Coordinating Innovation: Evidence from Open Source Software Development

Dissertation
of the University St. Gallen,
Graduate School of Business Administration,
Economics, Law and Social Sciences (HSG)
to obtain the title of Doctor of Business Administration

submitted by

Petra Kugler
from
Germany

Approved on the application of

Prof. Dr. Georg F. von Krogh
and
Prof. Dr. Johannes Rüegg-Stürm

Dissertation no. 3094
Difo-Druck GmbH, Bamberg 2005
The University of St. Gallen, Graduate School of Business Administration, Economics, Law and Social Sciences (HSG) hereby consents to the printing of the present dissertation, without hereby expressing any opinion on the views herein expressed.

St. Gallen, June 30, 2005

The President:

Prof, Ernst Mohr, PhD
“Free software will take up the lead on innovation.”
(Oskar Sandberg, The FREENET Project, 2000)

“... you cannot build network organizations on electronic networks alone .... If so, ... we will probably need an entirely new sociology of organizations.”
(Nitin Nohria & Robert G. Eccles, 1992)

“With the artists and sculptors I know, there’s a sort of free play idea. You try things; you experiment. It’s kind of naïve and childish, it’s like kids in a playpen. Scientists work that way, too – for example genetic scientists that I have been involved with seem to work similarly. It’s kind of throwing things out and then following the ideas, rather than predicting where you’re going to go.”
(Frank O’Gehry, Architect)
# Table of Contents

**Introduction & Paper Abstracts**

**Part One**

*Coordination, Communities, Creative Innovation:*

Three Peculiarities of Open Source Software Development

1. Introduction 3
2. Literature Review: Organizational Coordination 6
   2.1. What is coordination and why is it essential to organizations? 6
   2.2. Market- versus firm coordination 8
   2.3. Formal firm coordination 9
   2.4. Informal firm coordination 15
   2.5. Firm coordination, organizational capabilities and new forms of organizing 19
   2.6. Research gaps 22
3. The Open Source Software Development Phenomenon 25
   3.1. What is open source software development? 25
   3.2. Open source software origin and licenses 26
   3.3. Influence on the global software- and hardware industries 29
4. Three Peculiarities of Coordination in Open Source Software Development 33
   4.1. Public good and the failure of traditional formal coordination mechanisms 33
   4.2. Reduced need for coordination in open source communities 41
   4.3. Simultaneous realization of stability and volatility: Dynamic capabilities 48
5. Conclusions 58

Endnotes Part 1 61
## Part Two

### Coordinating through Dominant Knowledge:
Evidence from the Freenet Open Source Software Development Project

<p>| 1.  | Introduction                      | 65 |
| 2.  | Literature Review; Open Source Software Development | 68 |
| 2.1. | External perspective             | 69 |
| 2.2. | Internal perspective             | 71 |
| 3.  | Methods                          | 80 |
| 3.1. | Exploratory single case study design | 80 |
| 3.2. | Data sources &amp; collection methods | 84 |
| 3.3. | Data analysis in the tradition of grounded theorizing | 88 |
| 4.  | Case Study Freenet; First Order Findings | 91 |
| 4.1. | Freenet project basics           | 91 |
| 4.2. | Coordination through shared, project-specific knowledge | 92 |
| 4.2.1. | Coordination despite of the lack of traditional mechanisms | 92 |
| 4.2.2. | Core developers and peripheral members | 94 |
| 4.2.3. | Coordination through taken for granted, shared knowledge | 96 |
| 4.2.4. | Three types of knowledge         | 97 |
| 4.2.5. | Project-specific knowledge versus project-related knowledge | 108 |
| 4.3. | Evolution of coordinating knowledge | 111 |
| 4.3.1. | Individual creation of coordinating knowledge | 111 |
| 4.3.2. | Capturing &amp; replication of coordinating knowledge | 118 |
| 4.3.3. | Transfer of coordinating knowledge | 121 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Case Study Freenet: Second Order Findings</td>
<td>129</td>
</tr>
<tr>
<td>5.1. Defining dominant knowledge as coordination mechanism</td>
<td>129</td>
</tr>
<tr>
<td>5.1.1. First condition: Three knowledge dimensions</td>
<td>130</td>
</tr>
<tr>
<td>5.1.2. Second condition: Organizational and repeatedly in use</td>
<td>132</td>
</tr>
<tr>
<td>5.1.3. Dominant knowledge definition</td>
<td>133</td>
</tr>
<tr>
<td>5.2. Evolution of dominant knowledge</td>
<td>135</td>
</tr>
<tr>
<td>5.2.1. Multi-step evolutionary process</td>
<td>135</td>
</tr>
<tr>
<td>5.2.2. First knowing advantage</td>
<td>138</td>
</tr>
<tr>
<td>6. What Dominant Knowledge is <em>not</em></td>
<td>140</td>
</tr>
<tr>
<td>7. Concluding Remarks</td>
<td>146</td>
</tr>
<tr>
<td>Endnotes Part 2</td>
<td>152</td>
</tr>
</tbody>
</table>
Part Three
Informally Hierarchical: Structuring Innovative Work through a Knowledge Hierarchy in Open Source Software Development, Model Generation and (partial) Testing

