selected as a project hosting provider because it is ad-free and it provides the most professional looking user- interface. Unlike the other alternatives, Savannah is dedicated to free software advocacy and has strict hosting policies, such as code review before accepting a new project as a Savannah project. The Savannah administration requires that all hosted software 1) is licensed under a free software license compatible with the GNU General Public License,
2) runs on at least one free operating system such as Linux, and 3) is not dependent on any non-free software. Non-free software includes all proprietary programs (such as Macromedia Flash and the Microsoft SQL Server), non-free file formats (such as the Graphic Interchange Format), and non-free programming languages (such as the Sun implementation of Java). [FrA06]

In order to express support to the Eclipse community, it was decided to apply for the membership of the Eclipse Foundation. Associate membership is free of charge for non-commercial entities like universities and research institutes. The Eclipse Foundation requires that associate members commit to 1) deliver added value to the Eclipse community within 12 months and 2) publicly announce joining the Eclipse Foundation within 90 days [EcA06]. While the associate membership does not grant decisive power, it grants rights to participate in project reviews, project creation and discussions on Eclipse intellectual property policy. It also entitles the member to use the Eclipse Foundation Member logo in marketing activities. [Ec06]

In order to make the project known to the public, we decided to publish announcements on some popular, open source related websites and to send one-time postings to carefully selected mailing lists. We chose not to send email to several, large mailing lists because such postings are easily considered as spam. As well as regular internet users, open source developers dislike irrelevant information that is blocking their communication channels [Ra06]. Further, we promoted the project at various open source related events, such as the OpenMind seminar (http://www.openmind.fi/). A brochure and poster were designed for marketing the project. We also contacted lecturers of software development courses at the University of Oulu and offered the Stylebase for Eclipse project as a practical work for their students.

The Eclipse Plug-in Central (EPIC) is obviouslya website where we wanted our plug-in to be listed. This is a place where Eclipse users search for commercial and open source plug-ins. An announcement was also placed at a few open source development portals, at FreshMeat.net, for example, which Fogel [Fo05] mentions as the number one place to be seen.

5 Experiences

5.1. Starting up the project- project hosting

As stated above (see Section 3.3), we evaluated four project hosting services and selected Savannah as the most appropriate one for our purposes. Savannah administration had very strict requirements on how the GNU General Public License should be applied to the code artifact. The requirements include placing copyright and free software statements both at the beginning of each source file and in a README file in every subdirectory containing binary files. According to the Free Software Foundation, this is the only way to ensure that the code is effectively protected by the license [FrB06]. What made things inconvenient was extending the requirement to apply to all external libraries distributed at Savannah. In the case of the Stylebase, this meant, for example, that one needed to open the JAR (**J**ava **Ar**chive) file of the MySQL client, placea README file into dozens of sub directories and then to package the source code and binaries back into a JAR file. Savannah also required that the source code of all external libraries was distributed with the plug-in. It was not sufficient to provide instructions on where to download the source code [RoA06].

Some interesting lessons were learned with different implementations of Java class libraries. Until November 2006, Sun Java libraries were published under Sun Community Source License (SCSL), which is a hybrid between an open source license and a traditional proprietary license [Ga6]. Many free software developers were unhappy with the Sun licensing scheme [So05] and Free Software Foundation coordinated the development of GNU Class Path, a GPL licensed implementation of Java. As the Stylebase plug-in was developed on Sun platform, some Sun-only features were inadvertently used. Later on, it was required to redo these features because the code could not be compiled with GCJ (GNU Compiler for Java), a widely-used open source Java compiler. In November 2006, as a response to many requests, Sun finally decided to publish its Java implementation under GNU General Public License [Su06]. Time shall show whether the decision will diminish or increase differences between Java implementations. In any case, making an open source product dependent of one particular implementation would hardly bea good choice.

The application process to Savannah took longer than planned. This was most probably due to the following reasons. Not all the requirements for application were clearly stated at once, but rather mentioned one after another each week. The voluntary nature [Go05] of open source communities was really put into action when our emails were replied to mostly on weekends and there was often a long delay before receiving a reply. Due to the strict deadline of publishing our open source project we had to set up a site at SourceForge.net where the application was processed and accepted in less than a day. Considering the amount of bogus projects found at SourceForge.net, the quick approval appears to reflect a loose hosting policy rather than efficiency.

5.2. Integrating OS components - license issues

At the time the Stylebase for Eclipse was about to be published, the Stylebase for Eclipse tool was dependent on an XML (eXtensible Markup Language) object model called JDOM, which was published under its own open source license specifically written for the JDOM project. Due to the fact that JDOM uses a permissive license similar to the MIT (Massachusetts Institute of Technology) license, we considered it to be GPL- compatible. However, Savannah refused to accept the JDOM component as a part of the project. The clause that prohibits the use of the name JDOM in derivative works can be seen as an additional restriction and this would rule out the GPL-compatibility [RoB06]. A polite posting [He06] to an open source forum regarding the GPL-compatibility of JDOM caused an intense "flame war". It seems that whenever a project uses its own license, compatibility issues area matter of debate. The JDOM component was therefore replaced with another document object model which had been published under the GNU Lesser General Public License (LGPL). The Open Source Watch [Os06] recommends that GPL-licensed software be combined with licenses that are listed as GPL-compatible on the website of the Free Software Foundation.

5.3. Attracting users and contributors - marketing activities

At the time of writing this paper, not all the marketing activities mentioned above (Section 3.3.) were executed. However, initial experiences were collected as follows. Applying for the membership of Eclipse Foundation was quite straightforward. However, while reviewing the membership agreement of Eclipse Foundation, it was unclear whether contributions would be accepted under any other open source license than the Eclipse Public License (EPL). The Eclipse Foundation confirmed that the strict EPL requirement applies only to contributions which are part of the official Eclipse project and hosted at eclipse.org [Sm06]. Soon after that, the application for Eclipse Foundation Associate Membership was accepted [Mc06].

The most effective marketing activity seemed to be giving a demonstration at a seminar targeted for special audience interested in open source. A concrete metric set for assessing the effectiveness of marketing activities was provided by the number of downloads at stylebase.sourceforge.net. A few days after the demonstration we were able to witness nearly 50 downloads from different IP addresses.

6 Conclusions

The aim of this paper was to describe a case study on contributing to the Eclipse open source community and to report some experiences. The initial experiences were reported soon after the official publication of the project in autumn 2006. The case study was concerned with a software architecture tool called Stylebase for Eclipse, which was licensed under the GNU GPL and implemented as an Eclipse plug-in. Developing Eclipse plug-ins was quite straightforward and therefore the most interesting experiences were related to (1) licensing issues while integrating several open source components together, (2) project hosting services while starting up an open source community, and
(3) marketing activities while attracting users and contributors for the community.

Based on the case study, the most important conclusions are as follows. First, Savannah by the Free Software Foundation was selected as a project hosting provider because it was ad-free and provided the most professional looking user-interface. One of the experienced advantages of Savannah was that it would host only carefully reviewed projects (causing a disadvantage of long acceptance process). We also tried out another option - SourceForge – where, although the acceptance process was very fast, the responsibility of project quality was left for the respective community.

Second, it seems that whenever an open source subcomponent uses its own license written especially for the project, compatibility issues will be a matter of debate. Resolving compatibility issues takes time and therefore should be taken into account in schedules, or in our case regarding a GPL license, one should only use open source components with GPL compatible licenses accepted by the Free Software Foundation.

Third, the marketing of an open source project was deemed to be crucial in order to gain users and contributors for your project. We used several marketing channels such as passing information down by word of mouth, handing out brochures, and demonstrating the proposed tool at a seminar for open source related audience. Marketing activities caused an immediate tenfold increase in code downloads at the project website.

Acknowledgements

The publication of this paper has been supported by Eureka ITEA research project COSI funded by the National Technology Agency (Tekes) and VTT.

References

[Be04] Beck, K.; Gamma, E.: Contributing to Eclipse. Dr. Dobb's Journal. 29, 9, 2004; P. 74-8.

[Bu96] Buschmann F; Meunier R; Rohnert H; Sommerlad P; Stal M: Pattern-oriented software architecture - a system of patterns. Wiley, 1996.

[Do02] Dobrica, L.; Niemela, E: A survey on software architecture analysis methods. Software Engineering, IEEE Transactions on 28(7): 638-653. 2002.

[Ec06] Eclipse Foundation: Eclipse Rights by Membership Category http://www.eclipse.org/membership/become_a_member/How2Join%20Eclipse%20Rights%20by%20Membership%20Category.pdf. Visited 26.10.2006.

[EcA06] Eclipse Foundation: Eclipse Membership Application or Change in Representation. http://www.eclipse.org/membership/become_a_member/Membership %20Application.pdf. Visited 26.10.2006.

[Er06] Erickson, J: Eclipse Foundation Releases Embedded RCP. The World of Open Source. Available at http://www.ddj.com/blog/opensourceblog/archives/2006/10/eclipse_release.html

[Fl04] Fleury, M.; Lindfors J.: Enabling component architectures in JVMX. http://www.onjava.com/pub/a/onjava/2001/02/01/jmx.html. 2001. Visited 30.6.2006.

[Fo05] Fogel, K.: Producing Open Source Software. How to run a successful free software project. O'Reilly, 2005; P. 34, 43, 45-47.

[Fr06] Free Software Foundation: GPL-incompatible Free Software Licences. http://www.fsf.org/licensing/licenses/index_html#GPLIncompatibleLicenses. Visited 20.10.2006.

[FrA06] Free Software Foundation: Savannah Services and Requirements https://savannah.nongnu.org/register/requirements.php Visited 26.10.2006

[FrB06] Free Software Foundation: Frequently Asked Questions on GNU GPL: Using the GPL for your programs. http://www.gnu.org/licenses/gpl-faq.html. Visited 27.10.2006.

[Ga04] Gamma, E.; Beck, K.: Contributing to Eclipse - Principles, Patterns, and Plug-Ins. Addison Wesley, 2004; P. 395.

[GaA04] Gacek, C.; Arief, B.: The many meanings of open source. IEEE Software. 21, 1, 2004; P. 34-40.

[Ga06] Gabriel,R; Joy, W: Sun Community Source Licensing (SCSL) – Principles. http://www.sun.com/software/communitysource/principles.xml. Visited 26.10.2006.

[Go05] Goldman R.; Gapriel R: Innovation Happens Elsewhere. Open Source as Business Strategy. Elsivier, 2005; P. 15-16, 157, 190-191, 224, 256.

[Gr04] Griffin, C: Transformations in Eclipse. The proceedings of 18th European Conference on Object-Oriented Programming. Norway, 2004.

[Gr05] Gruber, O. et al.: The Eclipse 3.0 platform: adopting OSGi technology. IBM Systems Journal. 44, 2, 2005; P. 289-99.

[He06] Henttonen, K.: Incompatibility with GPL [email]. Message to: jdom- interest@jdom.org. 28.9.2006. Cited 26.10.2006. Available at http://www.jdom.org/pipermail/jdom-interest/2006-September/015549.html

[Ki05] Kidane, Y.; Gloor, P: Correlating Temporal Communication Patterns of the Eclipse Open Source Community with Performance and Creativity. NAACSOS Conference, June 26 - 28, Notre Dame IN, North American Association for Computational Social and Organizational Science, 2005. Available at http://www.ickn.org/documents/Naacsos_Kidane_Gloor.pdf

[La05] Lammers, D.: Tool developers rally around Eclipse. Electronic Engineering Times. 1369, 2005; P. 47-49.

[Mc06] McGaughey, S.: Welcome VTT Technical Research Center of Finland to Eclipse Foundation as an Associate Member [email]. Message to Katja Henttonen. 4.10.2006. Cited 1.11.2006.

[Me05] Merilinna, J: A Tool for Quality-Driven Architecture Model Transformation. Espoo, VTT. VTT Publications. 2005. http://www.vtt.fi/inf/pdf/publications/2005/P561.pdf

[MN05] Merilinna, J; Niemelä, E: A stylebase as a tool for modelling of quality- driven software architecture. Proceedings of the Estonian Academy of Sciences. Engineering December 2005. Special issue on Programming Languages and Software Tools. vol. 11, 4. 2005

[Ni05] Niemelä, E.; Kalaoja J. : Toward an architectural knowledge base for wireless service engineering. IEEE Transactions on Software Engineering 31(5): 361 - 379. 2005.

[Os06] Open Source Watch: What is open source software? http://www.oss-watch.ac.uk/resources/opensourcesoftware.xml. Visited 27.10.2006.

[Or06] O'Riordan, C: The Transcript of the Speech by Eben Moglen (Section 7e) at the Opening Session of the First International GPLv3 Conference on January 16th 2006. Available at http://www.ifso.ie/documents/gplv3-launch-2006-01-16.html.

[Ra03] Raymond, E: The Cathedral and the Bazaar. O'Reilly, 2001. Available at http://catb.org/~esr/writings/cathedral-bazaar/.

[Ra06] Raymond, E.; Moen R: How to Ask Smart Questions The Smart Way. http://catb.org/esr/faqs/smart-questions.html. Visited 27.10.2006.

[Ro06] Robson, S.: [task #5865] Submission of Stylebase [email]. Message to Katja Henttonen and savannah-register-public@gnu.org. 24.9.2006. Cited 26.10.2006. Available at http://www.mail-archive.com/savannah-register-public@gnu.org/msg06275.html

[RoA06] Robson, S.: [task #5865] Submission of Stylebase [email]. Message to Katja Henttonen and savannah-register-public@gnu.org. 12.10.2006. Cited 26.10.2006. Available at http://www.mail-archive.com/savannah-register-public@gnu.org/msg06347.html

[RoB06] Rowan, W.: Open Source Development - An Introduction to Ownership and Licensing Issues. University of Oxford, 2006. Available at http://www.oss-watch.ac.uk/resources/iprguide.xml.

[Sm06] Smith, D.: RE: Question on Eclipse membership [email]. Message to: Katja Henttonen. 2.10.2006. Cited 26.10.2006.

[So05] Souza, B.: How Much Freedom Do you Want? In compilation Cris Dibona & al. (edit): Open Sources 2.0. O'Reilly, 2005; P219-224.

[Su06] Sun MicroSystems Inc. Sun Open Sources Java Platform and Releases Source Code Under GPL License Via NetBeans and Java.net Communities. http://www.sun.com/smi/Press/sunflash/2006-11/sunflash.20061113.1.xml. Visited 28.12.2006.

II

OPEN SOURCE BASED TOOLS FOR SHARING AND REUSE OF SOFTWARE ARCHITECTURAL KNOWLEDGE

by

Katja Henttonen & Mari Matinlassi, 2009

2009 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) & European Conference on Software Architecture (ECSA), pp. 41–50
https://doi.org/10.1109/WICSA.2009.5290790

Open Source Based Tools for Sharing and Reuse of Software Architectural Knowledge

Katja Henttonen, Mari Matinlassi
VTT Technical Research Centre of Finland
katja.henttonen@vtt.fi

Abstract

Sharing and reuse of software architectural knowledge (SHARK) has become an emerging topic of discussion and research in the field of software architecture development. SHARK is efficient with tool support, particularly so when that support is appropriate. However, there seems to be little guidance for selecting a suitable SHARK tool for use in an organization. The contribution of this paper is twofold. First, we present an evaluation framework that can be used for presenting an overview of SHARK tool features or comparing their differences. Secondly, we use the presented framework to evaluate three publicly available SHARK tools. The evaluated tools are the following: WebOfPatterns, Stylebase for Eclipse and PAKME.

1. Introduction

The complexity and size of software systems and system families has increased and the stakeholders involved in software architecture are many. When software is developed in a multi-site global company or in an open source community, the stakeholders are also geographically distributed. For these reasons, SHARK has become an increasingly important topic. Storing, reusing and sharing architectural knowledge, both from a design and design decision perspective, requires appropriate tool support.

While several papers [1, 3, 4, 5] state many desirable features for a SHARK tool, a common framework for evaluation of SHARK tools has not been provided. Ihme et al. [20] present an evaluation framework aimed at computer aided software engineering tools. However, this framework is quite general i.e. not targeted especially at SHARK tools and, on the other hand, too detailed to be used for presenting an overview of tool features. The NIMSAD (Normative Information Model-based Systems Analysis and Design) evaluation framework [21] has been presented earlier. This framework has been developed for evaluating methods in any category, but it can also be adapted for evaluating tools.

Some excellent literature studies on the state-of-the-art of SHARK tools have already been presented [see e.g. 12]. However, there is obviously a substantial difference in what functionality has been outlined in literature and what has been already implemented and made available to users. Some industrial studies on Enterprise Architecting tools [e.g. 25,28] touch model sharing features briefly, but focus on business (instead of software) architecting. Therefore, the participants of the 2nd SHARK workshop agreed that a more empirical analysis of software architectural knowledge sharing solutions is required [1].

The contribution of this paper is to present an evaluation framework for SHARK tools and then use the framework to evaluate three open source based solutions. The framework is based on the frameworks introduced above. The topics addressed in the framework have been widely recognized [e.g. 1-5] as desirable features for a SHARK tool. Evaluation is entirely based on public and free material available in the web pages of open source projects. The evaluation was performed by installing and using the case tools.

The following open source tools were selected for evaluation: Web of Patterns, Stylebase for Eclipse and PAKME. These tools were selected because they are general purpose (i.e. not limited to a particular programming language or platform) and currently in active development. The study uses open source tools to exercise the framework, because they are publicly available, thus making the evaluation more transparent [15]. The open source approach brings also many benefits to users, such as easy customization [14]. Solutions, which are only available as research papers, were excluded due to the empirical nature of the study.

The remaining parts of this paper are organized as follows. First, a framework for evaluating the SHARK tools is presented with a rationale and background of question categories, criteria and evaluation questions. After that, each case tool is introduced and its evaluation data provided. In the end, evaluation results and framework capabilities are discussed. Conclusions close the paper.

2. Framework for Evaluation

2.1 Question Categories

The evaluation framework for tools that support sharing and reuse of architectural knowledge is presented in Table 1. Categories of the framework are based on the NIMSAD evaluation framework [21], the CASE tool evaluation criteria of Ihme et al. [20] and results of the second SHARK working group [1]. The evaluation questions have been defined based on a large body of previous work from an active research community [e.g. 2, 11-13]. In line with NIMSAD, the entire problem solving process is used as the basis of evaluation. Our framework proposes four view points to the evaluation of SHARK tools (Figure 1) as follows.

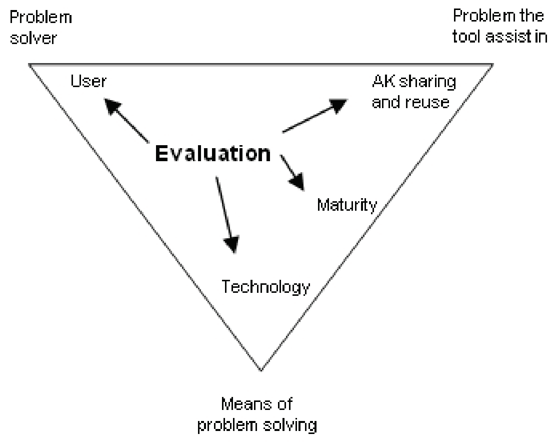


Figure 1: Elements of SHARK tool evaluation

Problem the tool assists in. Reuse and sharing of software architectural knowledge means capturing it in a reusable form and sharing it with others a meaningful way. This is the problem a SHARK tool solves. The questions in this category evaluate how well the tool covers the problem area of SHARK.

Problem solver. Problem solver is usually the tool user. In the case of knowledge sharing tools there are groups of users from several interest groups. This category of questions evaluates the tool from the viewpoint of problem solver e.g. learning, ease of use and installation issues.

Means of problem solving. SHARK tools may be implemented with several technologies. Technology selections may either restrict or open up tool features. The questions in this category evaluate (1) what kind of technological and implementation solutions have been selected for the tool and (2) how well these solutions serve the problem areas defined.

Maturity of the tool. The maturity of any tool is an important aspect for decision makers when selecting a tool for an organization. This category of questions evaluates the maturity of the tool.

2.2 Question criteria and evaluation questions

Knowledge reuse is about using the same knowledge more than once. Architectural knowledge (AK) reuse is the most efficient when first storing knowledge and then retrieving knowledge from the repository in order to use, reuse and maintain it. This kind of repository also enables sharing of AK.

According to [1, 22, 29, 30] architectural knowledge consists of the *design solutions*, i.e. how the system is designed, plus the *design decisions* and their rationales i.e. why the system is designed the way it is. As we know, software architecture has many definitions and many meanings depending on the context e.g., do we record the software architecture of a small application, family of applications or family of embedded software-intensive systems? Therefore, representation of architectural design knowledge may have to be done on several *abstraction levels* to present all the relevant information. Examples of abstraction levels include conceptual architecture, concrete architecture and detailed design-level knowledge. The second SHARK workshop aimed to specify what architectural decision documentation *must include* and what it *can include* in addition to must-to-have documentation [1, 4]. In summary, architectural design decisions documentation should describe what is decided, why the decision was made and what the discarded options were. The above mentioned issues of storing knowledge are addressed in questions 1-4 of the framework in Table 1.

The previously mentioned SHARK working group also considered tool support for architectural knowledge management. Firstly, *searching the knowledge has to be efficient* and users need to find what they are looking for [1]. Finding reusable knowledge may be difficult for various reasons [16, 30] e.g. knowledge utilizers may be unaware that the knowledge they need is available through a tool. Secondly, the essential information may be only implicit in the repository. In the end, the knowledge may have to be reconfigured in some way in order to meet the requirements of the task in hand. However, knowledge retrieval features of SHARK tools should help with at least some of these problems. Free text searches and first-order predicates are common approaches to querying AK data. [1]. In addition to conventional search functions, SHARK tools should provide mechanisms for navigating through architectural knowledge [4,5,17]. This means navigating between abstraction levels (e.g. by graphical zooming) and through decision paths (e.g. by tree branching or

Table 1: Evaluation framework for SHARK tools

Category	Criterion	Evaluation questions
Problem the tool assist in	Knowledge reuse	1. How does the tool support storing design solutions
		2. How does the tool support storing design decisions and their rationales
		3. Can the tool store both accepted and discarded design decisions?
		4. What is the abstraction level of stored knowledge?
		5. What kind of search mechanisms does the tool support?
		6. Does the tool support navigating through abstraction hierarchy / decision paths? How?
		7. Does the tool provide descriptive assistance (instead of imposing solutions)?
		8. What knowledge is modelled explicitly in the tool?
	Knowledge sharing	9. Does the tool support a shared repository with multi-user access?
		10. Does the tool support collaborative interactions or integrate with collaborative SW?
		11. Does the tool indicate the author of the knowledge (encouraging personal contacts)?
Problem solver	Target user group	12. What is(are) the target user group(s) of the tool?
		13. Can users get information on the granularity level of their choice?
	Target environment	14. How does the tool aim to integrate into the architecting process?
		15. Can templates be customized to accommodate organizational needs? How?
	Required resources	16. How much effort does it require to learn to use the tool?
		17. How much extra effort does it require to create and update the knowledge?
		18. What resources does it require to deploy the tool in an industrial organization?
Means of problem solving	Implementation technologies	19. What technologies have been used in the tool?
		20. In which format is the knowledge stored?
		21. Can the knowledge be imported/exported in some other formats?
		22. How well do the selected technologies/formats serve the problem of the tool?
	Openness	23. To what extent is the tool open (i.e. licensing terms)?
		24. Is the tool integrable with other tools? Which tools? How?
	Security	25. Does the tool implement access control? How?
Maturity	History	26. How long has the tool (and/or its vendor) been on the market?
		27. Has the tool been successfully used in an industrial setting? Where?
	Support	28. What kind of tool documentation is provided?
		29. What kind of support is available in using the tool?

hyperlinks). These issues of retrieving knowledge are addressed in questions 5 and 6.

A strong tendency in AK management seems to be that SHARK tools are not meant to replace people [1, 2, 11, 12]. According to the SHARK working group [1] "*the tool itself should not be prescriptive*, or even advisory, in its capacity to support the architecting process." Since architecting is essentially an art form, the tool should not limit the creativity of a software architect [12]. Experience has shown that "best practices" are found by peer-to-peer discussions rather top-down imposition of compliance rules [1]. This issue is taken into account by question 7.

Other important issues to consider when selecting a SHARK tool are *what knowledge is modelled explicitly in* the tool and how the modelled knowledge is tied to the development process [1]. The tool may derive e.g., from the background of modelling, process analysis or programming [1, 8] – and this impacts on how it attaches itself into the development process. The working group agreed that a SHARK tool should integrate with existing processes and tooling [1]. Such integration makes the tool more intuitive to use and thereby lowers the learning curve [1,19]. It also helps AK management to be seen as part of the "normal" architecting process, rather than as some extra, resource-consuming activity [1,30]. These issues are addressed in questions 8, 14 and 23.