1. Introduction

2. Model Generation
2.1 Methods
2.1.1 Inductive single case study design and qualitative data
2.1.2 Case sampling: The Freenet project
2.1.3 Data types, data sources and collection methods
2.1.4 Data analysis in the tradition of grounded theorizing
2.2 First order analysis
2.2.1 Indicators of a hierarchy in the Freenet project
2.2.2 Bottom and top of the hierarchy
2.2.3 Knowledge hierarchy
2.2.4 Project dynamics and knowledge accumulation
2.2.5 Factors that influence the individuals’ project-specific knowledge
2.2.6 Entry, exit and mobility barriers
2.3 Second order analysis:
Knowledge hierarchy definition and model

3. Model Testing
3.1 Methods
3.1.1 Community sampling
3.1.2 Variable operationalization and data sources
3.1.3 Data analysis
3.2 Findings and discussion of quantitative findings

4. Discussion and Concluding Remarks

Endnotes Part 3
Dissertation Summary & Conclusions 253

References 259

List of Appendices 283

1  The open source definition 285
2  Interview partners: Name and date 288
3  Guidelines for interviews with Freenet core developers (first and second round) 290
4  First contact email to Freenet peripheral contributors 293
5  Open questions survey sent to Freenet peripheral members (third round) 294
6  Comparison of qualitative findings: Differences between Freenet core developers and peripheral members 297
7  Comparison of quantitative findings: Differences between Freenet core developers and peripheral members 299
8  First contact email to project maintainers: Questionnaire on a knowledge hierarchy 300
9  Web-based questionnaire introduction 302
10  Web-based questionnaire to OSS project members 304
List of Tables & Figures

**Part One**

| Table 1.1 | Characteristics of formal and informal coordination mechanisms | 10 |
| Table 1.2 | Property rights, principal-agent relationship and coordination in open and closed source software development | 37 |
| Figure 1.3 | The layered model of software development in firms and communities; Differentiation, interdependence and size | 45 |
| Table 1.4 | A comparison of core developers and peripheral project members in open source software development | 54 |
| Figure 1.5 | Stability and volatility through stable and changing membership of core developers and peripheral project contributors | 56 |

**Part Two**

<p>| Table 2.1 | Core developers, peripheral project members and coordination in Freenet; Case evidence | 95/96 |
| Figure 2.2 | Knowledge about behavioral community rules: Case evidence | 99/100 |
| Table 2.3 | Knowledge about technical standards; Case evidence | 103/104 |
| Table 2.4 | Knowledge about the overall project direction: Case evidence | 106/107 |
| Figure 2.5 | Freenet peripheral members’ entry to the project by means of submission of their first email to the developer mailing list | 112 |
| Figure 2.6 | Freenet core developers’ entry to the project by means of submission of their first email to the developer mailing list | 112 |
| Table 2.7 | Evolution of coordinating knowledge, first knowing advantage and individual knowledge creation; Case evidence | 116/117 |
| Table 2.8 | Individual knowledge capturing and replication: Case evidence | 120 |</p>
<table>
<thead>
<tr>
<th>Table</th>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>2.9</td>
<td>Knowledge transfer media: Case evidence</td>
<td>123</td>
</tr>
<tr>
<td>Table</td>
<td>2.10</td>
<td>Active and passive knowledge transfer behavior: Case evidence</td>
<td>126</td>
</tr>
<tr>
<td>Table</td>
<td>2.11</td>
<td>Evolution of coordinating knowledge: Case evidence</td>
<td>128</td>
</tr>
<tr>
<td>Figure</td>
<td>2.12</td>
<td>Three dimensions of dominant knowledge: Overall direction, technical standards, community rules and coordination at two different points of time, t and t+1</td>
<td>131</td>
</tr>
<tr>
<td>Figure</td>
<td>2.13</td>
<td>Evolution of dominant knowledge</td>
<td>136</td>
</tr>
</tbody>
</table>