Furthermore, the architecting environment dictates *what knowledge is worth storing* [11,30]. Data structures and templates should therefore be flexibly customizable to accommodate an organization's needs [1, 11]. The relevance of the knowledge also depends on the audience [1, 11, 12]. For example, a developer is usually interested in much more detailed architectural knowledge than a project manager [12, 18]. Therefore, one has to identify the stakeholders of the architecting process (i.e. programmers, designers, architects, managers) and ensure that they can access the knowledge at the desired granularity level [11, 12]. These issues are covered by questions 12, 13, 15.

In addition to knowledge reuse, the *sharing of architectural knowledge* is another main problem that SHARK tools shall address. A central requirement is that a tool supports a shared knowledge repository which can be accessed by multiple users and tools simultaneously [13, 33]. This approach enables easy retrieval and reuse of solutions that have proven themselves in the past [11]. However, industrial experiences have proven that such repositories may turn into "information junk yards" if social incentives between people are not encouraged [2]. In addition to making knowledge available in repositories (*codification*), a SHARK tool should help people to communicate (*personalization*) [1,2,12]. In order to enable personal communication, the tool should identify the author of the knowledge [2, 12]. Furthermore, social incentives can be encouraged by collaborative features, such as discussion boards or voting [2, 10]. User feedback received by us [17] also suggests that social networking support is at the top of the "wish list" of many SHARK tool users. These knowledge sharing features are considered in questions 9-11.

Ease of insertion and learnability are mentioned as important CASE (Computer Aided Software Engineering) tool evaluation criteria in [20]. Installation of a tool and its related learning curve sometimes are key issues among busy users. According to our experiences in developing open source tools [17, 18], a high threshold for installing a tool may prevent using it at all. The easy manipulation of content improves the user's commitment to use the tool and helps to keep architectural decision-making up to speed [12]. However, acquiring architectural knowledge from architects' heads takes some time, no matter how good the tool is. These issues are considered in questions 16-18 as *required resources* criteria.

The purpose of the *implementation technologies* criterion is to give an overview about technologies that are selected to implement the tool for a certain problem area (described in knowledge sharing and reuse criterion above). Technology selections affect several issues e.g., use of the tool, integrability with other tools and tool adaptability for a specific environment. A tool's input and output data format provide interesting information for deciding how to further utilize AK provided by the tool. These issues are addressed in questions 19– 22.

According to the earlier interviews conducted in the industrial sector [26, 27, 31], companies want to remain tool vendor neutral, whenever possible. Tool vendor neutrality decreases risks. Openness in software tools is a major trend today [14, 31] and therefore, the *openness* issue is considered in question 24. Openness also relates to the previously discussed issues of integrability and ease of customization. Like with any knowledge-intensive software tool, data security must be also considered [20,29]. Question 25 addresses access control.

Maturity of the tool is a criterion that should provide a quick overview of the professional nature of a SHARK tool. With regard to the maturity of a CASE tool, Ihme et al [20] emphasize its history, documentation and available support services (e.g. user groups, maintenance, help lines). These issues are covered in questions 26-29.

3. Web of Patterns

The WebOfPatterns[1] (WOP) [8,9] consists of two parts: (1) a language neutral format to describe design patterns and micro-architectures and (2) a set of Eclipse[2] extensions which use this format. WOP facilitates searching pattern instances in Java projects, searching patterns from online repositories, publishing one's own patterns and rating patterns published by others.

3.1 Problem the tool assists in

(Question 1:) A formal, machine-processable pattern language is used to define design patterns and related concepts such as pattern instance, pattern participants and pattern refinement. The data is presented in a table form as depicted in Figure 2. *Question 2:)* The general design rationale behind a pattern can be briefly stated in a designated field. *(Question 3:)* The tool supports storing several version of each pattern, which enables storing discarded or alternative solutions. *(Question 4:)* The tool can store concrete software designs, i.e. solutions that address coding problems.

(Question 5:) The tool supports the discovery of pattern instances from source code and online repositories can be searched for pattern definitions. Complex set of rules can be defined in order to locate desirable patterns. *(Question 6:)* Because the tool focuses only on micro-architectures, there is no need to navigate between abstraction levels. Navigation through decision paths is not supported. *(Question 7:)* The tool is advisory in nature. *(Question 8:)* The shared knowledge is not explicitly modelled in the tool, but retrieved from source code instead.

(Question 9:) The tool supports access to multiple shared repositories which all can be managed in the same view *(Question 10:)* The users can provide free text feedback on each pattern and bookmark them, wherein the number of bookmarks can be seen as a quality indicator. The tool also integrates with collaborative web services such as Blogger. *(Question 11:)* The tool identifies both the publisher and the original author of the knowledge.

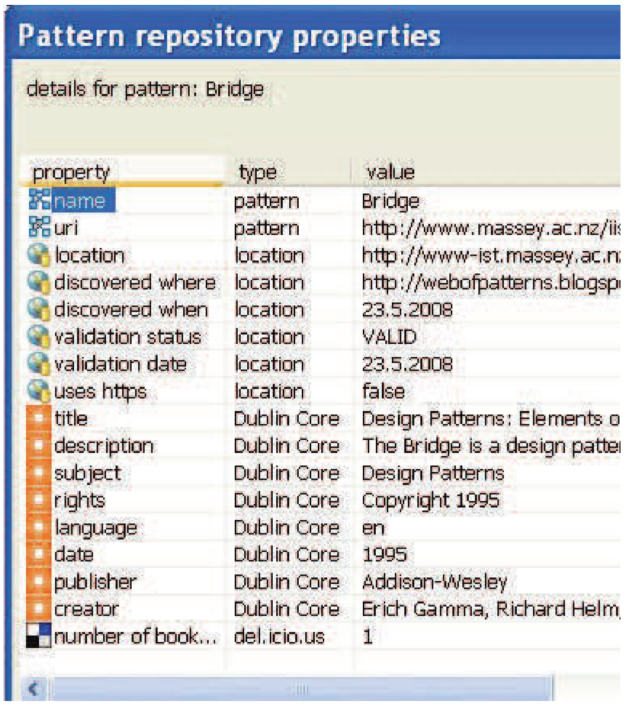
3.2 Problem solver

(Question 12:) The tool is aimed primarily at one user group – programmers. *(Question 13)*: Therefore, there is no need to filter the knowledge by abstraction level. *(Question 14:)* The tool integrates into programming

[1] http://www-ist.massey.ac.nz/wop/

[2] http://www.eclipse.org

work: new patterns are published from source code and source code is analyzed by identifying used patterns. However, there is no way to automatically transform pattern definitions into code. (*Question* 15:) The tool requires that patterns are defined in a particular language developed for the purpose and, in this sense the tool is not flexible. However, the underlying pattern language can be extended to accommodate special needs.



Pattern repository properties

details for pattern: Bridge

property	type	value
name	pattern	Bridge
uri	pattern	http://www.massey.ac.nz/ii
location	location	http://www-ist.massey.ac.n
discovered where	location	http://webofpatterns.blogsp
discovered when	location	23.5.2008
validation status	location	VALID
validation date	location	23.5.2008
uses https	location	false
title	Dublin Core	Design Patterns: Elements o
description	Dublin Core	The Bridge is a design patte
subject	Dublin Core	Design Patterns
rights	Dublin Core	Copyright 1995
language	Dublin Core	en
date	Dublin Core	1995
publisher	Dublin Core	Addison-Wesley
creator	Dublin Core	Erich Gamma, Richard Helm,
number of book...	del.icio.us	1

Figure 2: Viewing pattern properties in WOP

(Question 16:) The user interface is intuitive and learning to use the tool takes only a quarter of an hour. Getting to know the pattern definition language is time consuming, but not required for basic use. *(Question 17:)* It takes only a few mouse-clicks to create a new pattern by identifying relevant elements in the source code. *(Question 18:)* When deploying the tool, one needs to install client software on each workstation and store pattern definitions on web server. Client installations can be done remotely with operating system commands.

3.3 Technology

(Question 19:) The client has been implemented as an extension to the Eclipse IDE; standard web technologies are used to distribute pattern definitions. *(Question 20:)* Pattern definitions are stored in the RDF/OWL format. *(Question 21:)* The results of a pattern scan can also be aggregated and exported in the XML format to create reports. *(Question 22:)* The Eclipse environment helps to integrate the tool into the programmers' daily work. The simple, file-based distribution approach facilitates easy integration with search engines and collaboration tools such as CVS or Blogger. Compared to UML, the OWL-based approach facilitates more precise pattern description, which is required to underpin design recovery.

(Question 23:) The tool can be integrated with Web services and Eclipse-compatible OWL editors. *(Question 24:)* The tool is open source, apparently complying with the Open Source Definition[3], but the choice of license is not indicated. *(Question 25:)* Access control is not explicitly supported. However, standard web technology can be used to control access to repository servers.

3.4 Maturity

(Question 26:) The first code artifact was published in 2004 and a few major releases have been issued afterwards. *(Question 27:)* According to our knowledge, the tool has not been deployed in any industrial organization. *(Question 28:)* Related research papers are freely available on the Internet, but there is little other documentation. The provided online instructions are brief, but clear. *(Question 29):* The WOP mailing list is active and support requests are attended to quickly. The support seems to rely mostly on one person, however.

4. Stylebase for Eclipse

Stylebase for Eclipse[4] [17,18] is a tool for creating, maintaining and searching a software architectural knowledge base, aka stylebase. Stylebase can store different architectural models, e.g. architecture patterns, design patterns and reference architectures. In addition to the knowledge sharing function, the tool aims to guide architects in selecting solutions which best provide a system's desired quality goals.

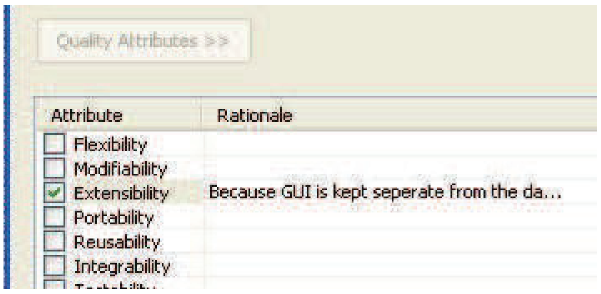
4.1 Problem the tool assists in

The solution concentrates on reusable models as a form of architectural knowledge. *(Question 1:)* Each model is associated with the following information: basic properties (e.g. name, abstraction level), picture, model structure (a mark-up language document), usage instructions (a free text document) and a set of quality attributes. *(Question 2:)* When a model is associated with a quality attribute, one is asked to enter a design rational as shown by Figure 3. *(Question 3:)* A more detailed discussion of design decisions, including discarded options, can be included in the free text document. *(Question 4:)*. The tool can store knowledge at

[3] http://www.opensource.org/docs/definition.php

[4] http://stylebase.tigris.org

any abstraction level; example contents consist of both macro and micro architectures.



Attribute	Rationale
☐ Flexibility	
☐ Modifiability	
✓ Extensibility	Because GUI is kept seperate from the da...
☐ Portability	
☐ Reusability	
☐ Integrability	

Figure 3: Giving a design rational in Stylebase

(Question 5:) Models can be searched by name, intent, abstraction level and quality attributes (see Figure 4). Search criteria can be constructed from multiple properties, which are joined by using AND operation. *(Question 6:)* The current version of the tool does not implement any navigation mechanism; nor does it support the presentation of relationships between design decisions. *(Question 7:)* The tool is completely advisory in nature, i.e. architects decide on whether and how to use the provided knowledge. *(Question 8:)* Most knowledge is modelled in third-party tools and only imported into the stylebase.

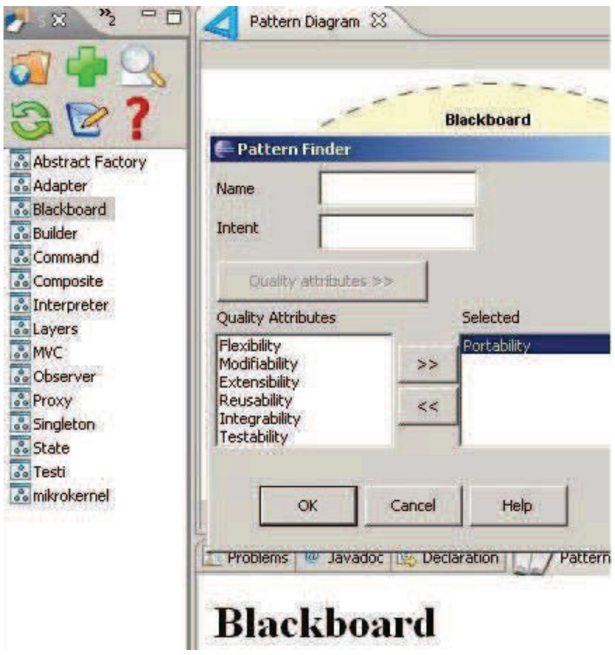


Figure 4: Searching patterns in Stylebase

(Question 9:) The tool supports a shared repository, uses locking to control simultaneous updates and provides a mechanism for uploading and downloading models between local and shared repositories .*(Question 10:)* The tool does not support collaborative interactions, all communication happens through the shared knowledge repository. *(Question 11:)* The tool does not help to identify the author of the knowledge.

4.2 Problem solver

(Question 12:) The tool is primarily aimed at software architects and designers, but programmers could also benefit from the software design knowledge. *(Question 13:)* There is a possibility to define user roles, which allow the filtering of knowledge based on abstraction level.

(Question 14:) Knowledge maintenance is intended to happen simultaneously with modelling. When a software architect creates or updates models in an Eclipse-compatible modelling tool, they can be automatically saved to the local knowledge repository. *(Question 15:)* The tool supports the customization of knowledge base contents to a large extent: first, the user can determine the contents and appearance of the free text document and, second, the tool imposes no rules on the internal structure of stored models.

(Questions 16:) Learning to use the tool takes 20 minutes with an instructor or 1-2 hours by self-study, providing that a person is familiar with Eclipse. *(Question 17):* Knowledge maintenance is laborious, mostly for two reasons. Models must be downloaded from the shared repository and saved locally before they can be updated. Text document must be manipulated with an HTML editor, which is not very convenient. *(Question 18:)* When deploying the tool, one needs to install database on a server and client software on each workstation. Client installations can be done remotely.

4.3 Technology

(Questions 19:) The tool has been implemented in Java as a plug-in to the Eclipse IDE; the repositories are based on relational database technology and are queried with SQL. *(Questions 20:)* Each record in the database has three large fields: a picture of a model in binary format (JPG/PNG/GIF), a guide text in format understood by an internet browser (typically HTML) and the model's structure in a mark-up language (typically XML). *(Questions 21:)* While the tool supports the storage of knowledge in heterogeneous formats, the knowledge can only be exported in exactly the same format in which it was stored.

(Questions 22:) The Eclipse platform brings the tool closer to the developers, but also creates distance to architects who don't work with Eclipse on a daily basis. The relational database approach facilitates quick and sufficiently powerful searches. The heterogeneous documentation formats allow maximum integrability, but also cause a problem: users of the same shared repository

need to explicitly agree in which format they store the knowledge.

(Question 23) Because the tool supports diverse documentation formats and provides many extension points, integration with Eclipse-compatible tools is straightforward. *Question 24)* The tool is open source and published under the GNU General Public License[5]. *(Question 25):* The tool exploits the MySQL access control mechanisms. Username and password are requested upon each connection to the remote database.

4.4 Maturity

(Question 26:) The first code artifact was published in October 2006 and five other, rather small, releases have been issued by June 2009. *(Question 27:)* The tool has not been deployed in any industrial organization. *(Question 28):* User guides are available on the Internet, along with a variety of technical documents. *(Question 29):* Mailing lists are inactive, but there are other support channels such as email and IRC. Chargeable services (e.g product customization) are also offered.

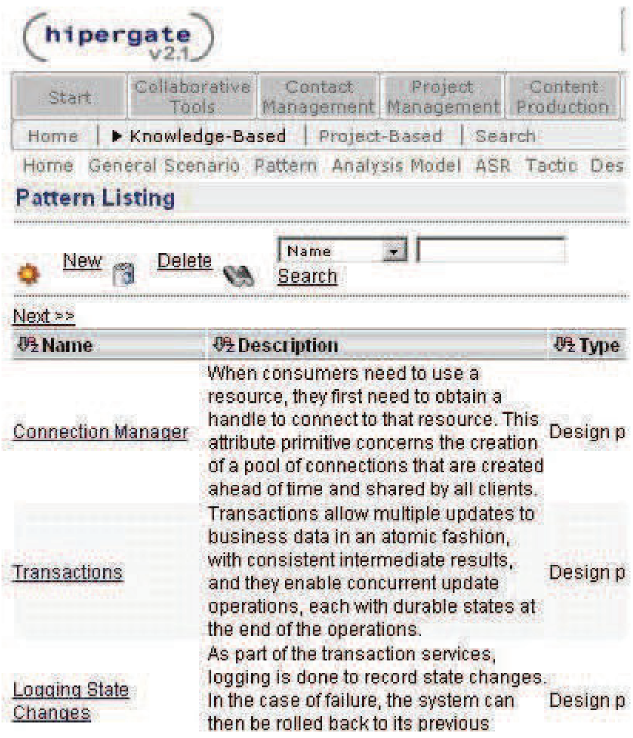
5. Pakme

PAKME (Process-based Architecture Knowledge Management Environment) [3] is a process-based SHARK tool, which has been built on top of an open source groupware platform called Hipergate[6]. The solution supports storing both contextual knowledge (e.g. design rationales, requirements) and technical knowledge (e.g. patterns, styles, tactics).

5.1 Problem the tool assists in

(Question 1:) PAKME supports storing architecting scenarios, architecture-specific requirements, analysis models, patterns and tactics. The tool uses forms for entering and presenting the knowledge (see Figure 5). *(Question 2:)* The tool supports defining "design option cases", which contain a description (e.g. patterns, tactics) and a rational. The design decision is a selected design option. *(Question 3:)* Each design decision can be linked to discarded design options. *(Question 4:)* The information can be stored at any abstraction level.

(Question 5:) The tool has a mechanism for searching scenarios, patterns, analysis models, requirements, tactics and design options. One can search them by name or, alternatively, search for a string in a given field (e.g. description). Logical operators can be used in advanced searches. *(Question 6:)* The web-based tool uses hyperlinks to navigate through architectural structures and decision paths (see Figure 5). *(Question 7:)* The tool is essentially advisory in nature and doesn't impose solutions. Some features provide descriptive support in the decision-making process. *(Question 8:)* Practically all knowledge is modeled explicitly in the tool.



Name	Description	Type
Connection Manager	When consumers need to use a resource, they first need to obtain a handle to connect to that resource. This attribute primitive concerns the creation of a pool of connections that are created ahead of time and shared by all clients.	Design p
Transactions	Transactions allow multiple updates to business data in an atomic fashion, with consistent intermediate results, and they enable concurrent update operations, each with durable states at the end of the operations.	Design p
Logging State Changes	As part of the transaction services, logging is done to record state changes. In the case of failure, the system can then be rolled back to its previous	Design p

Figure 5: Pattern listing in PAKME

(Question 9:) As the tool is based on a groupware solution, the repository is on a web server and accessible by multiple users. *(Question 10:)* PAKME can be integrated with collaboration features of the underlying Hipergate platform. *(Question 11:)* The current version of the tool does not identify the author of the knowledge, except in the case of architecting scenarios and requirements.

5.2 Problem solver

(Question 12:) The tool is aimed at anyone who needs knowledge on software architecture design. *(Question 13:)* There is no possibility to filter knowledge based on the abstraction level. However, patterns can be searched by the type field which expresses the level of granularity (e.g. design pattern, architecture pattern etc). *(Question 14:)* The elicitation and codification of knowledge could take place either simultaneously with knowledge creation or as a separate task. Since the tool constraints the modeling of knowledge, some changes into the architecting process are required. *(Question 15:)* The

[5] http://www.fsf.org/licensing/licenses/gpl.html

[6] http://www.hipergate.org

Hipergate platform supports easy customization of templates, which brings flexibility.

(Question 16:) One can learn to browse the knowledge base in a half an hour, but advanced usage requires proper training. Learning is naturally easier if one is already familiar with the Hipergate platform. *(Question 17:)* As stated previously, PAKME is based on the concept that a significant amount of knowledge is modelled in the tool itself. This naturally takes time despite of easy-to-use data manipulation features. Textual descriptions of architectural models need to be very detailed, because graphical presentation is not supported *(Question 18:)* When deploying the tool, one needs to install Hipergate, a relational database solution and a Servlet container on a server.

5.3 Technology

(Question 19:) The tool is based on Hipergate groupware, which in turn relies on relational database technology and a Tomcat Servlet container. The Internet browser is used as a client software *(Question 20:)* The knowledge is stored in the text fields of a relational database. *(Question 21:)* It is not possible to import or export knowledge in any other format. *(Question 22:)* The Hipergate platform provides good support for a collaborative activity and knowledge is easily accessible with a web browser. The database technology enables quick and powerful searches. Unfortunately, the selected technology fails to support graphical diagrams, which is seen as a very effective means of presenting architectural information [16]. Storing knowledge in a single format facilitates advanced features, but obviously harms integrability and reusability.

(Questions 23:) Other tools can query or update the database through Hipergate API[7]. However, converting the data into another format seems difficult. *(Questions 24:)* Hipergate is open source and published online under the Hipergate Public License[8]. The trial version of PAKME[9] is available on the Internet, but not distributed online. *(Questions 25:)* PAKME relies on the security features of the Hipergate platform. Username and password are entered upon starting a session.

5.4 Maturity

(Question 26:) PAKME was first introduced at the beginning of 2007. *(Question 27:)* We have no knowledge on who is using PAKME, but academic papers [2,3] suggest that the approach has been validated in an industrial setting. (*Question 28:)* There are many academic publications on PAKME, but little other documentation. A user guide is reportedly available by request. (Q*uestion 29:)* Commercial support and consultation services are advertised by the Australian research centre NICTA[10].

6. Discussions

6.1. Analysis of Evaluated Tools

The evaluation relieved the advantages and disadvantages of the three tools as follows.

The Web of Patterns (WOP) tool facilitates sharing concrete, design-level solutions. Because models are mapped directly into source code, knowledge maintenance requires little extra resources. Deployment is easy for loosely-organized, distributed development teams since there is no need to install server side software. Thanks to the code-centric and lightweight approach, the tool should be well-suited for most open source developers. The obvious limitation is that the tool cannot store architectural patterns or other high-level design solutions.

Stylebase for Eclipse can store design knowledge at any level of abstraction. Design knowledge is presented in a clear and illustrative way by using both text and graphics. One of its main advantages is integrability: Eclipse users can continue working with their existing modelling and documentation tools and only exploit output in Stylebase for Eclipse. However, the evaluation made it evident that the tool lacks essential functionality. There is no mechanism for navigating through complex architectural structures. With the exception of a shared repository, there is no support for collaborative interactions. Browsing models is easy, but knowledge maintenance functions lack usability. For these reasons, the current version of the tool seems best fitted for storing reference models, e.g. patterns, which do not change frequently.

PAKME was clearly the most comprehensive of the three architecting tools that we evaluated. Unlike the two others, PAKME seems suitable for managing all the software architectural knowledge of an organization. The tool has an impressive set of advanced functions from complex searches to detecting relationships between architectural models. Thanks to the integration with Hipergate groupware platform, PAKME also boasts very good collaboration features. However, the comprehensive approach comes at a price of lengthy adoption. In practice, users must abandon their existing tooling and start modelling the knowledge with PAMKE instead. Further, PAKME only supports textual

[7] http://www.hipergate.org/docs/api/4.0.0/

[8] http://www.hipergate.org/license/

[9] http://193.1.97.13:8080/

[10]http://nicta.com.au

presentation of architectural models. This can be a threshold for developers who have grown used to graphical diagrams.

6.2 Suitability of the framework

The framework reflects the coverage of the tool features on the SHARK domain, basic tool user viewpoints, selected technologies and the maturity of the tool. These criteria encourages using the framework in selecting a SHARK tool for use in an organization. However, the suitability of a SHARK tool depends on an architecting environment. Before selecting a tool, there must be understanding e.g. on who are the stakeholders, what kind of knowledge is relevant for them and what they expect from the tool. Once the architecting environment has been analyzed, the evaluation framework can be used to get an overview of SHARK tools and short out potentially suitable ones. The case study demonstrates (see 6.1) that the evaluation framework helps in identifying the advantages and disadvantages of each tool. Developing a two-step framework, which would help to analyze both the target environment and the tool, could be an interesting research topic for the future.