**Part Three**

<table>
<thead>
<tr>
<th>Table</th>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>3.1</td>
<td>Inequality of ranks; Case evidence</td>
<td>171/172</td>
</tr>
<tr>
<td>Table</td>
<td>3.2</td>
<td>Inequality of decision rights; Case evidence</td>
<td>174</td>
</tr>
<tr>
<td>Table</td>
<td>3.3</td>
<td>Inequality of degrees of freedom to act; Case evidence</td>
<td>176</td>
</tr>
<tr>
<td>Table</td>
<td>3.4</td>
<td>Inequality of attention &amp; reputation; Case evidence</td>
<td>178</td>
</tr>
<tr>
<td>Table</td>
<td>3.5</td>
<td>Knowledge influences the hierarchy; Case evidence</td>
<td>182</td>
</tr>
<tr>
<td>Figure</td>
<td>3.6</td>
<td>Monthly increase of <em>emails</em> posted to Freenet’s developer mailing list and accumulated number of emails</td>
<td>184</td>
</tr>
<tr>
<td>Figure</td>
<td>3.7</td>
<td>Monthly increase of <em>code logs</em> posted to Freenet’s code repository and accumulated number of code logs</td>
<td>184</td>
</tr>
<tr>
<td>Table</td>
<td>3.8</td>
<td>Relation between individual knowledge &amp; project knowledge, project dynamics; Case evidence</td>
<td>186</td>
</tr>
<tr>
<td>Table</td>
<td>3.9</td>
<td>Knowledge absorption <em>before</em> project entry; Case evidence</td>
<td>189</td>
</tr>
<tr>
<td>Figure</td>
<td>3.10</td>
<td>Thread initiation and thread reply in Freenet, year 2000</td>
<td>193</td>
</tr>
<tr>
<td>Table</td>
<td>3.11</td>
<td>Knowledge absorption and creation <em>post</em> project entry; Case evidence</td>
<td>194</td>
</tr>
<tr>
<td>Table</td>
<td>3.12</td>
<td>Point of time of entry to the project and longevity on the project; Case evidence</td>
<td>196</td>
</tr>
<tr>
<td>Figure</td>
<td>3.13</td>
<td>Entry, exit and mobility barriers</td>
<td>197</td>
</tr>
<tr>
<td>Table</td>
<td>3.14</td>
<td>Entry barriers and new contributor selection; Case evidence</td>
<td>199</td>
</tr>
</tbody>
</table>
Table 3.15  Exit barriers; Case evidence  202
Table 3.16  Horizontal mobility barriers; Case evidence  203
Figure 3.17  Absolute number of Freenet contributors (core developers and peripheral members) who contributed at least once to the mailing list in month X in 2000  206
Table 3.18  Vertical mobility barriers; Case evidence  207
Figure 3.19  Knowledge hierarchy model  212
Table 3.20  Characteristics of the communities studied; CrystalSpace, HSQlDB, Stepmania, TikiWiki  217
Table 3.21  Overview of hypotheses – testing and description  224
Table 3.22  Knowledge absorption *before* project entry: Survey responses  226
Table 3.23  Knowledge absorption *post* project entry: Survey responses  229
Table 3.24  Other survey responses, descriptive statistics  231
Table 3.25  Discriminant function summary  237
Table 3.26  Discriminant analysis, goodness tests  238
Table 3.27  Discriminant analysis group statistics  239
Table 3.28  Discriminant analysis classification results  239
Figure 3.29  Stepmania’s accumulated number of CVS commits, emails, new contributors over time  241
Figure 3.30  TikiWiki’s accumulated number of CVS commits, emails, new contributors over time  241
Table 3.31  Monthly increases and accumulated numbers of Freenet’s emails and CVS code logs in 2000  248
Table 3.32  Correlations between the use of knowledge transfer media for knowledge absorption *before* project entry  250
Table 3.33  Correlations between the use of knowledge transfer media for knowledge absorption *post* project entry  251
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>Joint-stock company</td>
</tr>
<tr>
<td>AIM</td>
<td>AOL Instant Messenger</td>
</tr>
<tr>
<td>Arpanet</td>
<td>Advanced Research Project Agency Network</td>
</tr>
<tr>
<td>CEO, COO</td>
<td>Chief executive officer, chief operating officer</td>
</tr>
<tr>
<td>CMS</td>
<td>Content management system</td>
</tr>
<tr>
<td>CSS</td>
<td>Closed source software</td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent versioning system</td>
</tr>
<tr>
<td>dev.</td>
<td>Developer</td>
</tr>
<tr>
<td>Df.</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>e.g.</td>
<td>Exempli gratia, for example</td>
</tr>
<tr>
<td>ed.</td>
<td>Editor</td>
</tr>
<tr>
<td>et. al., etc.</td>
<td>Et alii, et cetera, et cetera</td>
</tr>
<tr>
<td>F</td>
<td>F-value, F-statistic</td>
</tr>
<tr>
<td>FSF</td>
<td>Free Software Foundation</td>
</tr>
<tr>
<td>GNU GPL</td>
<td>GNU General Public License</td>
</tr>
<tr>
<td>GUI</td>
<td>General user interface</td>
</tr>
<tr>
<td>IRC</td>
<td>Internet relay chat</td>
</tr>
<tr>
<td>LOC</td>
<td>Lines of code</td>
</tr>
<tr>
<td>M1, M2</td>
<td>Model 1, model 2</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>N</td>
<td>Basic population</td>
</tr>
<tr>
<td>no.</td>
<td>Number</td>
</tr>
<tr>
<td>o.V.</td>
<td>Author is unknown (‘ohne Verfasser’)</td>
</tr>
<tr>
<td>OSS</td>
<td>Open source software (development project/community)</td>
</tr>
<tr>
<td>p.</td>
<td>Page</td>
</tr>
<tr>
<td>periph.</td>
<td>Peripheral</td>
</tr>
<tr>
<td>sd.</td>
<td>Standard deviation</td>
</tr>
<tr>
<td>sig.</td>
<td>Significance</td>
</tr>
<tr>
<td>Vol.</td>
<td>Volume</td>
</tr>
</tbody>
</table>
Acknowledgements