The framework could also be used as guideline for tool vendors and open source communities to present an overview of the features of their particular SHARK tool. However, the purpose of the framework is not to rate the SHARK tools based on how many times the questions have the “yes” answer but, moreover, to compare “how” the particular features are supported in the tool and how the tool approaches the problem domain. The difficulty in defining this kind of framework is to make it short and simple enough to be used in a reasonably short time and represent the results succinctly and, on the other hand, to present enough features and viewpoints to reflect the different aspects of the tool. It was also difficult to categorize the questions explicitly into only one criterion group at the time. For example, a question “Is the tool integrable with other tools?” could be as well categorized under “target environment” than under "openness” criteria.

The evaluation questions seemed well-suited for assessing all three tools – despite of the fact that each tool takes a very different approach to AK management. This study focussed on open source tools, but the framework is meant for evaluating all SHARK tools, not depending on their licensing terms. Some evaluation questions are even more applicable to the closed source environment. For example, issues related to openness's and integrability are more relevant when there is a real thread of "vendor-lock-in" [14]. It should therefore be interesting to apply the framework into the evaluation of proprietary SHARK tools. The framework is at the beginning of its lifecycle and we hope it will be developed and improved while being used in the future.

Conclusions

The contribution of this paper was to define an evaluation framework for SHARK tools and make a comparative study of three open source SHARK tools. The framework helps to assess the capabilities of a SHARK tool from four view points: problem domain, target user group, technologies and a tool’s maturity. The evaluation questions concentrate on not only *whether* the tool supports certain capabilities, but also *how* it supports them. The main challenges in the development of the evaluation framework were related to categorizing questions and steering them towards answers that are reasonably short and still sufficiently informative.

Three open source SHARK tools, Web of Patterns, Stylebase for Eclipse and PAKME, were evaluated with the framework as examples. The evaluation revealed the “pros and cons” of each tool, however, target environment must be understood before selecting a tool to be deployed in an organization. WOP is a lightweight tool which enables the sharing of concrete design solutions in an agile manner. Stylebase for Eclipse has poor collaboration features, but is suitable for maintaining a centralized pattern repository. PAKME is a comprehensive tool for managing all software architectural knowledge, but its deployment requires substantial effort.

The evaluation framework is meant to be used in evaluating SHARK tools, comparing them to each other and presenting a summary of their features. We hope that the framework will be actively used and thereby improved and enhanced in the future.

Acknowledgements

The publication of this paper has been supported by the following research projects: EU-SMEPP, ITEA-COSI and ITEA-CAM4HOME. The first mentioned project is funded mostly by the European Commission while the two others are co-funded by Tekes and VTT. The authors have been involved in the development of the Stylebase for Eclipse tool. The evaluation of Stylebase for Eclipse was therefore performed by Mr. Pekka Aho, who we thank for his contribution. We also thank Dr. Jan Dietrich and Dr. Muhammad AliBabar for their help in the testing of WOP and PAKME respectively.

References

[1] Avgeriou, P., Kruchten, P., Lago, P., Grisham, P., and Perry, D 2007. "Architectural knowledge and rationale: issues, trends, challenges". *SIGSOFT Softw. Eng. Notes* 32, 4 (Jul. 2007), 41-46.

[2] Babar, M. A., de Boer, R. C., Dingsoyr, T., and Farenhorst, R. "Architectural Knowledge Management Strategies: Approaches in Research and Industry" In *Proceedings of the Second Workshop on Sharing and Reusing Architectural Knowledge*, IEEE Computer Society. 2007.

[3] Babar, M. A., and Gorton, I. "A Tool for Managing Software Architecture Knowledge." In *Proceedings of the Second Workshop on Sharing and Reusing Architectural Knowledge* (SHARK 2007), IEEE Computer Society. 2007

[4] Capilla, R., Nava, F., and Duenas, J., . "Modeling and documenting the evolution of architectural design decisions" in *Proceedings of the Second Workshop on Sharing and Reusing Architectural Knowledge* (SHARK2007), IEEE Computer Society, 2007

[5] Capilla, R., Nava, F., Pérez, S., and Dueñas, J. "A web-based tool for managing architectural design decisions." *SIGSOFT Softw. Eng. Notes* 31, 5 (Sep. 2006), 2006.

[6] Clements, P., Kazman, R., Klein, M., Devesh D, Reddy and Verma P., "The duties, skills, and knowledge of software architects" in *Proceedings of the WICSA 2007 Conference*, 2007.

[7] Clerk, V. "Towards architectural knowledge management practices for global software development" in *Proceedings of the Third Workshop on Sharing and Reusing Architectural Knowledge* (SHARK2008), IEEE Computer Society, 2008.

[8] Dietrich J. and Elgar C, "A formal description of design patterns using OWL" in *Proceedings of The Australian Software Engineering Conference* (ASWEC) 2005, IEEE Computer Society, 2005.

[9] Dietrich J. and Elgar C. "Towards a web of patterns" in *Proceedings of Workshop on Semantic Web Enabled Software Engineering* (SWESE), 2005.

[10] Dietrich J. and Jones N. "Using Social Networking and Semantic Web Technology in Software Engineering--Use Cases, Patterns, and a Case Study" in *Proceedings of the 2007 Australian Software Engineering Conference* (ASWEC'07), 2007

[11] Farenhorst, R. "Tailoring knowledge sharing to the architecting process". *SIGSOFT Softw. Eng. Notes* 31, 5 (Sep. 2006), 3, 2006.

[12] Farenhost R., Lago P., Vlient H. "Effective tool support for architectural knowledge sharing", *Lecture Notes in Computer Science* 4758/2007. Springer-Verlag, Berlin Heidelberg, 2007.

[13] Farenhorst, R., Lago, P., van Vliet, H. "Prerequisites for Successful architectural knowledge sharing" in *Proceedings of The18th Australian Software Engineering Conference* (ASWEC 2007), 2007.

[14] Goldman R., Gapriel R. *Innovation Happens Elsewhere. Open Source as Business Strategy*. Elsivier, Boston, 2005

[15] Gram, S., "From research software to open source" in Wilhelm R. (Ed.) *Informatics. 10 Years Back. 10 Years Ahead*. Springer-Verlag Berlin, 2001. pp. 195-2008.

[16] Grisham, P., Hawthorne M., Perry D., "Architecture and design intent: experience report" in the *Proceedings of the Second Workshop on Sharing and Reusing Architectural Knowledge* (SHARK2007), IEEE Computer Society, 2007.

[17] Henttonen. K. *Stylebase for Eclipse. An Open Source Tool to Support the Modelling of Quality Driven Software Architecture*. VTT Research Note 2387. VTT Technical Research Centre of Finland, Espoo, 2007.

[18] Henttonen K, and Matinlassi M. "Contributing to Eclipse: A Case Study*", Proceedings of 2007 Conference on Software Engineering* (SE2007), 2007.

[19] Henttonen, K., Matinlassi M., Niemelä E., and Kanstén T. "Integrability and Extensibility Evaluation From Software Architectural models - A Case Study". *Open Software Engineering Journal* 1(1), 1-20, 2007.

[20] Ihme, T., Kumara, P., Suihkonen, K., Nolsti, N., and Paakko, M. 1998. "Developing application frameworks for mission-critical software" *VTT Research Notes 1933.* VTT Technical Research Centre of Finland, 1998.

[21] Jayaratna N. *Understanding and Evaluating Methodologies: NIMSAD: A Systematic Framework*. London: McGraw-Hill, 1994.

[22] Crichton P., Lago P, and van Vliet H. "Building up and reasoning about architectural knowledge." *Lecture Notes on Computer Science* 4214, 43-58. Springer-Verlag, Berlin, 2006.

[23] Kruchten P., Lago P., Van Vliet, H., and Wolf, T. "Building up and exploiting architectural knowledge", *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2005.

[24] Matinlassi M., Niemelä E., and Dobrica L. *Quality-Driven Architecture Design and Quality Analysis Method. A Revolutionary Initiation Approach to a Product Line Architecture*. VTT Publications, Espoo, 2002.

[25] Matthes F., Buckl S., Leitel J., Schweda C. *Enterprise Architecture Management Tool Survey 2008*. Sebis, München, 2008.

[26] Merilinna J. and Matinlassi M. "State of the art and practice of open source component integration", proceedings of the *32nd Euromicro Conference on Software Engineering and Advanced Applications* (SEAA), 2006.

[27] Mäki-Asiala, P. and Matinlassi. M. "Quality assurance of open source components: Integrator point of view. "*Proceedings of the Second International Workshop on Testing and Quality Assurance for Component-Based Systems* (TQACBS 2006), IEEE Computer Society, 189 - 192, 2006.

[28] Shekkerman J. Enterprice Architecture Tool Selection Guide v.4.2. Institute for Enterprise Architecture Developments, 2007. Available at http://www.cioindex.com/nm/articlefiles/51796-EAToolSelectionGuide.pdf

[29] van der Ven, J., Jansen, A. NijHuis J., Bosch, J. "Design decisions: the Bridge between rational and architecture" in Dutoit, McCall, Mistrík and Paech (Eds.) *Rationale Management in Software Engineering.* Springer-Verlag, Berlin, 2006.

[30] van Vliet, H.,"Software architecture knowledge management" in Proceedings of *The 19th Australian Conference of Software Engineering (ASWEC 2008),* 2008.

[31] Walli, S., Gynn D. and von Rotz B. Growth *of Open Source in Organizations*, Boston, 2005.

III

LIBRE SOFTWARE AS AN INNOVATION ENABLER IN INDIA: EXPERIENCES OF A BANGALORIAN SOFTWARE SME

by

Katja Henttonen, 2011

Open-Source Systems: Grounding Research. OSS 2011. IFIP Advances in Information and Communication Technology, vol 365, pp. 220–232, Springer.

https://doi.org/10.1007/978-3-642-24418-6_15

Libre Software as an Innovation Enabler in India Experiences of a Bangalorian Software SME

Katja Henttonen

VTT Technical Research Centre of Finland
Oulu, Finland
katja.henttonen@vtt.fi

Abstract. Free/Libre and open source software (FLOSS) has been advocated for its presumed capacity to support native software industries in developing countries. It is said to create new spaces for exploration and to lower entry barriers to mature software markets, for example. However, little empirical research has been conducted concerning FLOSS business in a developing country setting and, thus, there is not much evidence to support or refute these claims. This paper presents a business case study conducted in India, a country branded as a 'software powerhouse' of the developing world. The findings show how FLOSS has opened up significant opportunities for the case company, especially in terms of improving its innovative capability and upgrading in the software value chain. On the other hand, they also highlight some challenges to FLOSS involvement that rise specifically from the Indian context.

Keywords: Open source, innovation, India, free software, software business.

1 Introduction

Free/Libre and open source software (FLOSS) has been widely advocated [e.g. 1-4] as a way to promote endogenous software innovation in developing countries. The developmental opportunities created by the FLOSS phenomenon have been noticed both by international development institutions (e.g. World Bank and UNDP) and many of the developing countries themselves [1,3,4]. However, despite the enthusiasm, there remains very little empirical research on how developing country companies could successfully integrate FLOSS efforts into their internal innovative activities. Studies on commercially-motivated FLOSS in the US and Europe abound, but the results may not be directly applicable to the diverse innovation environments in the global South. This paper presents some key results of a qualitative case study [5] conducted in India, the country with the most well-known software industry in the developing world. The aim is to understand FLOSS-created opportunities and challenges from the viewpoint of an indigenous software SME.

The focus of the study is on the impacts of FLOSS on the innovativeness and profitability of the case company. Herein, innovativeness means the ability to create and implement new ideas which generate commercial value [cf. 6]. This can entail improvements to products, internal operations or a mix of markets. The study concerns modest incremental innovations, which an SME can generate on a regular basis. The rest of the paper is structured as follows. The second chapter is divided into two sections: the first summarizes theoretical concepts underlying the study and the second one briefly introduces the current debate on whether and how Indian primary software sector could benefit from FLOSS.[1] The third chapter describes the research approach and methods employed in this study, and also very briefly introduces the case company. The fourth chapter presents the actual case study results; it is organized in three sections reflecting three different approaches to open innovation (more on these below). The fifth chapter discusses the meaning of some findings for further research. Conclusions close the paper.

2 Background

2.1 FLOSS as Open innovation: Three Archetypes

This study builds on the Chesbrough's [7] Open innovation theory, which describes the recent tendency of companies to 'open up' their innovation processes. In open innovation, not all good ideas need to be developed internally, and not all ideas should necessarily be further developed within firm's boundaries [8]. Chesbrough and Crowther [9, cf. 10] distinguish two archetypes of open innovation: inbound and outbound. In the case of inbound open innovation, companies monitor the surrounding environment of the firm to find technology and knowledge to complement in-house R&D. In the case of outbound open innovation, companies are looking for external organizations to take internally developed technology into new markets. An additional approach to openness is an interactive value co-creation in strategic partnerships [11, cf. 10] Here, the focus is on innovating together rather than on bringing resources over company borders (inside or outside) [8].

From a perspective of a private company, FLOSS involvement becomes open innovation when it is combined with a sustainable business model [12]. The aforementioned 'subtypes' of open innovation can be used to categorize how primary software companies engage with FLOSS [5, cf. 12,13]. In inbound open innovation, a company sources free-of-charge intellectual property (IP) from FLOSS communities and uses it to produce commercial software products or services. Typically, the main goal is to save own R&D expenses and/or achieve faster time to market[2]. The outbound open innovation entails what West and Gallagher [12] call “open source spin-out”: a company brings internally developed IP into FLOSS domain. It may aim to to create demand for associated commercial offerings or advance strategic goals such as standards creation, for example. OSS communities can also be platforms for open value co-creation where diverse stakeholders join forces to achieve a common R&D goal and pooled contributions are made available to all [cf. 12].

[1] The focus is on introducing the points put forwards in the development literature; the discourse is somewhat different in the FLOSS business literature. For the comparison of discussions in the two disciplines, please see [5].

[2] This does not necessarily equal to a 'parasite approach': a company may motivate external innovation, e.g. by financially sponsoring FLOSS development [5, cf. 13,14].

2.2 FLOSS-Based Innovation in the Indian Context

While Indian software exports have grown exponentially over the past two decades [15,16], many observers have pointed out that the industry's innovative capability has remained relatively low [15,17,18]. The vast majority of Indian software exports consists of low-value-adding off-shoring services such as maintenance of legacy systems [15,17,18]. Due to barriers such as heavy financial constraints, 'late-comer disadvantage' and geographical distance from key customers, many Indian software entrepreneurs struggle to upgrade in the software value chain [15,17]. Meanwhile, 'FLOSS debate' is getting heated: academics and policy makers are arguing [e.g.4,19-22] on whether FLOSS could help some Indian software companies, especially SMEs, to increase innovativeness, add more value and capture more returns.

The proponents point out that sourcing technology from FLOSS communities (i.e. inbound open innovation) saves R&D time and costs and can thereby help Indian companies to overcome financial constraints and 'catch-up' to older players on the global software markets [3,4,23]. Another key argument relates to inter-organizational learning through gradually deepening FLOSS participation (in open co-creation). Unlike off-shoring parent companies, who often have a strong incentive to prevent knowledge spill-overs, FLOSS communities are very motivated to share knowledge across organizational and geographical boundaries [24,25]. This is said to offer valuable learning opportunities to Indian and other Southern companies [2,3,22]. Interestingly, the possible benefits of outbound open innovation has not been discussed much in the development literature, perhaps reflecting a tacit assumption that relevant IP and technical knowledge flows 'from the West to the Rest' rather than vice versa.

Some critics have argued that any competitive advantage derived from FLOSS-enabled cost and time savings is mitigated by GPL-like licensing terms [19,26]. As these licenses make it difficult to sell mass-distributed packaged software, they are said to deprive Southern software companies from the opportunity to benefit from the 'economies of repetition' [19]. Others have pointed out that 'price parity' with pirated software is shirking the markets for FLOSS in the South [21,27]. It also widely acknowledged that the cultural and linguistic barriers may hinder learning trough participative process in FLOSS communities [20,28].There are also significant differences between FLOSS communities on how they draw the boundaries of peripheral participation: some are highly inclusive, while others welcome only very advanced programmers [28,29]. Further, open co-creation and outbound open innovation both require significant investments in non-(directly) revenue generating activities [13] and because Indian companies typically face heavier financial constraints than their Western counterparts, affordability can become a major problem [20]. Launching an own FLOSS project is considered particularly costly and challenging human resource wise [30-32].

Somewhat surprisingly, despite the lively debate, empirical studies on FLOSS activities of primary software companies in India are almost non-existent. Some authors [e.g. 19] even dismiss the subject by saying that FLOSS plays no role in the Indian software industry. However, an international survey [33-35] indicates that, while commercially-motivated FLOSS involvement seems to be a relatively weak phenomenon in India (e.g. in comparison to Europe or Brazil), many small FLOSS companies are still 'out there' and FLOSS experience is also highly appreciated by recruiters in more 'mainstream' software companies. The survey [34,35] also suggests that most Indian companies limit themselves to inbound open innovation as far as FLOSS is concerned. Outbound open innovation seems particularly rare, only three Indian companies were found to author their own FLOSS projects [33]. Mahammodan and De [36] also analysed FLOSS reuse by six proprietary software producers in India. While these organizations reportedly attained significant cost savings by using FLOSS components as 'black box', their developers often lacked sufficient time or skills to even read the source code, leave alone contribute back.

3 Research Approach and Methodology

The paper presents results from a single case study conducted in a company called Mahiti Infotech Private Limited[3] (in short 'Mahiti') which is headquartered in Bangalore and employs 70-90 people. The company employs a customized product development model [37]*:* it develops 'semi-finished' products, often co-creatively with FLOSS communities, and later adds value by customizing them to the needs of individual end-clients. The tailored products go to market either as bespoke software or through the application service provision (ASP) model. Technical consulting provides additional revenue streams.

While planning to conduct more case studies in the future, the author believes that findings from this one case study alone may be valuable for the research community. Especially so, because, to the knowledge of the author, no previous academic study has aimed to 'give a say' to FLOSS entrepreneurs in India. Further, even though the case cannot be argued to be perfectly 'revelatory' nor 'exemplifying' in a strict sense [cf. 38], there are certainly many interesting characteristics to it. For example, despite its relatively small size, Mahiti has an extremely visible role in the Indian FLOSS scene. It can also be regarded as a notable example of an SME which has successfully used FLOSS strategically in order to upgrade in the software value chain. The case company also integrates elements from all the three archetypes of open innovation, thereby allowing to analyse outbound/inbound open innovation and open co-creation within the same organization.

The primary method of data collection was semi-structured interviews of the case company personnel. Three directors, the company's chief executive officer (CEO), chief technical officer (CTO) and marketing director were interviewed along with few senior developers. Two other sources of evidence, documentation (e.g websites and mailing list archives) and unobtrusive observation (mostly of employee interaction on FLOSS related IRC channels) were used to collaborate and augment evidence collected in the interviews. In order to cross-check data further [cf.39,40], some questions were also made to representatives of partner organizations. Most interviews were recorded and transcribed; in few cases, it was necessary to rely on note taking instead. A qualitative method called Template Analysis [41] was employed to thematically analyse the interview transcripts and, to a much smaller extent, some documentary evidence. In short, this means that a coding template was developed

[3] Researchers have argued both for and against disclosing the organization's name in case studies, see [47] for an overview. In this study, the company directors were given a choice and they selected recognition over anonymity.

iteratively whilst the analytical process moved forwards. The final template served as a basis for interpreting the data and writing up the findings. In addition to the thematic coding, some aspects of the Value Network Analysis [42] approach were used. The role of this method was complementary and it is not elaborated herein.

This study aims to confirm to the criteria that Guba [43,44] suggests for qualitative research: credibility (a parallel of internal validity), dependability (a parallel of reliability) and transferability (a parallel of external validity). To improve *credibility*, the study relies on several data sources and two different analysis methods as explained earlier [cf. 40]. The results report has also been shown to key informants for confirmation [cf.39,45]. To ensure *dependability*, complete records have been kept of the collected raw data (a case study database) so that other researchers can check them per request [cf, 22,50]. As for *transferability,* the results from a single case study are not generalisable to other situations, but they can still contribute to the understanding of the target phenomena and thereby provide valuable leads for future research [40,46]. Further, a longer research report available online [5] provides additional contextual information which can help others to make judgements on the transferability of the findings to other settings [cf. 38].

4 Case Study Results

4.1 Experiences in Inbound Open Innovation

In order to save costs, Mahiti intensively encourages the use of FLOSS code and components in all of their software projects. One of the founders gauged that an average Indian software company pays approximately 15% of their profits back in licensing fees, an expense they avoided. However, the cost savings and their profitability implications varied a great deal in practice as illustrated by two recent customer projects (see Fig.1). In the first case, the company only needed to make minor modifications to an existing FLOSS product, but could still charge a 'premium price', higher than that of all proprietary software vendors participating in the bid. This is because

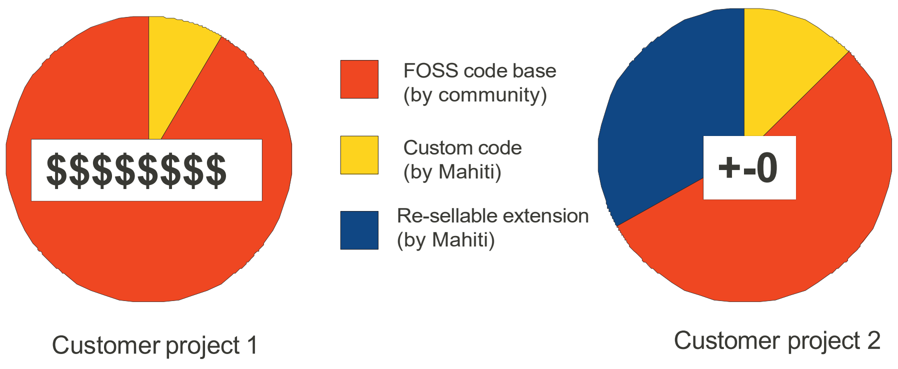


Fig. 1. Proportions of FLOSS and 'own' code in two projects [5]

the FLOSS product in question met well with the needs of the customer as such and Mahiti could offer the fastest lead-time. The profit margin was very high. In the second case, the company had to build almost half of the source code by itself before customer requirements were met, but could still charge a much lower price. The project was not profitable alone, but was still worth doing because the extension developed in this project was expected to be resold to several other customers over time.

FLOSS also brings cost savings to customers and, according to Mahiti's experience, this is helping to expand bespoke software markets in India and other developing countries. Interestingly, unlike most Indian software SMEs [15,18], Mahiti has highly diversified export markets with customers in countries such as Mongolia, the Bahamas, Brazil and Tajikistan. They believed this is partially because FLOSS based solutions are more affordable to Southern customizers, though it is obvious that many other factors are also at play. Nevertheless, it is noteworthy that, while the ease of piracy diminishes the cost advantage of FLOSS on the realm of mass software, the impact is not the same on the bespoke software markets. Pirated mass products can be customized to a certain extent (e.g. Microsoft Excel with macros), but such possibilities are very limited.

FLOSS licensing did not cause any fundamental changes to the company's revenue models[4]: instead of tailoring proprietary software packages, they were customizing FLOSS solutions. The latter allowed them to add-more value in-house, thanks to the low 'purchase price' and unlimited customization options. Profiting from the 'economies of repetition' through closed-source product development was seen infeasible due to financial constraints and highly mature markets: “basically, the curve to recover the funds is very high and this kind of [business] model is not viable for a company like ours”. To the question of whether FLOSS licensing terms limit profit-making, the CEO replied:

– “Yes, if you choose to build your product with open source, you will most probably not become Bill Gates or Steve Jobs. But this is like any career choice, well, you can become a mortgage banker or a broker. [...] Microsoft is what it is today because they have spent money on every single line of code that they wrote. But I cannot start from scratch and build an operating system, I cannot achieve anything like that unless I do it with open source. And when I benefit from the efforts of others, I cannot expect to keep all of the profits alone.”

When asked about the main difficulties in FLOSS reuse, directors pointed to difficulties in finding recruits with any previous experience on FLOSS technologies and development practices. This was seen as stemming from the tendency of local engineering education to ignore the skills needs of FLOSS companies and from the relatively small number of volunteer FLOSS developers [cf. 33] in the country. There was a recognition of some recent positive developments on the field of education. However, while some FLOSS technologies were slowly making their way to the engineering curricula, general code reuse skills were reportedly not. Consequently, the vast majority of new recruits were totally unfamiliar with the whole concept code reuse, only vaguely associating it with 'cut and paste'. They had to be taught 'by hand' which tended to bring up training expenses. As a strategy to address the skills gap, the company has started to offer free-of-charge lectures on FLOSS skills to local engineering colleges.

[4] For more information on FLOSS licensing issues in the case context, please see [5].

For the case company, another concern is that, as the vast majority of FLOSS projects originate from the global North [cf. 48], they do not always address regional needs as well as locally created software could. For example, the directors pointed out that there are practically no FLOSS applications addressing non-urban development needs in India, such as monitoring the quality of water or coordinating rural health-care. “All of these are possible with FLOSS, but there are very few projects moving despite a huge demand”, the CTO said. He added that, while many FLOSS applications are relatively easy to localize in terms of language or metric systems, there are also more fundamental differences in software requirements between countries and regions. Referring to the cultural diversity within India, he continued: “this is a vast country and on the way from Bombay to Delhi the requirements change also... so no matter how much FLOSS there is in a market, it is not enough.”.

4.2 Experiences in Outbound Open Innovation

Mahiti is one of the very few Indian software companies [cf. 33] to author its own FLOSS development project. Recently launched OurBank (www.ourbank.in) is a micro-finance software community which has attracted dozens of volunteer developers, mostly Indian engineering students, and NGOs from as far as Brazil have contributed localization effort. Based on their positive experience, Mahiti's directors are convinced that it is feasible for a resource-constraint SME to launch its own FLOSS community. Success on this arena was seen to depend on “energy and passion” as well as certain key capabilities (e.g. social networking skills) rather than large monetary investments. On the other hand, the CEO did admit that capturing returns from FLOSS spin-outs can be difficult: “Creating a product, architecting it, developing it, convincing other people that it is good and building a community - it is a painful thing to do, but it is sustainable in the long run. However, it does not provide you with returns like these [FLOSS customization projects].”

The mentioned profit-making challenges exist despite some institutional donations (e.g. from EuropeAid) towards the development of OurBank. However, most difficulties were believed to relate to the incubation phase. In the long run, Mahiti plans to step aside from leading the community and become just one of the many contributors. Such partial 'hand-off' was seen necessary so that the community can “evolve naturally”. Time will show how the transition will work out in practise.

Apart from the spin-out described above, Mahiti has a longer history in doing releases which could be called ‘spin-offs’. Whenever they have a piece of 'surplus' source code, which has reached the end of it's life cycle, they put it freely available on SourceForge or similar OSS platform. Because nothing is invested in community building or even making a website, the cost of open-sourcing is very low in this case. The company reported concrete and significant benefits once the IP got 'new life' in FLOSS domain. For example, they once open sourced a very small business application, a leave management system, which was only meant to be used in-house. Later on, they were contacted by a large foundation, who had downloaded the software and wanted to have it extended. The company gained a very important customer in this way, but the marketing effort only consisted of a few mouse clicks.

4.3 Experiences in Open Value Co-creation

Mahiti also plays a globally important role in the development of some FLOSS products such as the Plone content management system. When asked about business gains from strategic FLOSS participation, the global marketing benefits were typically mentioned first. The company does not need to engage in conventional marketing, directors said, because “FLOSS gives us complete visibility”. Being listed as a partner on the Plone website was alone considered to be a major advantage. Further visibility resulted from employees' contributions, which they were always advised to do under the name of the company, and from co-organizing Plone conferences. However, it was not only about having one's name visible but, more importantly, about being seen as a 'shaper' of the technology: “They [customers] come to us because they see us as people who vision the [Plone] product and not only as people having [third-party] expertise on it”, explains the marketing director. In addition, FLOSS communities are specialized social networks were 'word-of-mouth' travels quickly. Recommendations from other FLOSS community members brought in many customers. To exemplify such discussions, a UK-based member recommends Mahiti to another organization on a Plone mailing list saying “I've been told Mahiti has very good Plone/Zope skills and also knows the server side”.

Somewhat expectedly, another group of reported benefits related to inter-organizational learning. The employee training at Mahiti is closely integrated with FLOSS participation. New employees started by following discussions on FLOSS forums and they were encouraged to gradually deepen their participation, very much in line with the classic 'onion model' [49] frequently stated in FLOSS research. The interviewed employees seemed genuinely enthusiastic about this training method. One said that while engineering education had only taught her to complete specific tasks, FLOSS participation had taught her to find solutions independently. From the management viewpoint, there are cost advantages because new employees are coached free-of-charge[5] by external experts. Some drawbacks were also mentioned, for example, FLOSS project administrators did not always explain why they rejected a contribution, which obviously constrained what an employee could learn from the experience.

As to other forms of inter-organizational learning, the company had benefited from adopting process innovations from FLOSS communities. For example, FLOSS involvement had prompted the company to adopt and improve the practice of end-user co-development. As a result, intense collaboration with domestic end-customers, who paid below-market prices in return of participating in R&D and beta-testing, had become a key part of their innovation process. Further, as a result of their FLOSS activities, the company has become geared towards writing well-commented and highly modular source code which is easy to reuse internally. They have even introduced an 'internal source forge', a repository where developers search for reusable source code developed in previous customer projects. These new development practices have enabled the case company to move away from one-time-project development towards developing 'semi-packages' out of reusable modules. In this way, FLOSS had clearly become a booster rather than a barrier to the 'economies of repetition' discussed earlier.

[5] Alternatively, the coaching can be understood as a social return from investments which the company makes to foster its relationship with FLOSS communities [cf. 13]. One developer said that Mahiti's 'good reputation' in communities helped her to get assistance.

The challenges discussed in the context of inbound open innovation also have an impact on open co-creation. In addition, directors acknowledged there are economic barriers to making FLOSS contributions. However, Mahiti has found several ways to keep expenses reasonable. Making minor contributions like bug fixes is integrated into employee training as explained earlier. The company also intermediates contributions made by others, for example, it facilitated local Myanmarian refugees to translate Plone into Burmese and put their contribution online. Or, as in the Hecker's [50] "sell-it-free-it model", large FLOSS contributions often consist of source code that has already been sold to one or more customers. The later model is not only an issue of affordability though; co-operation with end-customers was also seen crucial for ensuring the quality of the contribution. The CTO explains: "You cannot release something [to a FLOSS community] and expect miracles, unless you have tested it and the only way to test a product is to test with a customer...once it's a stable product only then the masses [of FLOSS users] can use it". Reportedly, most of the company's customers do not mind a contractual clause saying that the source code developed for them might be open source later.

Interestingly, developers said that they had not experienced any language barriers to FLOSS participation. Tertiary education had left them with a decent command of English and, besides, they felt that only technical terms are needed to talk on FLOSS related IRC channels. This is not an ethnographic study and it was not possible to detect how more 'subtle' cultural or linguistic issues may shape their identity building as FLOSS developers and effect their sense of belonging to a FLOSS community. On the surface, however, the employees seemed to feel sincerely proud of being well-recognized and respected members of the FLOSS communities where they contributed. For example, they very positively recalled that Joel Burton from the Plone Foundation had visited Mahiti and socialized with them. This was understood to be evidence that their participation is highly appreciated. "If we worked with Microsoft, Bill Gates would not come to chat with us", compared one.

5 Discussion

This paper does not aim to advocate Mahiti's experiences as a success model, which Indian software SMEs in masses could imitate. Firstly, it is appreciated that the study has succeeded to identify more opportunities than challenges. Despite cross-checking information from different sources (including non-company ones), the study still essentially relies on what the informants decided to share. Most people prefer to talk about their successes rather than their failures and the informants were supposedly not free from this common human tendency. Secondly, the case company seems to possess unique capabilities and also has a different market position than most Indian software SMEs. To exemplify the latter, Mahiti serves direct end-customers, over half of which are non-profit organizations. As such customers often agree to open-sourcing the code, which they have already paid for, the company can benefit from 'the sell-it-free-it' model. The scene is supposedly very different for most Indian SMEs, which do subcontracting work for multinational ICT companies.

The paper has 'scanned' several opportunities and challenges faced by the case company and none of these could be discussed in great depth. However, the author hopes that the paper has helped to highlight the wide range of perspectives, which one should take into consideration when discussing FLOSS business in India, or possibly other Southern contexts. For example, some development writers [e.g. 19] argue that FLOSS business models are 'less profitable' without discussing what are the likely alternatives for software companies in that particular country/region. Or, on the other side of the debate, the 'endogenous' nature of FLOSS is often strongly advocated [e.g. 1,4] without discussing the challenges that Southern organizations face when trying to launch their own FLOSS projects.

Most prior work on FLOSS-enabled learning, especially in the development context, focuses on technological knowledge transfer [e.g.2,25,29]. However, this study points to significant benefits from learning new development practices on customer co-development and code reuse. The study also suggests that FLOSS can have mixed impacts on the costs of employee training. These are both interesting subjects for further research, especially considering that the low level of code reuse (often below 5%) and high training expenses are often mentioned among key factors hindering profitability of Indian software SMEs [15,17,18]. Other topic, which deserves more attention, is the potential ability of FLOSS to expand low-cost markets for bespoke software in the South. The strong emphasis, which the interviewees placed on the global marketing benefits of FLOSS participation, is also noteworthy. Very hypothetically, this could related to the cost of international marketing (e.g. adverts in international magazines) being proportionally higher than the cost of R&D labour (i.e. FLOSS participation) for Indian companies.

From the viewpoint of the Open Innovation theory, Mahiti's experiences in uploading 'surplus' source code to the Internet are particularly interesting. Their habit strongly reflects one of the Chesbrough's [7] main “ethos”: one should never 'sit' on the surplus intellectual property. The case study hints that SourgeForge-like platforms might provide a low-cost route for releasing IP which is no longer creating value inhouse. If the released IP creates value elsewhere, there is a chance to claim a portion of that value. While getting theoretical support from Open innovation researchers [e.g. 51], this idea conflicts with many prior studies [13,30,32], which suggest that any commercially-motivated FLOSS release should be supported by significant investments in marketing and infrastructure building.

6 Conclusion

The study illustrates how FLOSS can blur the boundary between software vendor and third-party service provider, thereby opening up new opportunities for companies who lack resources to develop own products from 'scratch'. FLOSS co-creation has helped the case company to develop 'vendor-like' in-depth expertise and build an image as a co-creator of certain technologies. Due to the availability of source code and the absence of licensing fees, they can also add more value to FLOSS products than a non-vendor can typically add on proprietary products. In some cases, FLOSS releases have even helped to open up routes to new markets. Meanwhile, the case company continues to face many challenges such as the poor availability of new recruits with FLOSS competences in India. More research is needed to understand how the findings may apply beyond the single case setting and whether FLOSS has any potential to transform the Indian software sector at large.

Acknowledgements. I want to thank Mr. Pasi Pussinen, a research scientist at VTT, who gave some excellent feedback on a draft of this paper and all the informants, who spent their valuable time being interviewed. As to financing, this work has been supported by the research project ITEI-VTT, which is co-funded by VTT and Finnish National Technology Agency (Tekes).

References

1. Dravis, P.: Open source software. Perspectives for development. Global information and Communication Technologies Department, the World Bank, Washington (2003)
2. Tapia, A., Maldonado, E.: An ICT Skills Cascade: Government-Mandated Open Source Policy as a Potential Driver for ICT Skills Transfer. Information Technologies and International Development 5(2), 31–51 (2009)
3. Weerawarana, S., Weeratunge, J.: Open Source in Developing Countries. SIDA, Stockholm (2004)
4. Wong, K.: Free/open source software: government policy, UNDP Asia Pacific Development Information Program in cooperation with Elsevier, New Delhi (2004)
5. Henttonen K.: Open source as an innovation enabler: A Case Study of an Indian Software SME. Dissertation, The University of Manchester. Institute for Development Policy and Management (2011), `http://opensource.erve.vtt.fi/publications/henttonendissertation.pdf`
6. Subramaniam, M., Youndt, S.: The influence of intellectual capital on the type of innovative capabilities. Academy of Management Journal 48(3), 450–463 (2005)
7. Chesbrough, H.: Open innovation: the new imperative for creating and profiting from technology. Harvard University Press, Boston (2003)
8. Koskela, K., Koivumäki, T., Näkki, P.: Art of openness. In: Pikkarainen, M., Codenie, W., Boucart, N., Heredia, J. (eds.) The Art of Software Innovation. Eight Practice Areas to Inspire your Business. Springer, Heidelberg (2011) (to be published)
9. Chesbrough, H., Crowther, A.: Beyond high tech: early adopters of open innovation in other industries. R&D Management 36, 229–236 (2006)
10. Gassmann, O., Enkel, E.: Towards a theory of Open innovation: three core process archetypes. In: The Proceedings of the R&D Management Conference, Sesimbra, Portugal (2004)
11. Piller, F., Ihl, C.: Open Innovation with Customers – Foundations, Competences and International Trends. Trend Study within the BMBF Project. International Monitoring', RWTH Aachen University, Aachen (2009)
12. West, J., Gallagher, S.: Patterns of Open innovation in open source software development. In: Chesbrough, H., Vanhaverbeke, W., West, J. (eds.) Open Innovation:Researching a New Paradigm, pp. 82–106. Oxford University Press, Oxford (2006)
13. Goldman, G., Gabriel, R.: Innovation happens elsewhere. Open source as business strategy. Elsivier, San Fransisco (2005)
14. Dahlander, L., Magnusson, M.: Relationships between open source software companies and communities: Observations from Nordic firms. Research Policy 34(4), 481–493 (2005)
15. Arora, A.: The Indian software industry and its prospects. In: Bhagwati, J., Calomiris, C. (eds.) Sustaining India's growth miracle, pp. 166–215. Columbia University Press, New York (2008)

16. Athreye, S.: The Indian software industry. In: Arora, A., Gambardella (eds.) From Underdogs to Tigers: The Rise and Growth of the Software Industry in Brazil, China, India, Ireland, and Israel, pp. 7–14. Oxford University Press, Oxford (2005)
17. D'Costa, A.: Export Growth and Path Dependence: Locking Innovations. Software Industry, Science, Technology and Society 7(1), 51–81 (2002)
18. Nirjar, A., Tylecote, A.: Breaking out of lock-in: Insights from case studies into ways up the value ladder for Indian software SMEs. Information Resources Management Journal 18(4), 40–61 (2005)
19. Debroy, B., Morris, J.: Open to development: Open-Source software and economic development. International Policy Network, London (2004)
20. O'Donnell, C.: A case for Indian outsourcing: open source interests in IT jobs. First Monday 9(11) (2004)
21. Sharma, A., Adkins, R.: OSS in India. In: Dibona, C., Cooper, D., Stone, M. (eds.) Open sources: the Continuing Evolution, O'Reilly, Sebastopol (2005)
22. Suman, A., Bhardwaj, K.: Open Source Software and Growth of Linux: The Indian Perspective. DESIDOC Bulletin of Information Technology 23(6), 9–16 (2003)
23. May, C.: The FLOSS alternative: TRIPs, non-proprietary software and development. Knowledge, Technology, and Policy 18(4), 142–163 (2006)
24. Krogh, G., Spaeth, S., Lakhani, K.: Community, joining, and specialization in open source software innovation: a case study. Research Policy 32(7), 1217–1230 (2003)
25. Staring, K., TitleStad, O.: Development as a Free Software: Extending Commons Based Peer Production to the South. In: The Proceedings of the Twenty Ninth International Conference on Information Systems (ICIS 2008), Paris (2008)
26. Reddy, B., Evans, D.: Government Preferences for Promoting Open-Source Software: A Solution in Search of a Problem. Social Science Research Network (2002)
27. Heeks, R.: Free and Open Source Software: A Blind Alley for Developing Countries? IDPM Development Informatics Briefing Paper, Institute of Development Policy and Management. The University of Manchester (2005)
28. Vaden, T., Vainio, N.: Free and Open Source Software Strategies for Sustainable Information Society. In: O. Hietanen (ed.) University Partnerships for International Development: Finnish Development Knowledge, Finland Futures Research Centre, Turku (2005)
29. Wernberg-Tougaard, C., Schmitz, P., Herning, K., Gøtze, J.: (Evaluating Open Source in Government: Methodological Considerations in strategizing the Use of Open Source in the Public Sector. In: Lytras, M., Naeve, A. (eds.) Open Source for Knowledge and Learning Management: Strategies Beyond Tools, Idea Group Publishing, London (2007)
30. Henttonen, K., Matinlassi, M.: Contributing to Eclipse - a case study. In: Proceedings of the Software Engineering 2007 Conference (SE 2007), Hamburg, Germany (2007)
31. Järvensivu, J., Mikkonen, T.: Forging A Community? Not: Experiences On Establishing An Open Source Project. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) Open Source Development Communities and Quality, IFIP Working Group 2.13 on Open Source Software Systems (OSS 2008), Milano, Italy (2008)
32. West, J., O'Mahony, S.: Contrasting Community Building in Sponsored and Community Founded Open Source Projects. In: The Proceedings of the 38th Annual Hawaii International Conference on System Science. IEEE Computer Society, Los Alamitos (2005)
33. UNU-MERIT Free/Libre and Open Source Software: Worldwide Impact Study. D31: Track 1 International Report. Skills Study. United Nations University, Maastricht (2007)
34. UNU-MERIT Free/Libre and Open Source Software: Worldwide Impact Study. D7: Track 1 Survey Report - India. Skills Study. United Nations University, Maastricht (2007)

35. UNU-MERIT Free/Libre and Open Source Software: Worldwide Impact Study. D7: Track 2 Survey Report - India. Software Study. United Nations University, Maastricht (2007)
36. Madanmohan, T., De, R.: Open source reuse in commercial firms. IEEE Software 21(6), 62–69 (2004)
37. Codenie, W., Pikkarainen, M., Boucart, N., Deleu, J.: Software innovation in different companies. In: Pikkarainen, M., Codenie, W., Boucart, N., Heredia, J. (eds.) The Art of Software Innovation. Springer, Heidelberg (2011) (to be published)
38. Bryman, A.: Social Research Methods, 3rd edn. Oxford University Press, New York (2008)
39. Chetty, S.: The case study method for research in small and medium sized firms. International Small Business Journal 15(1), 73–85 (1996)
40. Yin, R.: Case Study Research: Design and Methods, 4th edn. Sage Publications, California (2009)
41. King, N.: Template analysis. In: Symon, G., Cassell, C. (eds.) Qualitative Methods and Analysis in Organizational Research: A Practical Guide, pp. 118–134. Sage Publications, California (1998)
42. Allee, V.: Value Network Analysis and Value Conversion of Tangible and Intangible Assets. Journal of Intellectual Capital 9(1), 5–24 (2008)
43. Guba, E.: Criteria for assessing the trustworthiness of naturalistic inquiries. Educational Technology Research and Development 29(2), 75–91 (1981)
44. Guba, E.G., Lincoln, Y.S.: Competing paradigms in qualitative research. In: Denzin, N., Lincoln, Y. (eds.) Handbook of Qualitative Research, pp. 163–194. Sage Publications, London (1994)
45. Rowley, J.: Using case studies in research. Management Research News 25(1), 16–27 (2002)
46. Flyvbjerg, B.: Five Misunderstandings About Case-Study Research. Qualitative Inquiry 12(2), 219–245 (2006)
47. Walsham, G.: Doing interpretive research. European Journal of Information Systems 15(3), 320–330 (2006)
48. Robles, G., Gonzalez-Barahona, J.M.: Geographic location of developers at SourceForge. In: The Proceedings of the 2006 International Workshop on Mining Software Repositories, Shanghai, China, pp. 144–150 (2006)
49. Ye, Y., Kishida, K.: Toward an understanding of the motivation of open source software developers. In: The Proceedings of the 25th International Conference on Software Engineering, ICSE 2003 (2003)
50. Hecker: Setting up shop: the business of Open-Source software. IEEE Software 16(1), 45–51 (1999)
51. Henkel, J.: Selective revealing in Open innovation process: The case of embedded Linux. Research Policy 35, 953–969 (2006)

IV

MANAGERIAL PERSPECTIVE OF OPEN COLLABORATION AND NETWORKED INNOVATION

by

Katja Henttonen, Pasi Pussinen, & Timo Koivumäki, 2012

Journal of Technology Management and Innovation, 7(3), 135-147

http://dx.doi.org/10.4067/S0718-27242012000300012

Journal of Technology Management & Innovation

Received June 29, 2012 / Accepted September 21, 2012 J. Technol. Manag. Innov. 2012, Volume 7, Issue 3

Managerial Perspective on Open Source Collaboration and Networked Innovation

Katja Henttonen[1], Pasi Pussinen[2], Timo Koivumäki[3]

Abstract

This study explores the managerial perspectives towards open source software and networked innovation. We analysed six software companies who use open source software as a significant part of their product or service offering. The study found notable differences in managerial attitudes, expected benefits and key challenges related to open source software and its role in innovative activities. While all companies were using same pieces of software with open source communities, there were different levels of engagement in the development of the software and information flows between companies and communities. A deeper level of involvement enables the exchange of more than just the code: like ideas, influences, opinions and even innovations or parts of them. The differences in managerial views on open source and networked innovation may be explained by industry domains, value chain position and leadership style

Keywords: open source; free software; external innovation; open innovation; technology management; digital commons.

[1]Service Development and Management, Digital Service Research. VTT Technical Research Centre of Finland. Kaitoväylä 1, FI-90571, Oulu, Finland. Email: katja.henttonen@vtt.fi. Phone: +358 40 821 7180.
[2]Value-Driven Service Business, Digital Service Research. VTT Technical Research Centre of Finland. Kaitoväylä 1, FI-90571, Oulu, Finland. Email: pasi.pussinen@vtt.fi. Phone: +358 40 351 4858
[3]Digital Service Research. VTT Technical Research Centre of Finland. Kaitoväylä 1, FI-90571, Oulu, Finland. Email: timo.koivumaki@vtt.fi. Phone: +358 40 507 3631

Authors

Katja Henttonen is a Specialist at Digital Services Research within the VTT Technical Research Centre of Finland. Her research interests are related to the delicate art of 'openness' in inter-organisational collaborations.

Pasi Pussinen is a Research Scientist in the Value-Driven Service Business Team at VTT Technical Research Centre of Finland. His research interests include open source software, business models and service dominant logic.

Timo Koivumäki is a Research Professor of Mobile Business Applications at the VTT Technical Research Centre of Finland and at the University of Oulu. His research interests include consumer behaviour in e-commerce, m-commerce and ubi environments, user-driven innovation, mobile marketing, digital economy and information goods.

1. Introduction

Various business models based on free and open source software (FLOSS) have been widely studied in academia (e.g. Bonaccorsi et. al. 2004, Favaro and Pfleeger 2011, Spiller and Wichmann 2002, Lerner and Tirole 2002). However, there seems to be relatively little research into why some open source companies take a very proactive role as FLOSS developers/advocates while others only use publicly available FLOSS resources and minimise any community involvement. This difference is not evident from FLOSS business-model literature because most known business models can be linked with either approach.

This study was born from a desire to understand key factors and determinants that turn companies into 'passive exploiters' or 'active contributors' in FLOSS. The focus is on analysing the difference in managerial perspectives towards open source and networked innovation. The selected research approach is a multiple case study of six software companies which all utilise FLOSS intensively but differ in terms of their engagement with FLOSS communities. Theory-wise, the study benefits from Chesbrough's (2003, 2006) open innovation paradigm and, more specifically, builds on the difference between 'external innovation' and 'open innovation' which was proposed by West and Gallagher (2004, 2006). This paper claims that companies who actively contribute to FLOSS development have adapted the open innovation paradigm, while mere exploiters employ the external innovation model. Following Valkokari et. al. (2009), we use the term 'networked innovation' to refer to all externally-orientated approaches to innovation, including both 'open innovation' and 'external innovation'.
The results suggests that FLOSS companies can indeed be meaningfully categorised into 'external innovators' and 'open innovators' and that there are significant differences between the two. The managers of the two kinds of companies view FLOSS differently, expect different operational benefits from it, face different challenges and, consequently, employ contradictory managerial techniques. For example, external innovators view FLOSS as a 'free lunch' and look solely for cost savings while, open innovations perceive FLOSS as a fundamental element of value creation and seek to become shapers of the technologies in question. Further, external innovators mostly attempt to 'work around' conflicts of interest with FLOSS communities, while open innovators seek to establish maximum synergy with them to reap the benefits of pooled R&D.