Writing a dissertation is like going on a journey. It is, however, no convenient all-inclusive trip, but it is rather like an expedition for which no ready-made paths do already exist. Therefore, doing research is similar to Robert F. Scott’s journey to the South Pole, or to John Franklin’s travels in search of the Northwest Passage. Taking such an adventure opens up new horizons, which may lie beyond what we are already familiar with – this is the exciting part of working on a research project. But also, searching for new insights and for ways nobody went before is often inconvenient, since there may be some unforeseeable obstacles, or sometimes, there are even dead ends – that is the challenging part of writing a dissertation.

After the completion of such a trip most explorers would agree that they have not only discovered a small piece of the world and of how it functions, but that they have also learned about the process of going on an excursion and what it takes to successfully complete it, which is the difference between an expedition and the all-inclusive way of traveling. In terms of learning, we can get out more only if we are willing to invest more efforts, more time, more thoughts and more curiosity when taking the adventure.

On my way I have encountered many people from various nations who helped me to find the right direction during different stages of my work, and I am grateful to (in alphabetical order) Daniela Blettner, Sabine & Bernd Degner, Hans Georg Graf, Simon Grand, Didier Guillot, Stephan Herting, Eric von Hippel, Georg von Krogh, Helga & Helmut Kugler, Karim Lakhani, James Lincoln, Robert Ruttmann, Raquel Serrano, Sebastian Späth, Johannes Rüegg-Stürm, Philipp Türscher, Matthäus Urwyler. I am additionally grateful to the generous support of the Swiss National Science Foundation, which enabled me to spend some exciting months at the University of California, Berkeley. My special thanks go to all members, maintainers and administrators of the open source projects I have studied for my work, who donated their sparse time and their interests to the ongoing research.

St. Gallen, July 2005

Petra Kugler
Introduction
&
Paper Abstracts
Introduction & Paper Abstracts

Seventy years ago, Nobel Prize laureate Ronald Coase asked, “Why is there any organization?” (1937: 388) and initiated a discussion that influenced many of today’s research streams in organization and management theory. Twenty years later, James March and Herbert Simon started their analysis and discussion on organizations with the remark: “This book is about formal organizations” (1958: 1). Since then, researchers have widely agreed that organizations consist not only of a formal part, but are also grounded in some informal organization (e.g., Schein, 1992; Smircich, 1982; Weick & Roberts; 1993, Prahalad & Bettis, 1986).