The rest of the article is structured as follows. The second chapter summarises theoretical concepts underlying the study and explores how they compare to some models used in prior FLOSS literature. The third chapter describes the research approach and methods employed in this study, and also briefly introduces the case study companies. The fourth chapter presents the actual case study results and presents a brief summary of them. The fifth chapter discusses the limitations of the study and gives suggestions for further research. Conclusions close the paper.

2. Theoretical background

2.1 Three innovation models: closed, external and open

Over the past decades, co-operation and networks have come to the fore in innovation research (see e.g. Tuomi 2002 or Chesbrough 2006 for a historical review). Relatively recent ideas on the collaborative nature of innovation include, for example, the concepts of extended enterprise (Dyear 2000), open innovation (Chesbrough 2003, 2006), user-driven innovation (von Hippel 2005) and creation nets (Hagel and Brown 2011). This article builds mostly on Chesbrough's (2003, 2006) idea's on open innovation. His theory describes the recent tendency of companies to 'open up' their innovation processes. The main claim is that not all good ideas need to be developed internally, and not all ideas should necessarily be further developed within a firm's boundaries (ibid; Koskela et. al. 2011). Two important characteristics of the Open Innovation theory are that it gives considerable attention to the purposive outbound flows of intellectual property (IP) and underlines the need to motivate the creation of relevant knowledge outside the company (ibid).

Based on Chesbrough (2003, 2006), West and Gallagher (2006a, 2006b) acknowledge three innovation models: closed innovation, external innovation and open innovation. In the closed innovation model, internal research and development (R&D) activities feed the company's production pipeline and products are brought to market by the company itself. In the external innovation model, the company seeks to develop what Cohen and Levithal (1989) termed 'absorptive capacity' and utilises external sources of innovation such as universities, customers, supplies and competitors. However, very much like in the closed innovation model, the outbound flows of intellectual property (IP) are viewed as unwitting 'spill-overs'. While sharing some characteristics with external innovation, the open innovation model goes beyond. Instead of merely exploiting what 'happens' to be available, open innovators employ a systematic strategy for motivating the creation of external knowledge. They also use purposive outward IP flows to reach new markets and maximise returns on internal innovation. Table 1 (on the next page) summarises the characteristics of each model, showing the managerial attitudes, key challenges and resulting managerial techniques associated with each. This paper focuses on the difference between external and open innovation. For clarity, the characteristics that distinguish open innovation from external innovation are underlined.

The difference between external and open innovation is in line with the recent study (Paasi et al 2010, Luoma et al 2010) on intellectual property management in inter-organisational networks. Based on their extensive empirical study (ibid) and the knowledge management theory of Grand and Baden-Fuller (2004), the authors recognised two categories of inter-firm relationships: 'knowledge co-creation relationships for knowledge exploration' and 'knowledge transaction relationships for knowledge exploitation'. The former focuses on joint knowledge creation and resembles open innovation; the latter focuses on the efficient utilisation of existing knowledge and can be associated with the external innovation model. The stated difference between external and open innovation also contains clear analogies with other categorisations of innovation practices such as 'inboud open innovation vs. open value co-creation' by Koskela et. al. (2011), 'user cluster vs. open cluster' by Indrissal et. al. (2012) and 'Explorers vs. Professionals' by Kaup and Gassman (2009).

The term 'open innovation' is sometimes used to describe all scenarios where companies create profits from open source software. However, increasingly many software-intensive companies appropriate assets from FLOSS communities and use them to create proprietary products, without making any noticeable contributions back (Dahlander and Magnuson 2005; Stams 2009). Lacking steps to motivate the in-flows of external IP or to benefit from outbound IP flows, such an approach exemplifies the external - rather than open - innovation model (West and Gallagher 2006). In contrast, the open innovation model entails some reciprocal interaction with FLOSS communities (ibid). Such reciprocity enables learning through co-creation (cf. Krogh et. al. 2003)

Innovation model	Managerial attitudes	Key managerial challenges	Related managerial techniques
Closed innovation	Only internal R&D matters, 'not invented here' syndrome Fierce protection against spill-overs	1. Attract the best talent into the company 2. Exploit own research commercially	1. Provide excellent compensation, resources and freedom to internal inventors 2. Provide a dedicated development function to link research with market knowledge
External innovation	Harvesting external ideas, 'innovation happens elsewhere' Modest protection against spill-overs	1. Explore a wide range of sources for innovation 2. Integrate external knowledge with own innovative activities	1. Scan environment carefully 2. Develop absorptive capacity, utilise networks
Open innovation	Facilitating external innovation, pooled R&D, 'innovation happens together' Willing spill-overs, 'never sit on surplus IP'	1. Motivate the creation and contribution of external knowledge 2. Incorporate external knowledge with own innovative activities 2. Maximise exploitation of diverse IP resources	1. Provide intrinsic rewards for contributions 2. As in external innovation 3. Share or give away IP to maximise returns from entire innovation portfolio

Table 1. Three innovation models summarised, modified from West and Gallagher (2004,2006)

and ensures so that communal resources are continuously replenished (Dahlander and Magnuson 2005). As open innovation companies have internalised the idea that willing spill-overs can be beneficial, they are not 'scared' of releasing their own IP to the FLOSS domain in order to achieve promotional or strategic goals (cf. Henkel 2006).

West and Gallagher (2006a; 2006b) name two main challenges for external innovation and three for open innovation. The first challenges for external innovators is exploring the wide range of knowledge sources, i.e. to perform environmental scanning to find out what happens on the FLOSS scene and what could be exploited from there. Meanwhile, open innovators define their challenge in terms of motivating external innovation i.e. how to keep open source continuously producing inputs that are beneficial for the company. The second challenge of integrating external and internal activities is shared by both external and open innovators and contains a diverse set of diverse legal, technical and business issues. The third challenge is only accepted by open innovators and it relates to the maximisation of returns by giving away 'surplus' intellectual property.

2.3. Other 'categorisations' of commercial FLOSS engagement

There are some prior studies which have aimed to categorise FLOSS companies according to the 'intensity' of their engagement with FLOSS communities. For example, Dahlander and Magnuson (2005) detected that companies adopt either a 'commensalistic', 'symbiotic' or 'parasitic' relationship with FLOSS communities (see also Lundell et. al. 2006). The symbiotic relationship resembles the open innovation model as both carry the idea of reciprocity and mutual benefit. Our study shows that what were herein describe as the 'external innovation model' can become a commensalistic relationship at best (this means benefiting from another entity while leaving it without harm) or turn into parasitic one at its worst.

Grand et. al. (2004) and Dahlander (2007) propose four modes of company involvement in FLOSS. Grand et al. (2004) understands their four levels as 'progressive': each level implies bigger investments and a greater reliance on FLOSS, but also more operational benefits and improved opportunities for knowledge sharing and learning. As presented in Figure 1, the 'lowest' level of involvement could be associated with external innovation and the two 'upper' levels with open innovation as defined herein. In turn, Dahlander (2007) presents commercial FLOSS participation as a 2x2 matrix where the variables are the intensity of FLOSS participation (low/high) and the initiator of the project (the company itself or a wider community). This study is located on the other side of the matrix, focusing mostly on how companies engage in FLOSS projects initiated by others.

Thus, the proposed distinction between 'external innovators' and 'open innovators' is not at odds with classifications

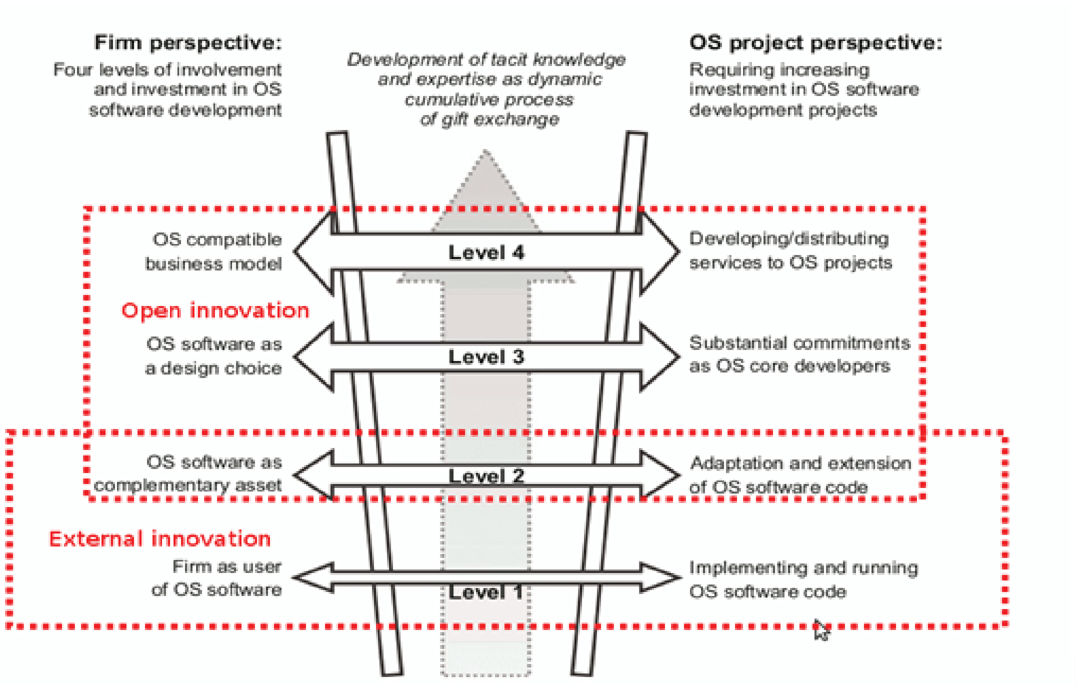

Firm perspective:
Four levels of involvement and investment in OS software development
Development of tacit knowledge and expertise as dynamic cumulative process of gift exchange
OS project perspective:
Requiring increasing investment in OS software development projects
OS compatible business model
Level 4
Developing/distributing services to OS projects
Open innovation
OS software as a design choice
Level 3
Substantial commitments as OS core developers
OS software as complementary asset
Level 2
Adaptation and extension of OS software code
External innovation
Firm as user of OS software
Level 1
Implementing and running OS software code

Figure 1. Levels of FOSS involvement (from Grand et. al. 2004, red text/symbols added)

in prior FLOSS literature. However, to our knowledge, this is the first article which studies the difference from the view point of innovation management.

3 Methodology

3.1 Research approach and case selection

As stated previously, six small-scale case studies of open source companies were performed. Herein, multiple case studies were not used for the purpose of literal or theoretical replication; there was not yet a well-formed theory to 'test'. Instead, considering the preliminary and exploratory state of the research, the goal was to maximise the 'richness' of information for qualitative analysis (cf. Flyvbjerg 2006). Consequently, we selected case study companies which differ from each other in several dimensions, e.g. geographic location, size and software sector. All case study companies were required to comply with the following selection criteria: (a) utilise FLOSS intensively as part of their product or service offerings and b) have different levels of activity within FLOSS communities and thus be placed differently on the 'continuum' from external to open innovation. The case selection was also influenced by the ease of access: in five out of six cases, the interviewing authors and company personnel had already collaborated on other research projects. These prior collaborations gave us in-depth understanding on the managerial philosophy and practices of the case study companies and actually pointed to the research problem at hand (see Henttonen 2011).

3.2 Data collection and analysis methods

The primary method of data collection was semi-structured interviews of company personnel. All interviews were literally transcribed. Other sources of evidence were online documentation and unobtrusive observation of employee interaction on FLOSS forums. These had a secondary role and were mainly used to collaborate and augment evidence collected in the interviews. In some cases, company partners were also contacted to confirm particular details. Qualitative method called Template Analysis was employed to thematically analyse the interview transcripts and, to a much smaller extent, some documentary evidence. In short, this means that a coding template was developed iteratively while the analytical process moved forwards. A short, initial version of the template reflected the pre-assumptions based on the theoretical frame while later versions were updated to reflect themes emerging from the data set. The final template served as a basis for interpreting the data and writing up the findings.

4. Case study results

This chapter presents the case study results. The first chapter analyses the managerial attitudes, challenges and techniques associated with the external innovation model. This model, in our view, is represented by companies F, C and A and, to a lesser extent, E. The second chapter discusses how the same managerial issues are faced by open innovators, i.e. companies D and B. Figure 2 presents how the companies are positioned on the continuum from external to open innovation.

'Alias'	Product offering	Size	Strategically important FLOSS products
Firm A	Embedded systems (hardware and software) for automotive and wireless industries	Personnel 1000-2000 , turnover 100-200 M€	Linux-kernel for most products lines, diverse FLOSS applications (games, multimedia, office etc.) for terminal end-user devices
Firm B	Customised business software solutions and consultation services for end-clients	Personnel 50-100, turnover below 2 M€	Linux desktop solutions (Ubuntu), FLOSS databases (MySQL, PostgreSQL), content management systems (e.g. Plone, Joomla) and development tools (e.g. Zope)
Firm C	Embedded systems (hardware and software) design as a subcontractor	Personnel 100-500, turnover 10-30 M€	Embedded Linux-distributions, development tools (e.g. Subversion, Bugzilla)
Firm D	Advanced web solutions for both IT contractors and direct end-clients	Personnel 20-50, turnover below 2 M€	FLOSS databases (MySQL), content management systems (e.g. Joomla, Drupal), e-commerce solutions (Magento) and development tools (e.g. Zend framework)
Firm E	Mobile software, media portals, web-based enterprise software	Personnel 1000-2000, turnover 50-100 M€	Mobile Linux (e.g. Android, Meego), content management systems (e.g. Alfresco), enterprise software platforms (e.g. Liferay)
Firm F	Embedded software for the mobile/cellular industry	Personnel below 20, turnover below 1 M€	Embedded Linux distributions, user interface development tools (e.g. GTK)

Table 2. Summary of the case study companies

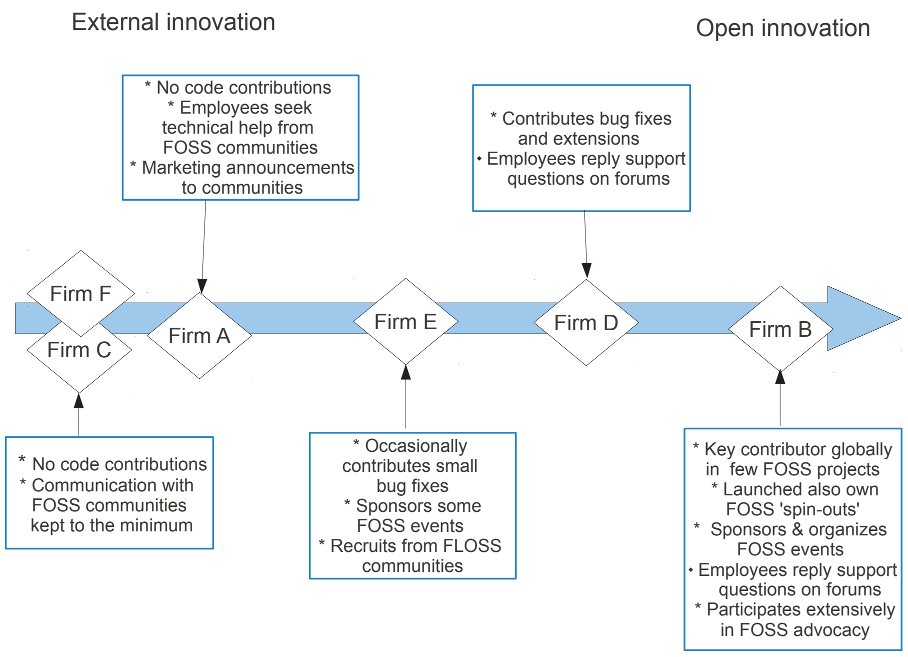

External innovation
Open innovation
* No code contributions * Employees seek technical help from FOSS communities * Marketing announcements to communities
* Contributes bug fixes and extensions • Employees reply support questions on forums
Firm F
Firm C
Firm A
Firm E
Firm D
Firm B
* No code contributions * Communication with FOSS communities kept to the minimum
* Occasionally contributes small bug fixes * Sponsors some FOSS events * Recruits from FLOSS communities
* Key contributor globally in few FOSS projects * Launched also own FOSS 'spin-outs' * Sponsors & organizes FOSS events • Employees reply support questions on forums * Participates extensively in FOSS advocacy

Figure 2. Six case study companies placed on the continuum from external to open innovation

4.1 FLOSS as external innovation

4.1.1 Managerial attitudes and goals

External innovators did not perceive FLOSS communities as part of their value network – instead, publicly available source code was seen more like a 'bulk' resource on which to feed. They all saw free-of-charge software artefacts as the main 'gain' from FLOSS, underlining how cost and time savings had helped them to offer reduced prices, make bigger profit margins and/or achieve shorter lead times to market. They also recognised that the cost advantage was far from being marginal: a couple of companies said that they could never have entered a particular market without FLOSS. 'There were initially very few [mobile] terminal vendors who could afford building platforms based on proprietary systems... but now we have an [open source] software pool that has helped to kick-start many new vendors [like us],' explains a manager from company E.

Considering the heterogeneity of the case study companies, it is hardly surprising that FLOSS has a different place in their innovation processes. For a couple of companies, FLOSS was a free-of-charge 'base brick' on which their own products were built. For example, a manager from company E explains, 'Open source is not value adding: it is simply a matter of getting the base software stack in a very mature state from day one and then you can concentrate on adding your true value.' Others viewed FLOSS more like a decorative chimney stack, bringing something extra on top of their own products: 'We keep receiving free-of-charge updates ... we can just say to the customer that "Hey, our next release contains this fascinating new feature" ...we hardly have to think about new features by ourselves.' However, what was common to all companies classified by us as 'external innovators' was their tendency to view FLOSS as a 'free lunch'. Despite gaining significant business benefits from FLOSS, the companies felt unwilling to contribute anything. Their answers reflected attitudes which are characteristics to closed and external innovation models. Mostly, FLOSS contributions were perceived as giving money to a charity: a benevolent, but eventually wasteful, activity that a company could not afford in the long run (with the possible exception of small PR investments). One interviewee from company E restructured the question in terms of 'Why would anybody contribute?'. He continued: 'You are not giving gold for free to anybody. It is like a joke that, you know, "Oh its open sourced, everybody contributes"... hah, nobody contributes, especially not the big companies focusing on assets protection'. FLOSS participation was also likened to a 'janitor's job' something that is undoubtedly necessary but not attractive technically or monetarily – so let somebody else do it. Further, some said that their customers were quasi-paranoid about openness: 'If we started hanging out on open source forums, they [customers] may think that we will tell their secrets to the world... even if we did not, it would cast a shadow of doubt.'

Despite the underling cost and time savings, two external innovators also tried to exploit other opportunities within FLOSS to a limited extent. For example, company A sends marketing announcements to the mailing lists of FLOSS communities when a new product comes out. Company E has gone a step further and builds its brand by contributing small bug fixes and sometimes even sponsoring FLOSS events. Their principal software architect explains how the company occasionally manages to get 'fifteen minutes of fame' with minimal contributions and concludes: '[FLOSS] communities are important for marketing, and not just marketing towards customers but also marketing towards developers, enabling the company to hire smart brains... no doubt about that.'

4.1.2 Managerial challenges

The first challenge mentioned for external innovators relates to exploration. Interestingly, several interviewees mentioned that it was very difficult to keep up-to-date on what is happening on the FLOSS scene. This was blamed on insufficient human resources internally or the need to keep all resources engaged in customer projects, leaving an impression that environmental scanning was seen secondary, after all. However, a senior specialist from company A expressed that his top-management should definitely pay more attention to the issue:

'We are pretty unorganised on this [scanning for FLOSS-related innovation]... it is useless to go randomly surfing the Internet every Tuesday morning like "la di da, can't find anything here, let's try again next week" – instead, we should really have a carefully managed process for staying up-to-date on the latest FLOSS developments.'

The second challenge is to 'integrate external knowledge with own innovative activities'. On this area, the key issues for the interviewed companies related to quality assurance, legal liabilities and release maintenance. The quality-related challenges were very much in line with what has already been widely reported. It was underlined that FLOSS never provides any guarantees on quality and therefore each component has to go through an internal quality assurance and testing pipeline.

As to legal challenges, it was said to require significant skills to 'make sense' of a multitude of FLOSS licences, e.g. regarding reciprocal compatibility and propagation mechanisms (cf. Dahlander 2005; Henttonen and Matinlassi 2008). Further, many customers were reportedly unwilling to accept the reciprocity demands of FLOSS licences, especially those made by the popular GPL licence . One interviewee (company A) noted that such resistance is often based on a fixed world view rather than careful business analysis: 'Even when it's a totally irrelevant component, no business secrets, no patents, no major expertise, nothing... they oppose [open sourcing it] merely out of principle... they just cannot imagine another way.'

The most prominent challenges related to release maintenance. Since the companies do not contribute their own modifications to FLOSS communities, they have to maintain a separate version of the product, a kind of in-house 'fork' (cf. Nyman and Mikkonen 2011) by themselves. Thus, major effort is required to synchronise their own FLOSS-based product with the community version. If an original FLOSS community splits apart, creating more 'forks', the company has to merge its own product with several versions. This is closely intertwined with the questions of control and power. The company may be very negatively affected by a community's decisions: on technology standards, product architecture and release cycles. But, as the company is 'nobody' inside the FLOSS community, there is little use to raise objections:

'This [influencing a FLOSS community] is tricky diplomacy... you cannot create a forum account on one day and go there on next day to tell people what to do. You are first required to build respect and a [brand] name for yourself by being committed and contributing. This applies to individuals and companies alike... Despite being a big company and enjoying an established position in the industry, we are absolutely nobody in the FLOSS world.'

(Senior specialist, company A)

4.1.3 Managerial techniques

Since the companies devote little or no effort in environmental scanning, how do they find out about new FLOSS projects with business relevance? The companies depend heavily on individual employees who are 'hobbyists' and contributors in FLOSS projects in their free-time. For example, when asked how they kept an eye on new developments with mobile Linux, the representative of company F replied openly: 'We cannot afford to use working time [on this].... but Linux is a hobby for the most [of the employees] so we will stay up-to-date that way.' Sometimes companies can maintain a dialogue with key FLOSS communities through individual employees who reportedly make significant contributions in their free-time and thus have a 'name' in those

GPL i.e. GNU General Public Licence is one of the most well-known and widely used FLOSS licenses. It is based the idea of 'copyleft' and is particularly strict in its requirements for developers to release the source code of derived or joint work (see e.g. McGowan 2005).

Such tools analyze software packages from a legal perspective, searching source code for license declarations and technical interdependences which impact how licensing terms propagate (for more information see e.g. Oksanen 2006) communities. However, despite their reliance on employees' own FLOSS enthusiasm, the companies do not give any specific rewards for this. So, somewhat surprisingly, external innovators seemed to depend, not only on the volunteer contributions from external FLOSS developers, but also on those from their own employees. Because these companies had indicated strong worries about 'unwitting' spill-overs, it was interesting that they preferred their employees to be involved in FLOSS communities as individuals, or 'hobbyists', instead of a more-controlled policy.

There were relatively well-defined managerial techniques in place to address challenges regarding legal liabilities and quality assurance. The companies use a combination of dynamic and static testing, very much like those described by some previous authors (e.g. Maki-Aisala and Matinlassi 2006). As to legal issues, two companies use automatic licensing tools and most add empty 'glue code' in order to isolate FLOSS code from their own or customer's code. The 'glue code' layers are often totally void of functionality and do nothing to aid component interoperability. Instead, their sole purpose was to stop GPL licensing terms from propagating and thereby avoid associated liabilities (e.g. reciprocity demands, patent licensing etc.) The practice was perceived to be a 'rule' rather than an exception in FLOSS business:

'The common line with these [mobile software] products is that what is open sourced are the trivial parts, because the real costly things, that is IPR... is never open sourced... one can do that with the GPL as well, because big companies are just using open source empty glue layers and then, you know, protecting their assets.'

(Principal software architect, company E)

Interestingly, even though interviewees emphasised challenges related to release maintenance, they could not name any concrete steps taken to address them. It seems that most took for granted that FLOSS projects are unpredictable/uncontrollable by 'nature' and this was just 'a risk to live with'. However, as a side note - the companies did share some of the same software development tools with communities, like GIT and Bugzilla.

4.2 FLOSS as open innovation

4.2.1 Managerial attitudes and goals

Open innovators had clearly different rationales for their FLOSS involvement. FLOSS communities were understood as an important part of the external value network and were deemed essential for global marketing, inter-organisational learning and joint development. Both companies B and D found most of their customers either through general visibility on FLOSS forums or through direct references from other community members. 'It is here [in the OSS world] where we get complete visibility,' says the marketing director of company B, explaining that they had practically abandoned conventional marketing in favour of FLOSS networking (cf. Henttonen 2011).