In recent years, the situation of organizations has changed in many ways and global networks or the internet boom are just two examples. Widespread changes brought about new forms of organizing that transcend our traditional understanding of what constitutes the kind of firm to which Coase or March & Simon have referred: New forms of organizing are networked, they are globally dispersed or they are virtual. But the new situation goes one step further, since it brought about organizations that have no more formal organization at all, such as open source software development projects.

These communities have no formal structure or hierarchy, they are entirely grounded in an informal organization. A surprisingly large number of open source communities are not only strikingly successful (e.g., Linux) and innovative (e.g., Freenet) but also challenge our established ways of thinking, our traditional theoretical categories, our taken-for-granted ‘rights’ and ‘wrongs.’ For traditional organization theory, open source communities are therefore a widely unexplained phenomenon. These considerations are the starting point for this dissertation.

My work takes a closer look at this new form of organizing, which is successful and innovative, and has no formal organization. I focus on open source communities and ask ‘How do these organizations function?’ ‘Why do they function?’ and ‘What is so peculiar about these organizations?’ So, this work is about informal organizations. I was able to find some exciting and new aspects on this special type of organization that will be presented in the body of this work. At the same time, every answer we find opens new questions,
which is also the case for this study. Many aspects of new organizational forms that are interesting or burning must therefore remain open for future work and the most intriguing question is: *Why is there any formal organization?*

<table>
<thead>
<tr>
<th></th>
<th><strong>Part 1</strong></th>
<th><strong>Part 2</strong></th>
<th><strong>Part 3</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Research methods</strong></td>
<td>Theoretical, deductive.</td>
<td>Inductive, single case study.</td>
<td>Inductive, single case study. (Partial) model testing.</td>
</tr>
<tr>
<td><strong>Sources of evidence</strong></td>
<td>Anecdotal evidence,</td>
<td>Qualitative (interviews &amp; open questions survey), Quantitative (archival project data),</td>
<td>Qualitative (interviews &amp; open questions survey), Quantitative (web-based survey, archival data),</td>
</tr>
<tr>
<td><strong>Contents</strong></td>
<td>Literature review, peculiarities of (coordination in) open source communities,</td>
<td>Dominant knowledge as informal coordination mechanism in OSS,</td>
<td>Knowledge hierarchy as informal structuring mechanism in OSS,</td>
</tr>
</tbody>
</table>

Comparison of dissertation parts.

The dissertation is structured in three complementary parts, which theoretically could serve as separate papers. Practically, however, they don’t, since the dissertation’s parts are both too extensive and too short. The parts are *too extensive* in order to serve as papers, as they go far more into detail than any (published) article would allow. They are, at the same time, *too short* to serve as separate papers, since I have tried to avoid overlaps between the separate parts wherever possible (it was *not* always possible). Some paragraphs of this work, such as literature reviews or a characterization of the open source phenomenon, are written in blocks although the information they give has some relevance for each of the three dissertation parts. In addition to this, the three parts of this work are complementary with respect to the *research methods* employed, the *sources of evidence* and with regard to the *papers’ content*, as the subsequent abstracts will illustrate.
Abstract Part One

Coordination, Communities, Creative Innovation: Three Peculiarities of Open Source Software Development

How organizations coordinate their collective efforts is of central concern to both organization theory (how do they achieve coordination?) and strategic management (how well do they do so?). Traditional coordination mechanisms refer to organizations’ formal structure and hierarchy and they correspond to informal instruments, such as organizational culture, cognition or routines. Both categories of coordination mechanisms, formal and informal, suffer from a trade-off between, on the one hand, coordination & stability, and on the other hand, innovation & volatility. In a Schumpeterian (1934 & 1942) world of rapid changes both these characteristics are essential and correspond with dynamic capabilities (Teece et al., 1997). Therefore, the crucial question is, ‘How can we create organizations that are stable enough to function as an entity, and that are, at the same time, open enough to allow for continuous change?’

Open source software development communities are one new form of organizing, which seems to be capable of achieving both coordination and innovation. Well-known empirical examples demonstrate this (e.g., LINUX, PERL or APACHE). Prior literature on this new but rapidly growing phenomenon has not yet been able to clarify what reasons and peculiarities lie behind the nature of open source communities as a promising mode of organizing innovative work. Against this background, this work is grounded in deductive theoretical reflections to reach the following research aims: Firstly, the paper reviews prior literature on coordination in organizations and gives an overview of major formal and informal coordination mechanisms. Secondly, the paper introduces the open source phenomenon, and finally, it aims at discussing and proposing three peculiarities of coordination in the open source setting.

All findings are theoretical or are based on anecdotal evidence, I conclude, firstly, that formal coordination mechanisms are bound to fail in open source software communities, since these organizations’ output is a public good instead of a private good. Secondly, my findings suggest that the total amount of organizational coordination necessary to coherently generate some organizational output is lower in average open source projects than in average software firms. In open source communities, the
contributors focus exclusively on explorative work, whereas supportive and administrative activities play no crucial role. This should decrease their requirements of coordinative efforts. Thirdly, while firms rely on a more or less stable workforce, the situation is different for open source projects. An average open source project combines a rather stable body of contributors (core developers) with a second body of peripheral members who hold a fluid membership in the communities. Taken together, the members of these two contributor groups simultaneously provide for stability and change in the projects of which they are members. These two factors together constitute the characteristics of dynamic capabilities.

Abstract Part Two

Coordinating through Dominant Knowledge: Evidence from the Freenet
Open Source Software Development Project

Coordination is a key topic in management and organization theory. At this stage, only little is known about how coordination is achieved in new forms of organizing beyond the market-firm continuum, such as it is the case for communities or networks. This paper employs inductive, qualitative & quantitative case study research to investigate coordination in an innovative open source software development project. This project cannot rely on traditional formal coordination mechanisms. My findings indicate that informal project-specific knowledge of rules and standards that is shared by the project members plays a crucial role as a coordination instrument. Project-related knowledge, often referred to as ‘hacker culture,’ plays only a minor role in this process.

This concept of knowledge with a coordinating function has to fulfill certain conditions: It must cover each of three distinctive dimensions that span the field of activity in which the project is situated and in which it evolves over time. These three dimensions include knowledge of the project’s overall direction, knowledge of technical standards and knowledge of informal community rules. For coordinative purposes, knowledge must be organizational (shared by at least two individuals) and repetitively in use. I will introduce the construct ‘dominant knowledge’ to describe knowledge that is able to coordinate an organization. The roots of dominant knowledge are grounded in the
early stages of a project’s inception, and it goes through a four-step evolutionary process to become dominant, thereby coordinating a project’s contributors. The knowledge transcends two boundaries, namely from the individual to the organizational level of analysis, and from a single-time to the repeated use of knowledge, Key mechanisms in this process that also may lead to an overall dominant knowledge base are knowledge creation, capturing & replication and knowledge transfer. Early project members are additionally found to profit from a ‘first knowing advantage’.

Abstract Part Three
Informally Hierarchical: Structuring Innovative Work through a Knowledge Hierarchy in Open Source Software Development. Model Generation and (partial) Testing

Traditional organization theory stresses formal organizational structures and an explicit hierarchy as coordination mechanisms in organizations on the market–firm continuum. Such mechanisms trace back to Weber’s idea of a rational bureaucracy (Weber, 1978), having been termed ‘mechanistic’ as opposed to ‘organic’ organizations (Burns & Stalker, 1961). Researchers find that such organizations are appropriate for stable markets, but that they are not suitable under volatile conditions demanding continuous change or innovation. New forms of organizing move away from the traditional hierarchical model of bureaucracy, such as in the case of open source software development communities.

Against this background, this paper aims at answering the following research questions; First, does a hierarchical principle exist in (seemingly democratic) open source software development communities? If so, what are the differences between hierarchies that are based on the authority given explicitly as we know it from a company setting, and a hierarchy as we can find it in open source communities? Secondly, in what is a hierarchy in open source software projects grounded, and what influences the hierarchical position of individuals in this setting?
To answer these questions, this study employs a two-step analytical and empirical process, which generates and (partly) tests a model of an informal hierarchy in open source communities. In a first step, qualitative single case data on the Freenet open source project serve to inductively generate hypotheses and a model on a knowledge hierarchy. In a second step, quantitative data, which were generated by means of an online survey or which originate in publicly accessible project data archives, served to test the findings that were generated in the first step. The second step is based on data on four open source projects, CrystalSpace, TikiWiki, HSQLDB, and Stepmania.