The importance of inter-organisational learning was also underlined; company B had integrated employee training with the participative learning methods of FLOSS communities. In addition to technical learning, the communities were a source of information on what is happening in the market. The CEO of company D explains that the FLOSS world is full of excellent online conferences, blog sites and other resources which help him to stay informed on, 'What is hot and what is up right now on the market,' adding that such market knowledge obtained from FLOSS forums, has greatly assisted him in positioning his company favourably.

The cost and time savings related to FLOSS were equally important to 'open innovators' – however, these were viewed as a successes of joined development or 'pooled R&D' rather than as a 'free lunch'. The importance of reciprocity was underlined in several accounts. For example, the CEO of company B explains that his company could have never built mature software from 'scratch' and continues, 'I could never achieve anything like that without open source and, well, when I benefit from the efforts of others I cannot expect to keep all profits to myself.' (cf. Henttonen 2011). He continues to underline the importance of contributing to the FLOSS projects on which his company depends:

'If you are part of the ecosystem you have do things to sustain that ecosystem. If you are just a consumer, then that ecosystem will sooner or later die… in order to make the open source ecosystem stable, you [a company] have to start looking at other aspects than just being a consumer... to contribute in different ways and make sure that the ecosystem stays alive.' (General Executive Officer, company A)
FLOSS participation was also seen as an opportunity become 'shapers' rather than just 'users' of a particular technology. The difference was also noticed by customers. 'They [customers] come to us because they see us as people who envision the [FLOSS] product and not only as people having [third-party] expertise on it,' says the marketing director of company A.

The interviewees either did not know or did not openly admit it, but the described 'glue code' models are known to be 'grey' or borderline cases legally (e.g. Hopner 2004). They are clearly not 'safe' but reduce risks compared to boldly ignoring the GPL terms.

Further, both companies D and B had internalised the open innovation 'philosophy' that one should never 'sit' on surplus IP. For example, whenever they have a piece of source code, which has reached the end of its life cycle, they put it freely available on SourceForge or another similar FLOSS platform. Sometimes there are surprising benefits when the IP gets 'a new life' in the FLOSS domain. For example, company B open sourced a very small business software, which was only meant to be used in-house (Henttonen 2011). Later, they were contacted by a big foundation, which had found the software from the Internet and wanted to have it extended. Thus, they got a very important customer with minimal 'marketing' effort

4.2.2 Managerial challenges

It seems that while external innovators struggle to stay tuned to developments on the FLOSS scene, open innovators use FLOSS forums to keep up-date-on on what is 'hot and in' , not only on FLOSS, but on the software markets in general. To address the challenge of motivation, open innovators make significant contributions to FLOSS communities, e.g. by committing resources to open software development and by co-organising FLOSS events. This raised an obvious question on how they can afford so many activities which do not generate revenues directly.

From the integration challenges mentioned previously, only quality assurance concerns were mentioned by open innovators. In stark contrast with external innovators, licence compatibility issues and other legal 'risks' were seen as fundamentally non-threatening due to close and friendly ties with legal copyright owners, i.e. the communities. The afore-mentioned problems on release maintenance were also eliminated: since base software was developed together with the community, there was no need to maintain a separate version for 'them' and 'us'. For open innovators, the biggest integration challenges related to developing the technical, social and business skills required by 'fully fledged' FLOSS involvement. Because such skills are not commonly taught in universities, they have to make significant investments in teaching the 'FLOSS ways' to new employees.

As to willing spill-outs and spin-offs, the biggest challenge named by open innovators were so called 'open source piracy'. This means that sometimes competing software companies appropriate the source code but illegally ignore the reciprocity terms of the FLOSS licence. This often means that potential benefits and 'credit' of the released IP goes to a competitor and nothing comes back to the original owner of the IP. This type of piracy was said to be common and it is exactly what some 'external innovators' in this study admitted to.

4.2.3 Managerial techniques

When asked about the affordability of non-direct investments, open innovators replied that whole-hearted FLOSS participation requires 'energy and passion' rather than big monetary investments. To exemplify such an attitude, their marketing director run a city marathon dressed as a blue elephant, a mascot of a well-known open source project, Postgre SQL. This earned the company five minutes of fame on a national TV channel. Once the CEO had contacted a local refugee centre and asked them to translate OpenOffice into an 'exotic' language. So, despite being a medium-sized company, they showed something that could be called 'community spirit'. While external innovators relied on the FLOSS enthusiasm of individual employees, the managers of open innovation companies were clearly the sources of such enthusiasm themselves.

Open innovators shared quality assurance concerns with external innovators, but adopted totally different techniques to address this challenge. First, they picked-up FLOSS software with a 'good reputation', they used their excellent social networks to accumulate knowledge on quality issues and made decisions on that basis. Secondly, they engaged some of their customers in the co-development and co-testing of products. The technical manager of company B says, 'The only way to test a product is to test with a customer and slowly start working to stabilise it... when I give software to them [certain customers] at a low cost, I can do some beta-testing, some R&D on them... this is how we bring in stable code.'

This was seen as an important continuum from FLOSS development practices which has always emphasised end-user involvement in R&D. This is just one example of the 'ways of doing things' adopted from FLOSS communities. Another example comes up when looking at how they respond to the aforementioned challenge of training new employees. In both companies B and D, employee training follows the classic 'onion model' (Ye at Kishida 2003) which is frequently used to describe how participative learning occurs in FLOSS communities. New employees started by following discussions on FLOSS forums and were encouraged to gradually deepen their participation and eventually make contributions of their own. Within FLOSS communities, new employees are 'coached' by external experts free-of-charge which supports in-house training efforts.

To fight against unwanted appropriation by competitors, the companies always used a GPL licence when giving out their own intellectual property. Interestingly, a licence that was mentioned as a management challenge for external innovators, was a protection technique for open innovators. While GPL licensing terms can also be circumvented (as shown in section 4.1.3), the companies saw it as a relatively efficient tool against unwanted appropriation (justifiably, see Hopner 2004).

4.3 Summary of the results

This study has explored managerial views on open source and networked innovation in six case study companies. The results are summarised in Table 3. The study showed that there are fundamental differences in the managerial attitudes: while the management of external innovators clearly present an exploitation attitude and see FLOSS as a 'free ride ' to cost savings, the managers of companies with an open innovation approach see FLOSS as a fundamental element of their value creation process. Furthermore, open innovators clearly see FLOSS as a 'two-way street' of giving and receiving. There are also notable differences in the way the managers see the main challenges and in the way these challenges are tackled. For example, external innovators see the reciprocity demands of FLOSS licences, especially GPL, as a major obstacle and actively seek ways to work around it. Open innovators have a totally opposite view on the issue: they view open source piracy as a major challenge and see strong licensing schemes such as GPL, as a valid protection technique.

5. Discussion

There are problems in the methodological design of the study. With hindsight, the case study companies did not have enough common denominators: while the heterogeneity of companies may indeed have contributed to a diversity of viewpoints, it also diminishes the value of analytical comparisons between the companies. Further, one of the key advantages of the case study approach is that 'rich' contextual information on the studied organisations can be provided (Chetty 1999; Bryman 2008). However, in this study, such information is confined to few company characteristics presented in a table form; a more elaborate description of the companies could have improved the value of the findings and helped others to assess their transferability to other settings (Bryman 2008; Flyvbjer 2006). Then, of course, there are the known limitations of the case study approach in general: even when multiple case studies are performed, the results cannot be generalised as such. With hindsight, qualitative interviews with a few dozen carefully selected companies could have better served our purpose than a multiple study design. On the other hand, intensive collaboration with a few organisations allowed us to build better 'rapport' with the interviewees and make them openly discuss sensitive issues such as legally 'shady' attempts to circumvent licensing terms. Further, we also find it interesting that certain clear regularities/similarities in management perspectives emerged despite the heterogeneity of the cases, suggesting that the proposed concepts do have some broader relevance.

Looking at the case study companies, we see that the companies with the most 'exploitative' relationship with FLOSS are embedded systems providers and positioned as subcontractors in the value chain. On the other hand, the companies which are most deeply involved in open innovation

		Managerial perspective		
Pattern of involvement	**Managerial attitudes**	**Main Benefit**	**Main challenges**	**Challenge management techniques**
External innovation	Exploitation attitude: -FLOSS is a bulk resource - FLOSS is a 'free lunch'	- Savings in development costs and time - Efficient resource utilisation	- FLOSS-related knowledge acquisition - Quality assurance - FLOSS licensing - Release maintenance	- Passive management (dependence on individual employees) - Dynamic and static testing - Use of automatic licensing tools and empty glue code
Open innovation	Contribution attitude: - FLOSS as an integral element of value (co-)creation - FLOSS seen as an effective way to shape technology	- Value co-creation with communities - Inter-organisational learning - Global networks for marketing	- Quality assurance - Maintaining the FLOSS talent pool - Open source piracy	- Utilisation of social networks to gain quality-related knowledge - Co-development and co-testing with customers - Adopting peripheral FLOSS participation as part of employee training - Strong FLOSS licensing schemes, particularly GPL

Table 3. Summary of the case study results

are software service providers with direct contact to end clients. For embedded systems providers, software forms a cost rather than a profit centre (cf. West 2007) and, thus, it might not be surprising that these companies stated cost savings as the biggest drivers for using open source software. Thus, even though 'widget frosting' can be considered as one of the most pivotal open source business models (e.g. Hecker 1999, Henkel 2006), in our study it not did seem to embrace innovations from community or interaction with the communities. For those companies who acted as subcontractors, contributing was also constrained by the fear of (customer's) sensitive information leaking out. In turn, the bespoke software companies that seemed more dependent on FLOSS communities as their main value proposition are focused around open source software. The core competence of these companies – services – is enhanced by the outside effects of open source and open innovation and there might be less threat of sensitive data or know-how leaking outside. It is obviously impossible to make conclusions in this regard based on six case studies, because the above relationships are probably incidental. However, the study suggest that differences in the value chain position and the related networked business models may create restrictions in the ways that open source can be applied. Beyond the software industry, prior studies (e.g. Savitskaya et. al. 2010) have found correlations between the value chain position and the 'openness' of innovation practices. It seems that more research is required to explore how the value chain position influences the companies' motivation and ability to contribute to FLOSS. One could also have deeper look at well-known open source business models to see whether they encourage a 'mindset' of exploitation or contribution. Because sustaining the pool of contributors is necessary for the long-term survival of the FLOSS phenomenon (cf. Hippel 2003, Dahlander and Magnuson 2005), the question is hardly trivial.

Alternatively, the difference between 'external innovators' and open innovators' could also be explained in terms of leadership (cf. Sanchez et. al. 2011): the former seem to apply passive, and the latter, an active management approach towards FLOSS. For example, external innovators acknowledge the 'uncontrollability' of FLOSS as a given business risk while open innovators participate in FLOSS communities for the very reason of being able to control. For another example, external innovators passively rely on the enthusiasm of individual employees as FLOSS 'hobbyists', while top managers in open innovation companies are inputting their own 'energy and passion' in order to catalyse active FLOSS participation. The relationship between leadership style and FLOSS involvement might also be an interesting subject for further enquiry.

6. Conclusions

The study investigated the management perspective towards open source collaboration and networked innovation in six software companies. From our empirical data, two opposite managerial views on community collaboration arose. The first view sees community participation as a cost or an unnecessary burden by the management. The open source community is seen as a resource pool of some kind, only in terms of a free-of-charge software artefact, and company interaction with the communities is limited to minimal. The second view is a complete opposite one, in this view collaboration with open source communities is seen as an investment. As a return of their investment, these companies expect opportunities for global marketing and inter-organisational learning as well as cost savings through pooled R&D. The latter view is compatible with the open innovation paradigm, while the former could be better described as 'external innovation'. The difference between the two managerial views could be explained in terms of industrial domain, value chain position, leadership style or even open source business models. More research is required to understand what causes a company to adopt either managerial perspective on FLOSS.

References

BADEN-FULLER, C. (2004). A Knowledge Accessing Theory of Strategic Alliances. Journal of Management Studies, 41(1), 1467-6486.

BONACCORSI, A. Rossi, C. Giannangeli, S. (2004). Adaptive Entry Strategies under Dominant Standards - Hybrid Business Models in the Open Source Software Industry. SSRN Electronic Journal, 1-23.

BRYMAN A. (2008). Social Research Methods. Third Edition. Oxford University Press, New York.

CHESBROUGH H. (2003). Open innovation: the new imperative for creating and profiting from technology. Harvard University Press, Boston.

CHESBROUGH, H (2006). Open Business Models: How to Thrive in the New Innovation Landscape. Harvard University Press, Boston.

CHETTY, S. (1996). The case study method for research in small and medium sized firms. International Small Business Journal 15(1), 73–85.

COHEN W. Levinthal, D. (1989). Innovation and learning: the two faces of R&D. The Economic Journal, 99(397), 569-596.

DAHLANDER, L. (2005). Appropriation and appropriable in open source software. International Journal of Innovation Management, 9(3), 259-285.

DAHLANDER, L. (2007). Penguin in a new suit: a tale of how de novo entrants emerged to harness free and open source software communities. Industrial and Corporate Change, 16(5), 913—943.

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

DYER, J (2000). Collaborative Advantage: Winning Through Extended Enterprise. Oxford University Press, New York.

FAVARO, J. Pfleeger, S. (2011). Software as a Business. IEEE Software, 28(4), 22-25.

FLYVBERG, B. (2006). Five Misunderstandings About Case-Study Research. Qualitative Inquiry, 12(2), pp-245.

GRAND, S. Von Krogh, G. Leonard, D. Swap, W. (2004) Resource allocation beyond firm boundaries: A multi-level model for Open Source innovation. Long Range Planning, 37(6), 591—610.

HAGEL, J. Brown J. (2011) Creation nets: harnessing the potential of open innovation. Journal of Service Science, 1(2), 27-40.

HECKER, F. (2009). Setting Up Shop : The Business of Open-Source Software. IEEE Software, 16(1), 45-51.

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

HOPNER, J. (2004). The GPL prevails: An analysis of the first-ever Court decision on the validity and effective of the GPL. SCRIPT-ed, 1(4), 628—635.

HENTTONEN, K. Matinlassi, M. (2007). Contributing to Eclipse - a case study. Proceedings of the Software Engineering 2007 Conference (SE2007). Hamburg, Germany, 27-30 March, 2007.

HENTTONEN, K. (2011). Libre Software as an Innovation Enabler in India: Experiences of a Bangalorian Software SME . In: Hissam S. Russo B. (Eds.), Open Source Systems: Grounding Research. IFIP Advances in Information and Communication Technology, Volume 365, Springer, Boston. pp. 220-232.
HIPPEL, E (2003). Open source software and the private-collective innovation model: Issues for organization science. Organization science, 209-223.

HIPPEL, E (2005). Democratizing Innovation. Cambridge, MA: MIT Press.

IDRISSIA, M. Amaraa, N. Landrya, R. (2012) SMEs' Degree of Openness: The Case of Manufacturing Industries. Journal of Technology Management and Innovation, 7(1), 186-210.

KEUPP, M. Gassmann, O. (2009). Determinants and archetype users of open innovation. R&D Management, 39(4), 331-341.

KING, N. (1998) Template analysis. In: Symon G. Cassell C. (Eds.), Qualitative Methods and Analysis in Organizational Research: A Practical Guide. Sage Publications, California. pp. 118-134.

KOSKELA K. Koivumäki T. Näkki P. (2011) Art of opennesses. In: Pikkarainen M. Codenie W. Boucart N. Heredia J. (Eds.), The Art of Software Innovation, Springer, Berlin. pp. 57-68.

KROGH G., Spaeth S. Lakhani K. (2003) Community, joining, and specialization in open source software innovation: a case study. Research Policy, 32 (7), pp. 1217–1241

LERNER, J. Tirole, J. (2002). Some Simple Economics of Open Source. Journal of Industrial Economics, 50(2), 197-234.

LUNDEL, B. Ling, B. Lindqvist, E. (2006). Perceptions and uptake of open source in Swedish organisations In: Damiani, E. Fitzgerald B. Scacchi, W. Scotto M. Succi G. (Eds.) Open Source Systems, IFIP International Federation for Information Processing, Volume 203, Springer, Boston. pp. 155-153.

LUOMA T. Paasi J. Valkokari, K. (2010). Intellectual property in inter-organizational relationships - Findings from an interview study. International Journal of Innovation Management, 14(3), 399–414.

McGOWAN, D. (2005) Legal aspects of free and open source software, In: Feller J. Fitzgerald B., Hissam S., Lakhani K. (Eds.), Perspectives on Free and Open Source Software, MIT Press, London. pp. 211–226.

MÄKI-AISALA, P., Matinlassi M. (2006). Quality Assurance of Open Source Components: Integrator Point of View. Proceedings of the 30th Annual International Computer Software and Applications Conference, Second International Workshop on Testing and Quality Assurance for Component-Based Systems, Chicago, September 17-21, 2OO6, 189 - 192.

NYMAN, L. Mikkonen, T. (2011). To Fork or Not to Fork: Fork Motivations in SourceForge Projects. International Journal of Open Source Software and Processes, 3(3), 1-9.

OKSANEN, V. (2006). State of Art on Legal research on FLOSS. In: Helender N. Martin-Vanhanen H. (Eds.), Multidisciplinary Views to Open Source Software Business. BRC Research Report #33. Tampere University of Technology and the University of Tampere, Tampere. pp. 10–19.

PAASI, J. Luoma T. Valkokari, K. (2010). Knolwedge and Intellectual property management in customer supplyer relationships. International Journal of Innovation Management , 14(4), 629–654.

SANCHEZ, A. Lago, A. Ferràs X. Ribera J. (2011). Innovation Management Practices, Strategic Adaptation, and Business Results: Evidence from the Electronics Industry. Journal of Technology Management and Innovation , 2(6), 14-39.

SAVITSKAYA J., Salmi P, Torkkeli M. (2010). Barriers to Open Innovation: Case China. Journal of Technology Management and Innovation, 5(4), 10-21.

SPILLER, D., Wichmann, T. (2002). Floss Final Report – Part 3: Basics of Open Source Software Markets and Business Models. University of Maastricht, Netherlands.

STAM, W. (2009). When does community participation enhance the performance of open source software companies? Research Policy, 38(8), 1288-1299.

TUOMI, I. (2002). Networks of Innovation, Oxford University Press, New York.

VALKOKARI, K. Paasi J. Luoma T. Ling, N. (2009). Beyond open innovation: The concept of networked innovation. In: Huizing, K. Conns, S. Torkkeli M. Bitran, I. (Eds) Stimulating Recovery - The Role of Innovation Managemen, International Society for Professional Innovation Management (ISPIM), New York.

WEST, J. (2007). Value Capture and Value Networks in Open Source Vendor Strategies. 40th Annual Hawaii International Conference on System Sciences (HICSS), January 2007.

WEST, J. Gallagher, S. (2006a) Patterns of Open innovation in open source software development. In: Chesbrough, H. Vanhaverbeke W. West J. (Eds.), Open innovation:researching a new paradigm, Oxford University Press, Oxford.
West, J. Gallagher S. (2006b). Challenges of Open innovation: the paradox of firm investment in open source software. R&D Management, 36(2), 316-331 .

YE, Y. Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. The proceedings of the 25th International Conference on Software Engineering (ICSE'03), 3-10 May 2003, 419- 429.

V

LIFECYCLE MANAGEMENT IN GOVERNMENT-DRIVEN OPENSOURCE PROJECTS – PRACTICAL FRAMEWORK

by

Katja Henttonen, Jukka Kääriäinen, & Jani Kylmäaho, 2017

International Journal of Information Systems and Project Management, 5(3), 23-41

https://doi.org/10.12821/ijispm050302

International Journal of
Information Systems and Project Management

ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

Lifecycle management in government-driven open source projects – practical framework

Katja Henttonen
VTT Technical Research Centre of Finland
Tietotie 3, 02150 Espoo
Finland
www.shortbio.org/katja.henttonen@vtt.fi

Jukka Kääriäinen
VTT Technical Research Centre of Finland
Kaitoväylä 1, 90530 Oulu
Finland
www.shortbio.org/jukka.kaariainen@vtt.fi

Jani Kylmäaho
National Land Survey of Finland (NLS)
Opastinsilta 12C, 00521 Helsinki
Finland
www.shortbio.org/jani.kylmaaho@nls.fi

Abstract:
In many parts of the world, public sector organizations are increasingly interested in collaborating across organizational (and even national) boundaries to develop software solutions under an open licence. However, without sound lifecycle management practices, the full benefits of open collaboration are not achieved and projects fail to achieve sustained success. This paper introduces a lifecycle management model and framework for government-driven open-source projects and reports about its use in a real-life case study. Our focus is on lifecycle management activities which take place between deployment and end-of-life. The framework was developed iteratively through a series of focus group discussions with representatives of public sector organizations. After the framework had been taken into use in our real-life case project, individual qualitative interviews were conducted to collect experiences on its benefits and weaknesses. According to the initial evidence, the deployment of the framework seems to have brought concrete benefits to the project, e.g. by contributing positively to community growth, software quality and inter-organizational learning.

Keywords:
public information systems; open source; open-source software; free software; e-government; public sector; software lifecycle management; software evolution; information systems; public sector.

DOI: 10.12821/ijispm050302

Manuscript received: 1 May 2017
Manuscript accepted: 10 September 2017

1. Introduction

In many countries, governments agencies have started to open up bespoke software developed with public funding, often by releasing it under an open source license [1]–[3]. This may stem from governments' desire to spur innovation (by letting all citizens to gain from software at no additional costs) and/or to improve transparency (e.g. by making source code of an electronic voting system subject to public scrutiny) [2]. Another key rational behind open sourcing is a belief that other government agencies, who have similar software development needs, can reuse the software [1]–[3].

For example, in Finland, it was noticed that public sector organizations did not sufficiently co-operate on the field of bespoke software development [1]. In the absence of inter-agency collaboration, software vendors could charge each administrative unit a full price for the same or similar customizations and, thus, in the worst case, the same piece of code was purchased multiple times with tax-payer money [1]. For these reasons, the Finnish Ministry of Finance [4] and Public Administration Recommendations [5] have started to encourage public sector organizations to co-purchase bespoke software and publish it under an open-source license.

However, avoiding duplicate effort by open sourcing is not straightforward. It may be difficult for other organizations to exploit the source code purchased by one organization, e.g. due to lack of support and maintenance, multiple parallel development paths and uncertainty on the future development direction [1], [3], [6]. Therefore, there is a need to build public sector communities around these software initiatives to collaboratively manage the lifecycle [1], [2], [6], [7].

To address these issues, Kääriäinen et al. [1] developed a model where public sector agencies co-produce and co-maintain open-source software products together. However, at the time, the model had not been tested in any organization and its presentation remained abstract. This article concretizes the model introduced by Kääriäinen et al. [1] and demonstrates its practical value. The aims of the study are two-fold: firstly, to develop a practical framework that facilitates adoption of the model and, secondly, to use the framework for organizing collaborative lifecycle management in a real-life case study. The case study is an open source spatial data visualization software called Oskari, which is currently being co-produced by more than ten public sector organizations and companies in Finland.

The authors have studied the concept of the lifecycle management previously focusing on the development phase of the software (SW) product [8]. The emphasis of this article is on the lifecycle management actions taken after the implementation of the first software version i.e. how the developed SW product under the operation and maintenance could be collaboratively maintained and further developed by the group of public sector organizations.

The article is structured as follows. The next section covers theoretical background and related work, reviewing different approaches to change/lifecycle management in software production and summarizing studies on open-source lifecycle management and government-driven open-source software development. The third section introduces the model on which the framework has been built. The fourth section introduces the research approach and methodology. The fifth section introduces the practical framework which supports the deployment of the model. The sixth section demonstrates the deployment of the model and the framework of the Oskari project and reports on the experience gained. Finally, discussion and conclusions are drawn.

2. Theoretical background and related work

2.1 Lifecycle management and software evolution

Software lifecycle management (SLM) is herein understood as a process of coordinating activities and managing resources (e.g. people, money, documentation, technical artefacts) during the entire lifecycle of a software product, from initial ideation to retirement [9]. This definition comes from the application lifecycle management (ALM) literature, but similar issues have also been addressed by studies on software configuration management (SCM) and software evolution. However, SLM is different from software product management (SPM), which focuses solely on managerial actions taken before customer delivery of a software product [10].