My findings suggest that although open source software development communities appear to be entirely democratic at first sight, they are only limitedly so. In the communities studied, a hierarchy prevails different from those we know from bureaucratic organizations. In open source communities, the prevailing hierarchy is entirely informal, emergent and it is grounded in the contributors’ project-specific knowledge. I will refer to this type of hierarchy as a ‘knowledge hierarchy’. My research reveals that an individual’s position in the knowledge hierarchy is determined by a variety of factors; in general, the earlier an individual enters a community, the more project-specific knowledge he or she is able to absorb, the higher this person’s position in the knowledge hierarchy. Knowledge was additionally found to create mobility, entry and exit barriers for the contributors to a certain project.
Part One

Coordination, Communities, Creative Innovation:
Three Peculiarities of Open Source Software Development
Coordination, Communities, Creative Innovation:  
Three Peculiarities of Open Source Software Development

1. Introduction

How organizations coordinate their collective efforts is of central concern to both organization theory (how do they achieve coordination?) and strategic management (how well do they do so?). Prior literature primarily discusses coordination mechanisms we are familiar with from organizations on the market-firm continuum. Authors refer to formal organizational structure and to a formal hierarchy or authority (e.g., Pugh et al., 1968; Child, 1973; Van de Ven et al., 1976), as well as to informal organizational culture, cognition, or routines (e.g., Schein, 1992; Bettis & Prahalad, 1994; Nelson & Winter, 1982).

Both categories of coordination mechanisms, formal and informal, suffer from a trade-off between coordination & stability, on the one hand, and innovation & volatility, on the other. In a Schumpeterian (1934 & 1942) world of rapid changes, both sides are essential, and they are a part of dynamic capabilities (Teece et al., 1997). The crucial question is therefore ‘how can we create organizations that are stable enough in order to function as an entity, while they are, at the same time, open enough to allow for continuous change?’

Only recently do authors refer to new forms of organizing, which move beyond the established notion of organizations on the market-firm continuum, and which have characteristics that seem to be more suitable in order to cope with today’s challenges. One new organizational form, which has only recently been able to attract considerable interest are communities such as open source software development projects. In open source communities, often strikingly innovative and successful software is created by voluntary, freely contributing developers. These communities seem capable of achieving both coordination and innovation, as well-known empirical examples demonstrate (e.g. Linux, Perl or Apache). Open source communities are independent from commercial players, and their contributors collaborate exclusively over the internet on a virtual platform. Prior literature on this new but rapidly growing phenomenon has not yet been
able to clarify what reasons and peculiarities lie behind open source communities’ nature as a promising mode for organizing innovative work. The reasons why coordination in this form of organizing may be different from what we already know from firms remains an unresolved topic.

Against this background, the objective of this study is threefold: Firstly, the study aims at giving an overview of previously discussed coordination mechanisms in organizations both formal and informal. Secondly, the study will give some background knowledge about the open source phenomenon. These communities are still young, but they have a significant influence on the global software- and hardware industries. Finally, the paper discusses three characteristics of open source communities which are different from those of organizations on the market–firm continuum and which illustrate that communities are an interesting new model for organizing innovative work.

More concretely, this work seeks to answer the following research question: What are the peculiarities of coordination in open source software projects, and why are these communities an interesting mode for organizing innovation? I will ground my reflections in various theoretical streams of literature and refer to organization theory, social network theory, and economic theory. In addition to the theoretical reflections, I will present some anecdotal evidence on software firms and on open source projects\(^1\).

Briefly summarizing, I conclude, firstly, that formal coordination mechanisms are bound to fail in open source software communities since their output is a public instead of a private good. Informal coordination mechanisms seem more suitable for coordinating in this setting. Secondly, my findings suggest that the total amount of organizational coordination that is necessary to produce a coherent product is smaller in an average open source project than in an average software firm. In open source communities, the contributors focus exclusively on explorative work, whereas supportive and administrative activities play no crucial role, which should decrease their need for coordinative efforts. Thirdly, while firms rely on a more or less stable workforce, the situation is different for open source projects. An average open source project combines a rather stable body of members (core developers) with a second body of peripheral members who hold a fluid membership in the communities. Taken together, both
contributor groups simultaneously provide for stability and change, which, at the same time, characterizes dynamic capabilities (Teece et al., 1997).

The remainder of this paper is structured as follows: In the second chapter, I will give an overview of prior work on coordination in organizations and discuss major mechanisms for achieving unity of effort, formal and informal. Next, I will provide an introduction to the open source phenomenon and give some background information, which is necessary in order to fully understand this organizational form. In chapter four, I will discuss three peculiarities of the open source mode of organizing and demonstrate its suitability for the generation of innovative output. Finally, I will briefly conclude with some promising avenues for future research.
2. Literature Review: Organizational Coordination

I will start with an overview of prior work on coordination in organizations, and refer to both formal and informal coordination mechanisms. Additionally, I will briefly discuss the role of coordination as an organizational capability, and show that new forms of organizing have characteristics which are different from the traditional firm model.