Application lifecycle management (ALM) is a relatively new concept [11]. Chappell [12] presents ALM as a combination of three functions: governance, development and operations – and three milestones: (start of) ideation, deployment and end-of life. Development takes place at the beginning of the lifecycle, between ideation and deployment, and then periodically (after deployment) when the application is updated. Operations, which involve monitoring and deployment of updates, always happen after deployment of the first software version. Governance, which means supervising the software's evolution towards predefined goals, is needed during entire lifecycle. The emphasis of this article is on lifecycle management actions which take place between deployment and end-of life. Out of the three functions, most attention is given to governance but development and operations issues are also touched.

Software Configuration Management (SCM) is a much older discipline and can be seen as the basis upon which ALM is founded [13]. SCM is essentially about controlling and tracking changes to the software, and it has been discussed in the literature for more than three decades [14]–[17]. SCM research has significantly impacted software engineering practices [18]. The (sub)areas of SCM provide techniques for change control boards, defect tracking, build and release management, versioning and team/workflow management, for example [15], [17].

The term software evolution was originally used to differentiate from software maintenance which, at the time, was seen as a post-deployment activity consisting only of bug fixes and minor adjustments [19]. Early software evolution literature [20] noted that requirements continue to change and software needs to be adapted during its entire lifetime. Because the idea of iterative software development has become widely accepted, some authors use the terms software maintenance and software evolution synonymously [19]. However, there are two prevalent perspectives to software evolution, dubbed 'what/why' and 'how' by [19]. The former (what/why) refers to academic research on the nature of the software evolution phenomenon, its driving forces and impact [19], [21]. The latter (how) refers to engineering studies on practical means (e.g. technology, methods, tools) to direct, implement and control software evolution [22]. The focus of this article resembles the 'how perspective' on software evolution. However, the authors felt that when talking about the purposeful actions taken to ensure that a software product develops in the desired direction, lifecycle management is a more suitable term.

2.2 Open-source software production in the public sector

The term open source can be used to refer either to a licensing model or a software-development model [23]. Open-source licensing allows anyone to access the source code of the software, modify the software as desired and share it with others by redistributing a modified or unmodified version [24]. As a development model, open source refers to projects where relatively loosely coupled individuals and organizations collaborate to co-develop a piece of software together, typically working over the Internet in a distributed environment [25], [26]. Practices typically associated with open-source development include agile development, meritocratic governance and volunteer participation [26] for example.

During the last decade, government agencies all over the world have also become interested in open-source software development. Several communities or repositories for public sector open-source software development have sprung up, e.g. European-level Joinup, Finnish Yhteentoimivuus and Government GitHub. Joinup is meant for sharing and reusing open-source software, semantic assets and other inter-operability solutions for public administrations. Yhteentoimivuus is a delivery channel for public sector interoperability assets administrated by the Finnish Ministry of Finance. Government GitHub allows government agencies to share code and data on the social coding platform GitHub. While some government-driven open-source development projects have been abandoned, many others are active and continue to grow: CONNECT Health, OskarEMR, WorldWind and CAMAC, for example. Surprisingly, while there is a large body of research on open-source adoption by government organizations, e.g. [27]–[30], very few studies have looked at open-source production by government organizations. The latter are reviewed below.

Mergel [3] studied a context where government agencies share code through a common repository but do not form an open-source project or otherwise co-ordinate collaboration. The most common activity was found to be forking: participants copied the code release of another organization and then possibly modified it for their own needs internally [3]. Contribution back to the original project was not usual [3]. In other words, participants seemed to favor the relatively passive process of 'copy and reuse' over active collaboration. These findings on government code-sharing mechanisms are in line with [1]: the absence of lifecycle management practices leads to multiple forks and therefore the potential benefits of collaborative development are not fully achieved.

Bryant and Ramsamy [31] analyzed ten open-source projects where public sector agents are key contributors. There are also few academic case studies on specific open-source collaborations in the public sector [32]–[34]. Studies demonstrate that organizational and political factors play a large role in government open-source projects [31], just like many other IS projects in public sector [35]. Success factors include trust between key stakeholders, skilled in-house ICT personnel and steady financial support [31], [34]. Projects were found to be particularly vulnerable to sudden changes in political leadership and loss of key personnel [31], [34] . Some studies underline the importance of retaining the agility/flexibility inherent in the open-source model [31] while others emphasize managerial control [32]. Interest conflicts are also a common challenge. Feldman and Horan [32] note that public and private sector participants had varying perceptions of value propositions. Bryant and Ramsamy [31] report that end users experienced difficulties in making their 'voices heard' over bureaucrats whose budgets paid for the development.

2.3 The community-based software lifecycle management model

Kääriäinen et al. [1] introduced the community-based software lifecycle management model (CO-SLM) aimed particularly at public sector organizations that finance and develop software collaboratively. In this model the term lifecycle management refers to actions taken after the implementation of the first software version. The model is applicable to free/open-source software development but also to other collaborative development models, as long as the licensing is sufficiently permissive to prevent vendor-locking and allows sharing of source code with other organizations. The model is depicted in Figure 1. The community has a common repository where the baseline version of the software product is stored. Each organization can use their own software supplier to take care of deployment, maintenance and customization of the software. However, they are encouraged to inform the rest of the community on changes made and contribute them back to the baseline version for integration. The integration work is coordinated by a 'product manager' and financed as agreed by the community (e.g. costs are equally shared by the community members). Parallel baseline versions are not maintained. The inclusiveness and openness of the development process are safe-guarded via a 'community manager' role.

According to Kääriäinen et al. [1], the core community consists of public sector organizations which have primary authority over lifecycle management decisions. Thus, the community becomes a key decision-making arena: individual government organizations can influence the development goals and evolution of the baseline software product by participating in the community. Very much like in 'traditional' open-source projects, the community can also become an arena for collaborative learning and knowledge sharing (e.g. sharing solutions to common deployment problems) and even collective innovation (e.g. ideating new functionality). Outside of the core community but still functioning as key partners are software companies that are tendered to develop the software [1]. However, in some cases companies may also participate in the community as full community members if the intention is to support the application of the software for the private sector as well (note that the model itself does not limit this). The community manages the software according to the lifecycle management plan initiated by the financier of the first version [1]. The plan defines who will do what and when in relation to the lifecycle management activities (e.g. documentation requirements, versioning model, change and release management practices and financing). Basically, this is a similar job that companies make for software products they own. Similarly, companies have product managers who are responsible for coordinating the lifecycle management actions to software products. However, when the group of public sector organizations start to jointly manage SW products the case is just more complex since there has to be found a consensus between the organizations what are the responsibilities, financing model, rights, etc.

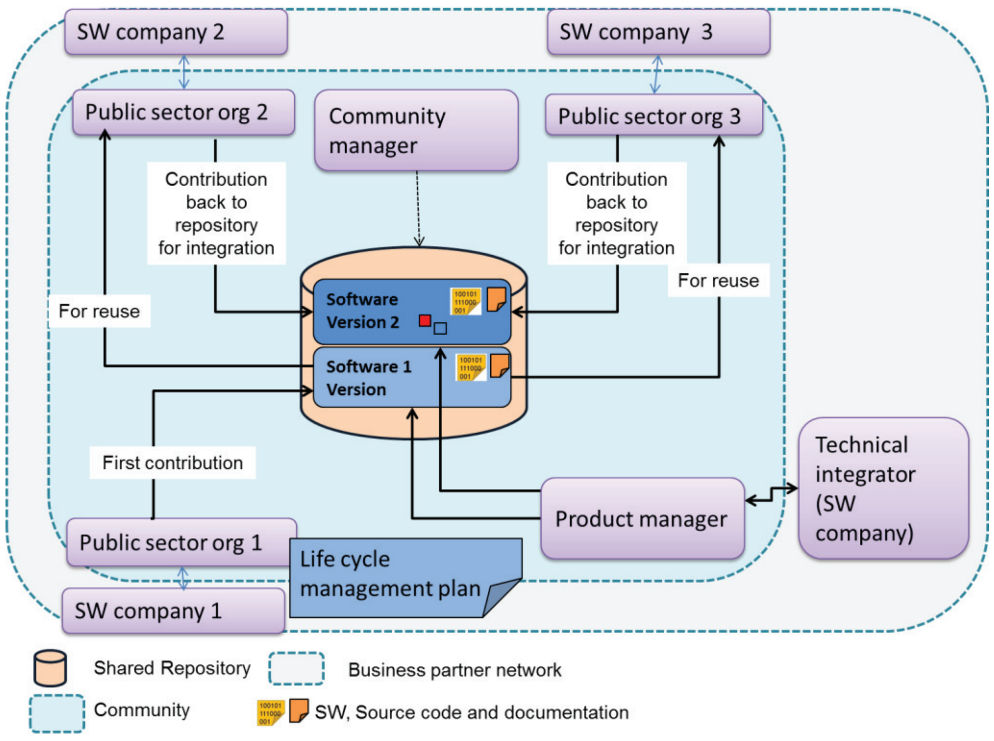


Fig. 1. Community-based Application Lifecycle Management Model

3. Research approach and methodology

The Ministry of Finance, our organization and a number of public sector organizations have collaborated in defining, piloting and deploying lifecycle management as depicted in Figure 2. The model creation process and the introduction to the models were published in [1]. This effort has since continued by piloting the planning of software lifecycle management in practice in the public sector. After successful piloting, the deployment of this model started in public sector organizations (the Finnish Ministry of Finance has accepted the model for production).

Prior and during the pilot phase, we developed the CO-SLM framework that is introduced in this article. The CO-SLM framework is a check list and documentation template to facilitate the definition of project-specific lifecycle management plans for software products. It helps software product communities to define a lifecycle management plan that describes who will do what and when related to the lifecycle management activities in the public sector software community environment. The framework has been tested and refined through deployment in real organizations.

The research presented in this article has an interpretive and an interventionary stance and, therefore, the approach could be described as an 'action case'. The term 'action case' was originally coined by Vidgen and Braa [36] to describe in-context information systems (IS) research which both aims to accumulate rich understanding on an organizational dilemma (interpretation) and to change the status quo in that organization (intervention). The interventionist phase typically takes place later in the research and involves the testing of the previously developed methods [36].

The CO-SLM framework was developed through iterative rounds of data collection and analysis. The primary data collection method was a series of six focus group discussions with information systems experts working in public sector organizations. There were participants from both municipal (e.g. City of Espoo, the Association of Finnish Local and Regional Authorities) and national (e.g. State Treasury, Finnish Ministry of the Environment) levels of government. In addition, the representative from the Finnish Centre for Open Systems and Solutions (COSS) and a lawyer appointed by the Finnish Ministry of Finance participated some of the sessions. In the discussion sessions, participants were prompted to assess draft versions of the framework and their feedback was used to improve the framework iteratively. The point of saturation was reached after six consequent meetings. In addition to the focus group discussions, six Finnish, SME-sized software companies were asked to reply interview questions by email and clarifications were asked on the phone where necessary. These companies were selected for email interview due to prior collaborations with the public sector and consequent good knowledge of the domain. Complementary data collection methods also included few workshops with the Finnish Ministry of Finance and informal discussions with different stakeholders.

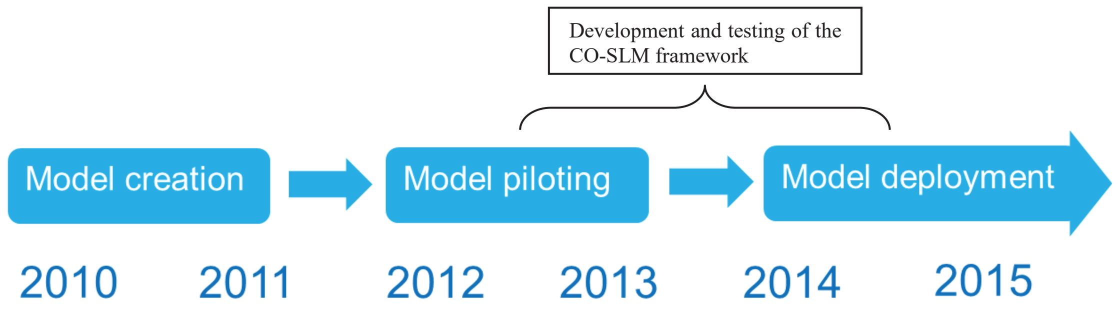


Fig. 2. Timeline for the development of the CO-SLM model

The testing and modification of the CO-SLM framework took place ‘in situ’ at multiple organizations during the years 2014 and 2015. For example, the National Land Survey of Finland (NLS) and the Ministry of Finance’s JulkICT Lab project have adopted the model for their operation. This article explains and analyses its deployment within “Oskari”, an NLS-led project which develops an open source geospatial toolkit. The reported experiences are based on two sources: 1) analytical observations from two of the authors who have been engaged in the Oskari project for a long time and (b) lengthy, semi-structured interviews of representatives from organizations who are key contributors to Oskari: National Land Survey of Finland (NLS), Finnish Transport Agency and The Finnish National Board of Antiquities. The interviewees were senior professionals who have co-ordinator responsibilities in the Oskari project, either in technical development or communications. Four out of five interviews were recorded and all were selectively transcribed. The thematic coding of the interview data followed a method called Template Analysis [37].

4. Framework for community-based lifecycle planning (CO-SLM framework)

In this chapter, we introduce a practical framework which helps with the application of the CO-SLM model into real-life situations where public sector organizations wish to develop software collaboratively. The framework focuses on the governance aspect of lifecycle management. The origins of the framework come from the SCM research area. One part of the SCM is a planning activity that forms an SCM plan [38]. The basic idea of the SCM plan is to define who is going to do what, when, where and how in relation to the configuration management [39]. When applied to the context of CO-SLM the goal is to help a consortium of public sector organizations to define what to manage, who will do the management, how the management will be done and how to finance the management and further development of a software product. Financing practices were included in the framework because collective purchasing and cost sharing is a significant and obvious concern for public sector organizations. Figure 3 depicts the four main elements of the product-management plan and Tables 1-4 present each of them in detail. The framework can be used as a check list and template to form a lifecycle plan for a software product that needs governance during its lifecycle. The following four tables then describe each element in detail. Each element contains issues that need to be considered and documented for any software under management. Therefore, when applying the CO-SLM model and CO-SLM framework it should be borne in mind that each software product – and its associated community – is unique. Thus the model and framework must be applied to suit the context. This means making adjustments to terminology and content when applicable.

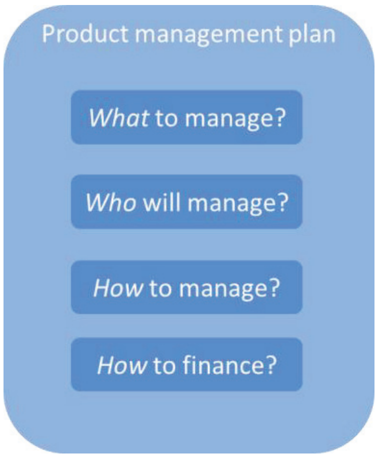


Fig. 3. Elements of the product-management plan

Table 1. What to manage?

Issues	Description
Name of the software	What is the name of the software program?
Licensing scheme	What are licensing terms for the source code and documentation?
User organizations	Which organizations will use the software?
Schedule for the first version	When is the baseline version of the software schedule to be ready?
Distribution channel	Where are the source code and documentation distributed?
Social media	Which social media channels are used by the project?

Table 2. Who will manage?

Issues	Description
Owner of the software product	Who 'takes care of' the software product? Who owns the copyright to the software?
Community structure and membership	How is the consortium of organizations structured? Are there community and steering groups? Who has the highest decision-making authority concerning the software product and its evolution?
Product manager (Development Co-ordinator)	Who supervises the software's evolution towards the commonly agreed goals? How is the role mandated? Coordinates the software product-related lifecycle management activities so that the software product evolves in the direction that serves the needs of the community and business.
Community manager (Openness Co-ordinator)	Who consolidates conflicts and protects the inclusiveness and openness of the development process? How is this role mandated? Coordinates the operation in the community. Checks that the licence is used as agreed by the community.

Table 2. Who will manage? (cont.)

Issues	Description
Repository maintainer(s)	Who maintains the shared repository containing the source code and documentation?
Technical integrator (Baseline Developer)	Who develops and maintains the baseline version of the software? Who is responsible for integrating the desired customisations to the baseline as agreed by the community? If the integrator function is outsourced, who does the tendering?
Providers of customisation and deployment services	Who can provide customisation and deployment services on the software product to individual member organizations?

Table 3. How to manage?

Issues	Description
Decision-making bodies	Responsibilities for making managerial and technical decisions regarding software development. How are the decision-making bodies (e.g. managerial board, change control board, steering group) organised, elected and assembled for a meeting?
Collaborative development approach	What are the key principles guiding collaborative development? How are the development efforts co-ordinated?
Road mapping	Who is responsible for creating and updating the roadmap documents? Who is responsible for accepting a new roadmap? Where are the documents located?
Change management	Who can initiate change requests, and how? Who analyses the change requests? How are requests prioritised? Who makes the final decision on what is included in the next software version?
Release management and versioning	How often are releases made? Who accepts a new baseline version for deployment? How are versions named/numbered?
Urgent bug fixes	Who/how to handle urgent bug fixes required to the baseline version already in deployment?
Communications	Who defines and supervises the community's communication strategy? What are the primary channels for internal and external communication?
Documentation	What documents are required and where are they located?

Table 4. How to finance?

Issues	Description
Co-ordination work	How are the efforts of the product and community manager financed?
Repository maintenance	How is repository maintenance financed?
Baseline development (Technical integrator)	How is the further development of the software product financed, including integration of external contributions? How to finance the bug fixes?
Deployment and customisation	Who pays for deployment and customisation work within an individual organization?
Community and steering group meetings	Who pays for organising and participating in the community meetings and steering group meetings?
New entrants	Who can join the community and how? Are there any joining fees?

5. Case study: Oskari software

This chapter presents a real-life case study where the CO-SLM framework has been applied. This chapter is structured as follows. The first sub-section introduces the Oskari case study. The second sub-section presents the lifecycle management plan for Oskari, which is based on the CO-SLM framework and briefly explains how the plan was made. The third sub-section reports on the benefits and challenges of lifecycle management as well as the experiences of using the framework in practice.

5.1 Introduction to the OSKARI case

Oskari is an open source software originally developed by the National Land Survey of Finland. Initially, Oskari was developed to offer easy-to-use browser-based tools to access and re-use information from various data sources, including the INSPIRE Spatial Data Infrastructure (SDI) and the Finnish National SDI. Oskari software has been adopted by about a dozen public sector organizations in Finland, including the City of Tampere, Finnish e-Government portal, Finnish Transport Agency and the Helsinki Regional Environmental Authority. Two major international co-operation projects utilising Oskari are currently running: European Location Services (ELS) and the Arctic Spatial Data Infrastructure (ASDI). The first independent Oskari installations are also emerging outside Finland: the National Land Survey of Iceland has set up Oskari, followed by Agency for Land Relations and Cadastre of the Republic of Moldovia.

Oskari makes it possible to view, visualize, analyze and even edit spatial data using just a web browser and standards-compliant APIs, such as OGC WMS (Web Map Service), OGC WFS (Web Feature Services) and OGC WPS (Web Processing Service). One of the most used features of Oskari implementations is the embedded maps functionality. It enables the user to choose applicable map layers and to create a map client using a WYSIWYG user interface without programming skills. The embedded map client can then be placed on any website in a similar manner as in Google Maps, just by placing a piece of HTML code into the website. The difference is that Oskari leverages standards-compliant APIs, which means that there are thousands of spatial data resources to choose from.

The Oskari network is a consortium of organizations that have entered into a formal agreement to co-develop the Oskari software. Oskari is published under open source licenses (MIT and EUPL) and therefore anyone can download the source code and utilise the software without joining the Oskari network. This means that anyone can try the software without committing to it or even without letting the network know about it, or use and extend it as they see fit. However, it is the appointed representatives of the steering committee member organizations who oversee which developed features or changes are integrated to the Oskari repository. The most important benefit of the steering committee membership is the ability to get support from other organizations and agree on the development goals together.

5.2 Lifecycle management plan for Oskari

The CO-SLM framework was used as a template and instructive guide when writing the lifecycle management plan for the Oskari software. The first draft of the plan was created by collecting existing practices found from websites and documents. Then the plan was discussed and refined to fill in any missing information. The plan template was also modified to be in the line with the terminology of software products and the software community. Finally, the plan was reviewed by the key members of the Oskari software team (the coordinator and the chairperson of the steering committee) and the plan was discussed and agreed (Version 1.0) in a steering committee meeting. The resulting plan is presented in the following tables 5-8 below.

Table 5. Basic information about the Oskari software

Issues	Details
Name of the software	Oskari
Licensing scheme	Open Source. Source code can be utilised using an MIT licence or EUPL licence.
User organizations	Public sector organizations, companies, non-profit organizations.
Schedule for the first baseline version	First public version was released 2011 (first version was financed by the National Land Survey Development Centre).
Distribution channel, repositories	Documentation, examples, etc.: http://www.oskari.org Source code: https://github.com/oskariorg General introduction to the software and the Oskari network (in Finnish): http://verkosto.oskari.org
Social media	Twitter: the @oskari_org Twitter channel reports new releases, bug and security fixes as well as events related to Oskari. The release plan and roadmap are presented on a Trello board (in Finnish): http://oskari.org/trello Slack: Slack is a team communication platform: https://oskari.slack.com

Table 6. Roles and organizations

Issues	Details
Owner of the software product	The Oskari network
Community structure and membership	The Oskari network is the development network for Oskari software that is open for anyone that signs the Memorandum of Understanding. Members (listed in Finnish): http://verkosto.oskari.org/oskari-verkosto/jasenet/ Organization of the Oskari Steering committee: representatives of projects that exploit Oskari and sign the Integration agreement, coordinator (chosen by steering committee) and 1-2 representatives from the Oskari network member organizations (nominated annually by the Oskari network). Members (listed in Finnish): http://verkosto.oskari.org/oskari-verkosto/ohjausryhma/
Technical Coordinator (Product manager)	National Land Survey Development Centre (Jani Kylmäaho, Inkeri Lantta) http://verkosto.oskari.org/oskari-verkosto/koordinaattori The coordinator was selected by the Oskari steering committee. The coordinator coordinates (using the available resources) the software product-related lifecycle management activities so that the software product evolves optimally in the direction that serves the needs of the network and businesses. Furthermore, the coordinator facilitates the network and its activities, provides support to the projects utilising the software and works as a secretary for the steering committee. An architecture board meets 2 to 3 times per year to discuss and agree upon changes proposed to the technical architecture.
Community manager (Openness co-ordinator)	The National Land Survey Development Centre has the responsibility for this task as well.
Repository maintainer(s)	Technical coordinator
Integrator (Baseline developer)	The coordinator takes care of the integration work. The selected integrator is responsible for technical coordination, e.g. regarding the architecture of the software core. The integrator takes care of the integration work: coding, testing, version updates, documentation and any necessary IT support. The integrator reviews pull requests proposed by contributors, maintains repositories and core documentation and manages software versions, working in close cooperation with the coordinator.
Providers of customisation and deployment services	Each customer organization that applies Oskari software may select an IT provider for Oskari customisations without limitation. Customer organizations are encouraged to follow the architecture principles defined by the Oskari network if they wish to include modifications or extensions into the Oskari software.

Table 7. Practices for lifecycle management

Issues	Details
Tasks of the decision-making bodies	The Oskari network is open for anyone who signs the Memorandum of Understanding. The agreement describes the goals, tasks and decision-making practices for the Oskari network. The network communicates information about the Oskari software and its development, as well as discussing the future needs of the software. It has a mailing list and communicates actively in social media. Network members are invited to networking days (the steering committee schedules networking days at least once a year). Agenda for networking days: ▪ Status reporting and future activities ▪ Presentations of projects and activities around the software ▪ Selection of representatives of the steering committee Furthermore, developer meetings take place and the architecture group assembles 3 to 4 times each year. The goal of the developer meetings is to collect input that supports the development of the Oskari software core. The tasks of the Oskari steering committee are: ▪ Overseeing the network and planning activities ▪ Choosing the coordinator and setting the annual fees ▪ Prioritising the roadmap ▪ Communicating with the coordinator The steering committee also checks the status of the Oskari network (new members, etc.), communications activities, ongoing development projects, planned development projects, the roadmap and updated documents. The coordinator works as a secretary of the steering committee. The steering committee can invite the representatives of development projects to introduce and discuss their projects.
Collaborative development approach	The Oskari software is reused in development projects that need to create a web map application, a geoportal or to embed map clients into other web applications. The development project downloads the Oskari software and applies it; and further develops it, if needed. The development needs will be discussed with the coordinator and other development projects to avoid overlapping development work. The project is requested to follow the Oskari architectural principles and to provide modifications (Oskari open source licence) back to the Oskari network for integration. The Oskari steering committee decides what will be integrated into the next public Oskari release (or road mapped into future releases) based on the coordinator's proposal. Development projects are requested to document new source code to facilitate reuse (a documentation guide can be found on the Oskari website). The coordinator is responsible for checking the documentation during integration.
Road mapping	The coordinator maintains the Oskari roadmap (short-term roadmap and longer-term (1 year) roadmap) and is responsible for introducing new releases in steering committee meetings. The steering committee has the responsibility of checking and agreeing on any major changes before release. The roadmaps can be found at: ▪ http://oskari.org/trello (in Finnish) ▪ http://www.oskari.org/documentation/development/roadmap (in English)
Change management	<u>Requesting changes</u>: Based on proposals from the development projects, the Oskari coordinator collects the new features that are proposed to be integrated into Oskari. Major changes in the software core are planned by the coordinator and presented to the Oskari architecture board, which discusses and agrees on the proposed changes. All other remarks and proposals will be reported as GitHub issues. <u>Change proposal</u>: The coordinator prepares the proposal. <u>Change decision</u>: The Oskari steering committee makes change decisions based on the coordinator's proposals. <u>Change implementation</u>: The coordinator arranges tendering for Oskari integration and core framework development work and makes acquisitions based on the tendering results. Tendering material templates are provided as guidance for other projects for their tendering purposes. The coordinator maintains the Oskari integration backlog in cooperation with the integrator. The coordinator updates the backlog based on the agreed integration tasks. The integrator is responsible for defining and scheduling more detailed tasks and setting foreseeable version numbers for backlogged items. The selected integrator takes care of the integration work: coding, testing, version updates, documentation and any necessary IT support.

Table 7. Practices for lifecycle management (cont.)

Issues	Details
Release management and versioning	After integration and testing, the integrator prepares a software version for release. The version numbering scheme is as follows: ▪ X = Major version with significant changes in architecture and/or APIs of the software: planned in the Oskari roadmap. ▪ Y = Minor version: planned in the Oskari roadmap. ▪ Z = Maintenance version: other small changes and bug fixes are marked with a maintenance version number. The coordinator introduces a new software release proposal (major or minor release) for the Oskari steering committee who check and agree on major integrations before the release. The steering committee is informed of any major plans to change the software core. The committee schedules the major changes to ensure smooth transition to the new version within member organizations. The coordinator takes care of any other necessary small changes and bug fixes (maintenance releases). Instructions on how to contribute to Oskari development using GitHub branches: http://www.oskari.org/documentation/development/how-to-contribute
Communications	The Oskari steering committee is responsible for the communication plan. The coordinator prepares change requests to the communication plan and the steering committee agrees them. The coordinator is responsible for implementing the communication activities as scheduled.
Documentation	Functional specification: http://www.oskari.org/ User guides: Developer guides for applying Oskari: http://www.oskari.org/ http://oskari.org/examples/rpc-api/rpc_example.html End-user guides: ELF service http://demo.locationframework.eu/ National Geoportal Map window: http://www.paikkatietoikkuna.fi/web/en/user-guide Installation and operational environment: http://oskari.org/documentation/ Technical description and instructions for Oskari developers: http://oskari.org/documentation/

Table 8. Financing practices

Issues	Details
Coordinator	Mostly integration fees collected from organizations who have signed the Integration agreement.
Community manager	Financed as part of the coordinator's work.
Integrator	Will be financed by the partners who have signed the Integration agreement (annual integration fee). The coordinator and development projects can also negotiate the sharing of integration costs if the integration fee turns out to be too low.
Oskari development	National Land Survey Development Centre/SDI team. Oskari network. Project funding.
Deployment and customisation	Each organization takes care of its own funding to apply the Oskari software.
Network and steering committee meetings	Each organization takes care of its own participation expenses. Meeting costs are covered by the integration fee.
New entrants	Oskari network: Free of charge. New members have to sign the Memorandum of Understanding. Oskari steering committee: steering committee members (development projects) sign the Integration agreement where they agree the annual integration fee. The steering committee agrees on the annual integration fee.

6. Experiences on deploying CO-SLM

This chapter reports experiences, particularly benefits, challenges and lessons from deploying CO-SLM model and framework within the Oskari project.

6.1 Product acceptance and quality

CO-SLM model and framework provided a strong governance model for cooperation by clearly defining responsibilities and processes. As one of the interviewees point out, "lifecycle management is what actually turns an open software application into a software product…*"*. Being a product means the availability of technical support and documentation, version schemes and roadmaps for future development, for example. This allows potential users to evaluate the suitability of the software to their current and future needs. The productization also includes communications and marketing activities, which – together with the robust management model – have helped to improve the "brand" of the software and make it more attractive to adopt. Overall, many interviewees talked about the Oskari brand and its importance to project acceptance.

The Oskari project had recently entered incubation process to join the OSGeo foundation, a not-for-profit legal entity supporting the open source geospatial community. This is expected to further improve the Oskari brand and acceptance of the software product, also internationally, but the process is in early stage. Generally, presence on platforms like GitHub and OSGeo where curious outsiders can explore the software without making financial or other commitments, is seen as a key to identifying stakeholders and growing the user base. One of the interviewees expressed this as follows:

> OsGEO, GitHub and other platforms where anybody can participate in the discussion are really good. You do not have to identify all stakeholders in advance, but just throw out something and interested parties will come to you. We have received inquiries from as far as Moldova…[ ]... If you want "fresh blood" [into the project], it is great that people can start following you without commitment and then deepen their involvement gradually.

The CO-SLM model has also helped to improve non-functional qualities of the software, particularly adaptability and extensibility. When a software is developed by a single organization alone, hectic demands and limited resources can cause focusing on immediate user needs at the expense of long-terms software quality, e.g. architecture design that allows software to adopt to future needs. Consequently, the software becomes hard to maintain within a single organization and impossible to share with other organizations without significant refactoring. However, CO-SLM model forces the owner to look at the software from a wider perspective, beyond their own immediate use cases. Each modification to the baseline version is considered from the viewpoint of multiple organizations, leading to improved adaptability. One informant explained:

> Our understanding [of software design] has broadened so much after we started talking with other organizations who have similar needs. It was a bit like 'oh, right, we do not have to reinvent that wheel'. We have learned that we can develop things collaboratively even though the needs are not exactly the same".

The CO-SLM model has also helped to secure resources for developing project-wide testing methods and tools available to all member organizations. This has reportedly decreased the number of software bugs in new releases.

6.2 Resource pooling

6.2.1 Human resources

For more than two decades, public sector organizations in Finland have been inclined to outsource all their software-development activities. This has caused a shortage of skilled in-house ICT personnel in many organizations, making it harder for them to take responsibility for software development and lifecycle activities. For example, when organizations buy Oskari implementations from software companies, they may not know how to communicate key architectural principles to the companies or conduct tendering process in a manner that obligates companies to follow them.

This has sometimes led to poor-quality code contributions, e.g. new functionalities have been placed in illogical parts of code structure and/or interfaces are used in a non-standardised way. The resulting problems require significant effort from the technical coordinator (NLS) who underlines that, in the future, more effort must be put into ensuring a common understanding of proper architecture principles and their inclusion in the request for tenders.

The alternative strategy to address human resource deficits has been to acquire manpower from software companies in a form of 'body shopping'. This differs significantly from a process where public sector organizations place a request for tenders (RFT) for software implementation. As the bidders aim to offer the lowest possible price, no requirements apart from those explicitly mentioned in the RFT will be taken into account. According to both the interviewees and prior studies [35], [40], the heavy-weight requirements specification (for RTF) makes it difficult to incorporate new ideas afterwards and can thus hinder innovativeness. Oskari community has noticed that, in complex development cases, it is often better to tender for individual developers instead of tendering for specific implementations. This approach has enabled agile software development processes and intensified knowledge exchange between public- and private-sector organizations. In this model, the leadership of the software-development process stays entirely with the public sector, which again requires specific skills, different from those required by mere software acquisition.

It was also repeatedly noted that, while CO-SLM model does indeed require new skills, it also creates an environment for inter-organizational learning and thereby helps building new skills. For example, interviewees gave statements like "inter-organizational learning is a key benefit [from Oskari participation]", "even organizations who do not contribute code have provided much valuable inputs [of skills and ideas]" and "we have learned so much just by talking to other organizations with similar needs".

6.2.2 Financial resources

Because Oskari is open source licenced, any organization could download it and use it without participating in development expenses. However, the majority of organizational users have chosen to pay an integration fee. The payment ensures them a membership in the steering group and an opportunity to influence the future development of the software. By participating in the decision-making, organizations can ensure that the software will continue to meet their needs in the future. This has been enough to motivate organizations to contribute financially, and, thus, open source licensing has not lead to a significant 'free riding' problem.

Despite this, financing the Oskari baseline software development has not been easy. The relatively low integration fee (currently EUR 5,000 per organization annually) has been sufficient to cover the integration, co-ordination and communication activities of the Oskari community but not maintenance of the baseline software. Steering committee members felt it was impossible to increase the fee without forcing member organizations to go through a significant amount of bureaucracy. For long time, the development of the baseline version was paid for entirely by the National Land Survey of Finland, which made the project extremely dependent on a single organization. However, the increasing number of participants has recently improved finances and Oskari community is moving towards a model where it is less dependent on NLS funding.

In general, interviewees felt that the CO-SLM model has enabled significant savings because development cost can be split with others. One of the informants put it as follows:

> One of the major goals of this collaborative development model has been to save funds. I feel that we have achieved that… We can go to a steering group and split up tasks [between organizations], like 'you do this and I do that'… If we did not have this collaboration, we would have had to pay everything alone.

In practice, there were two ways to finance major extensions to Oskari: (1) some member organizations pay for Oskari extensions alone but comply with architectural rules and share them with others for free and (2) some organizations form 'mini consortia' with other organizations who needed the same functionality and make the requirement analysis, tendering and financing jointly. The latter was considered as more mature form of co-development but required more inter-organizational communication and trust. One interviewee explained:

> The first level is that each organization manages their own projects but follows some commonly agreed principles. Meanwhile, others are waiting to get their hands on it. This is the most common way because it is fast and easy if you have money [within one organization]. The second level involves collaborative financing; it is much more complex and requires trust. One organization is chosen as a leader and then the leader organization makes consortium agreement with other organizations to co-finance and co-develop something together.

6.3 Project sustainability

Because big money is circulated in public sector ICT procurement, successful new models, which create savings for governmental organizations, unavoidably shrink revenues of some companies. Consequently, Finnish Location Information Cluster, an advocacy group of some established companies offering geospatial solutions, has been very critical of the Oskari project and tried to create political pressure against it. While no significant harm has been caused, aggressive industrial lobbying was noted to be a risk factor which can negatively influence sustainability of any government-driven open source project. CO-SLM approach partially helped to tackle the issue, e.g. by resourcing communication and public relations (PR) activities.

However, with Oskari, the biggest sustainability challenge is to decrease the project's dependence on the coordinator, NLS, and thus make it less vulnerable to changing management interests and/or shifts in key personnel within that organization. This was expressed in several interviews, for example as follows: "even though NLS has been a primus motor in the start, there is no particular reason why it should remain as a primary or principal actor" and "other actors must take more responsibility because we [the project] should not be overly dependent on NLS".

Significant informing and marketing effort has been undertaken to attract more organizations to the Oskari network. When more organizations are participating, more financing will come in and relevant technical knowledge will be distributed among multiple organizations and people. If the software is strategically important to a sufficiently large number of organizations, the development will continue even if the NLS decides to drop out. The project is now entering a new phase as the co-ordinator role is planned to be shifted from NLS to an outsourced project organization whose costs are covered with integration fees.

7. Discussion

7.1 Lessons for researchers and practitioners

For practitioners in the public sector who consider engaging their organizations in collaborative open source projects, the case study highlights the importance of ensuring sufficient in-house IS skills. This is in line with prior literature pointing out in-house IS skills as a key success factor to open source and other agile projects on the public sector [32], [35]. Even though it is possible and often recommended [34] to exploit external experts, open source development requires the public sector to take on the responsibility of a software owner. This requires both sufficient technical competence and knowledge in software lifecycle management. To support the latter, the CO-SLM framework acts as a document template for planning software governance and lifecycle activities.

The second lesson for practitioners has to do with the importance of enabling community growth. A 'critical mass' of active organizational users helps to ensure steady funding and guarantee project continuity, even if one dominant organization drops out. This is also in line with prior studies which have emphasised versatile developer and donor bases as a success factor to all types of open source projects [41], [42]. In part, CO-SLM supports community growth by making the software more attractive to new users. This is because clarified governance processes and responsibilities make the whole process more predictable and manageable. Software must also be 'generic' enough so that it can be adapted to the diverse use cases for heterogeneous organizations. As the software is developed further, one must keep a multi-organizational perspective in mind.

For researchers and consultants, a key lesson relates to appreciating the huge diversity of organizations and software projects. Due to the heterogeneity of environments, it has proven pointless to develop a detailed predefined set of lifecycle management practices for all public sector driven open source projects. We noticed that a high flexibility of the framework is more important, taking, e.g. the form of 'check lists' on lifecycle management issues to be taken into account. Each project can then take the framework as a basis and develop lifecycle management practices suitable to their particular circumstances. If one wishes to develop 'best practices' on lifecycle management, one must focus on a particular software domain and type of application, not public sector driven OSS projects in general.

7.2 Limitations of the study and further research

While diverse stakeholders were involved in the drafting and development of the CO-SLM framework (see Section 3 for details), we did not interview all members of the Oskari network after its deployment. Because we had an opportunity to interview only people from three heavily-engaged organizations (see chapter 3), the results are biased towards their perspectives. We acknowledge that 'peripheral' members of the Oskari community may have different perspectives that are not visible in this study. We also understand that a single case study is not enough to make definitive conclusions regarding the applicability of the framework in diverse public sector environments. Our next step is to deploy the CO-SLM model and framework in other public sector open source software development efforts and, thereby, to gain further experience on their applicability in different organizational settings. We also hope to collect and analyse more qualitative and quantitative data on the supposed benefits of the CO-SLM approach.

8. Conclusions

This paper introduced the CO-SLM model and flexible framework developed for helping public sector organizations to follow sound lifecycle management practices in open source development projects. The model and the framework were successfully deployed in a real-life setting, where a dozen public sector organizations were jointly developing spatial data analysis software under an open source licence. The adoption of CO-SLM benefited the software project by encouraging community growth, improving the 'image' of the software and enhancing software quality, especially regarding software maintainability and extensibility. Challenges stemmed from deficit software development and acquisition skills in some organizations and insufficient funding due to relatively low membership fees. Furthermore, the study shows that a project's financial and technical dependence on the leading organization should be decreased in the future to lower risks and ensure long-term sustainability.

9. References

[1] J. Kääriäinen, P. Pussinen, T. Matinmikko, and T. Oikarinen, "Lifecycle Management of Open-Source Software in the Public Sector A Model for Community-Based Application Evolution," ARPN Journal of Systems and Software, vol. 2, no. 11, pp. 279–288, 2012.

[2] J. C. Colannino, "Free and Open Source Software in Municipal Procurement: The Challenges and Benefits of Cooperation," *Fordham Urban Law J.*, vol. 39, p. 903, 2012.

[3] I. Mergel, "Open collaboration in the public sector: The case of social coding on GitHub," *Gov. Inf. Q.*, vol. 32, no. 4, pp. 464–472, Sep. 2015.

[4] Finnish Ministry of Finance, "Sade-ohjelma: open source approach," Helsinki, Finland: VM, 2012. Available: https://www.europeandataportal.eu/data/en/dataset/sade-ohjelma-avoimen-lahdekoodin-toimintamalli00

[5] JHS, *JHS 169 Use of Open Source software in Public Administration*. Helsinki, Finland: JUHTA (Advisory Comittee on Information Management in Public Administration), 2012.

[6] A. Nurmi, "Coordination of Multi-Organizational Information Systems Development Projects – Evidence From Two Cases," *J. Inf. Technol. Theory Appl.*, vol. 10, 2010.

[7] T. A. Pardo, A. M. Cresswell, F. Thompson, and J. Zhang, "Knowledge sharing in cross-boundary information system development in the public sector," *Inf. Technol. Manag.*, vol. 7, no. 4, pp. 293–313, Dec. 2006.

[8] J. Kääriäinen, "Towards an application lifecycle management framework," VTT Publications, no. 759. Espoo, Finland: VTT Technical Research Centre of Finland, 2011.

[9] S. Chanda and D. Foggon, "Application Lifecycle Management," in *Beginning ASP. NET 4.5 Databases*, Berkeley, USA: Apress, 2013, pp. 235–249.

[10] K. Vlaanderen, I. Van de Weerd, and S. Brinkkemper, "Improving software product management: a knowledge management approach," *Int. J. Bus. Inf. Syst.*, vol. 12, no. 1, p. 3, 2013.

[11] G. Weiß, G. Pomberger, W. Beer, G. Buchgeher, B. Dorninger, J. Pichler, H. Prähofer, R. Ramler, F. Stallinger, and R. Weinreich, "Software engineering - Processes and tools," *Hagenb. Res.*, no. 1, pp. 157–235, 2009.

[12] D. Chappell, "What is application lifecycle management," David Chappel and Associates, 2008, Available: http://davidchappell.com/writing/white_papers/What_is_ALM_v2.0--Chappell.pdf

[13] C. Schwaber, "The Expanding Purview Of Software Configuration Management", Forrester Research, 2009.

[14] E. H. Bersoff, "Elements of Software Configuration Management," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 1, pp. 79–87, 1984.

[15] J. Estublier, "Software configuration management," *Proc. Conf. Futur. Softw. Eng. - ICSE '00*, pp. 279–289, 2000.

[16] J. Koskela, "Software configuration management in agile methods," *VTT Publications*, no. 514. Espoo, Finland: VTT Technical Research Centre of Finland, pp. 3–54, 2003.

[17] M. E. Moreira, *Software configuration management implementation roadmap*. Chichester, England: John Wiley & Sons, 2004.

[18] J. Estublier, D. Leblang, A. Van Der Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber, "Impact of software engineering research on the practice of software configuration management," *ACM Trans. Softw. Eng. Methodol.*, vol. 14, no. 4, pp. 383–430, 2005.

[19] T. Mens and S. Demeyer, *Software evolution*. Springer Berlin Heidelberg, 2008.

[20] B. W. Boehm, "A spiral model of software development and enhancement," *Computer (Long. Beach. Calif).*, vol. 21, no. 5, pp. 61–72, 1988.

[21] M. Lehman and J. C. Fernáandez-Ramil, "Software Evolution," *Softw. Evol. Feed. Theory Pract.*, vol. 27, no. 4, pp. 7–40, 2006.

[22] T. Mens, "Introduction and roadmap: History and challenges of software evolution," in *Software Evolution*, T. Mens and S. Demeyer, Eds. Berlin, Heidelberg, Germany: Springer, 2008.

[23] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/Libre open-source software development: What we know and what we do not know," *ACM Comput. Surv.*, vol. 44, no. 2, p. 7, 2012.

[24] Open Source Initiative, "The Open Source Definition," *Open Source Initiative*, 2013. [Online]. Available: http://opensource.org/osd.

[25] J. West and S. O'mahony, "The Role of Participation Architecture in Growing Sponsored Open Source Communities," *Ind. Innov.*, vol. 15, no. 2, pp. 145–168, Apr. 2008.

[26] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/Libre open-source software development," *ACM*

Comput. Surv., vol. 44, no. 2, pp. 1–35, 2012.

[27] B. Rossi, B. Russo, and G. Succi, "Adoption of free/libre open source software in public organizations: factors of impact," *Inf. Technol. People*, vol. 25, no. 2, pp. 156–187, Jun. 2012.

[28] M. Shaikh and T. Cornford, "Navigating Open Source Adoption in the Public Sector," *18th Am. Conf. Inf. Syst. (AMCIS 2012)*, pp. 1–10, 2012.

[29] J. Allen and D. Geller, "Open source deployment in local government: Rapid innovation as an occasion for revitalizing organizational IT," *Inf. Technol. People*, vol. 25, pp. 136–155, 2012.

[30] O. Jokonya, "Investigating open source software benefits in public sector," in *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2015, vol. 2015–March, pp. 2242–2251.

[31] D. Bryant and P. Ramsamy, "Public Administration Code Release Communities," Madrid, Spain: ONSFA, 2014.

[32] S. S. Feldman and T. A. Horan, "Collaboration in electronic medical evidence development: A case study of the Social Security Administration's MEGAHIT System," *Int. J. Med. Inform.*, vol. 80, no. 8, 2011.

[33] M. Liu, B. C. Wheeler, and J. L. Zhao, "On Assessment of Project Success in Community Source Development," *Proc. Twenty Ninth Int. Conf. Inf. Syst. (ICIS 2008)*, 2008.

[34] M. Liu, X. Wu, J. Leon Zhao, and L. Zhu, "Outsourcing of Community Source: Identifying Motivations and Benefits.," *J. Glob. Inf. Manag.*, vol. 18, no. 4, pp. 36–52, 2010.

[35] J. Nuottila, K. Aaltonen, and J. Kujala, "Challenges of adopting agile methods in a public organization," *International Journal of Information Systems and Project Management*, vol. 4, no. 3, pp. 65–85, 2016.

[36] R. Vidgen and K. Braa, "Balancing interpretation and intervention in information systems research: the action case approach," in *Information Systems and Qualitative Research*, 1997, pp. 524–541.

[37] N. King, "Template analysis. Qualitative methods and analysis in organizational research: A practical guide. ," in *Qualitative methods and analysis in organizational research: A practical guide*, 1998, pp. 118–134.

[38] A. Leon, *Software Configuration Management Handbook*. Boston, USA: Artech House, 2005.

[39] F. J. Buckley, *Implementing Configuration Managment, Hardware, Software and Firmware*. IEEE Standards Office, 1996.

[40] P. F. Manso and A. Nikas, "The application of post tender negotiation procedure: A public sector procurement perspective in UK," *International Journal of Information Systems and Project Management*, vol. 4, no. 2, pp. 23–39, 2016.

[41] I. Chengalur-Smith, A. Sidorova, and S. Daniel, "Sustainability of Free/Libre Open Source Projects: A Longitudinal Study," *J. Assoc. Inf. Syst.*, vol. 11, no. 11, 2010.

[42] M. Stuermer, G. Abu-Tayeh, and T. Myrach, "Digital sustainability: basic conditions for sustainable digital artifacts and their ecosystems," *Sustain. Sci.*, vol. 12, no. 2, pp. 247–262, 2017.

Biographical notes

Katja Henttonen
Ms. Katja Henttonen is working as a digitalization specialist at VTT Technical Research Centre of Finland. She has over 15 years of experience in software development gained in both the public and private sector. Since joining VTT, she has worked in various research projects around the following topics: open source systems, open innovation and collaborative economy. She holds a M.Sc. degree in ICTs and socio-economic development from the University of Manchester and is studying towards a Phd.

www.shortbio.org/katja.henttonen@vtt.fi

Jukka Kääriäinen
Dr. Jukka Kääriäinen works as a Senior Scientist at VTT Technical Research Centre of Finland Ltd in the Digital Transformation team. He has received PhD degree in 2011 in Information Processing Science from the University of Oulu. He has over 10 years of experience with product management, configuration management and lifecycle management. He has been involved in various European ITEA, ITEA2 and Artemis research projects.

www.shortbio.org/jukka.kaariainen@vtt.fi

Jani Kylmäaho
Mr. Jani Kylmäaho is currently employed at the National Land Survey of Finland, where his position is Head of Development for topographic data production. Jani worked as the product owner for the Oskari open source software until January 2017. He has been working with OGC services and both national and international SDIs for 15 years. He has extensive experience of open source software, agile methods, collaboration networks as well as INSPIRE implementation. Jani holds an MSc degree in Geography from the University of Helsinki, Finland.

www.shortbio.org/jani.kylmaaho@nls.fi

VI

HEALTH AND ORCHESTRATION OF PUBLIC-SECTOR OPEN-SOURCE SOFTWARE ECOSYSTEMS: ROLES, RULES, AND TOOLS

by

Katja Henttonen, Mirja Pulkkinen, & Pasi Tyrväinen, 2024

Accepted to the Scandinavian Journal of Information Systems (SJIS), expected to be published in December 2024

Request a copy from the author.