2.1. What is coordination and why is it essential to organizations?

Organizational coordination is defined as the “process of achieving unity of effort among the various subsystems in the accomplishment of the organizational task” (Lawrence & Lorsch, 1967a: 4). Various authors synonymously refer to the identical mechanism as ‘coordination’ (e.g. March & Simon, 1958; Mintzberg, 1979; Van de Ven et al., 1976; Martinez & Jarillo, 1989), or ‘integration’ (e.g. Galbraith, 1973; Lawrence & Lorsch, 1967a). I will also use both terms synonymously.

Coordination is necessary since organizations are coping with complex tasks and goals which go beyond the capacity of a single individual. The overall organizational task becomes split up into a number of smaller sub-tasks and sub-goals that can be accomplished by individuals or by working groups that operate more or less in isolation. By means of differentiation, though, all sub-tasks and sub-goals are treated as if they were independent from each other, but in fact, this is only rarely the case. In order to advance the organizational goal, all sub-tasks and sub-goals must therefore be reintegrated and aligned through coordination. The more an organization is capable of coordinating its activities, the more efficient it will be. In essence, coordination aims at managing interdependencies of various kinds (March & Simon, 1958; Mintzberg, 1979; Barnard, 1938; Lawrence & Lorsch, 1967a; Crowston & Kammerer, 1998).

Authors agree that coordination is a most central organizational task, if not an organization’s very essence (e.g. Aoki, 1990; Chandler, 1990; Starbuck, 1992; Zander & Kogut, 1995; Grant, 1996). The underlying reason therefore is that coordination justifies an overall, all-encompassing organization instead of a loose collection of individuals or of independent resources (Alchian & Demsetz, 1972; Coase, 1937). How and how well
organizations achieve unity of effort is therefore a critical question for both organization theory (how?) and for strategic management (how well?).

Empirical work supports this line of thought, and a number of studies have been able to find a positive relationship between an organization’s coordination (capabilities) and its financial performance in a diversity of industries, such as scientific software development (Hoopes & Postrel, 1999), electronics (Helfat & Raubitschek, 2000), the automobile industry (Clark & Fujimoto, 1991), pharmaceuticals (Henderson & Cockburn, 1994), the semiconductor photolithographic industry (Henderson & Clark, 1990), or the chemical and optical industry (Leonard-Barton et al., 1994).

Lawrence & Lorsch (1967a) found additionally that high-performing organizations are both more strongly differentiated and more integrated than lower performing organizations. Specialization is therefore purposeful, provided that the organization can also achieve reintegration of its specialized efforts, Lawrence & Lorsch conclude that coordination is more important for organizational performance than differentiation.

In order to achieve coordination in organizations, a number of tools are available. One type of coordination tool encompasses the organization’s institutional or governance form. In analyzing and discussing various governance forms, prior literature has so far primarily focused on organizations on the market–firm continuum. Only recently have new forms of organizing, which go beyond the traditional understanding of what constitutes an organization, been able to attract the researchers’ attention. A second and a third type of coordination tools are an organization’s internal formal and informal coordination instruments, which provide for a structure to the organizations’ differentiation and to their realignment of tasks, I will briefly discuss each of the three types of coordination tools in the following and start with organizations’ governance forms.
2.2. Market- versus firm coordination

From an economic perspective, markets and firms are the two basic institutional forms for achieving coordination of exchanges between individuals and across organizational boundaries. Under perfect (market) conditions, every individual actor has access to all the information that is necessary for meeting the most efficient exchange decisions. Given this situation, markets range as the first-best solution since all activities can exclusively be coordinated over the price mechanism. Under perfect market conditions, prices are therefore capable of capturing all necessary information for efficient market transactions.

Under imperfect market conditions, however, actors in the market do not have access to the entire information necessary for making their decisions best, which is, for instance, the case for intangible goods or in complex, idiosyncratic exchange situations. Prices are a simplifying mechanism, which also fail whenever no market or no price exists. In such situations, seemingly rational actors will behave boundedly rationally (Simon, 1976) or opportunistically (Alchian & Demsetz, 1972) as a consequence of their information asymmetries (Powell, 1990). In order to overcome their information asymmetries, the actors will make use of a number of mechanisms and specify more detailed contracts or employ some monitoring behavior. However, such mechanisms create transaction costs such as the “costs for using the price mechanism” (Coase, 1937: 390-391), and (market) coordination is subsequently not a costless task2.

Under uncertain (market) conditions the transaction costs will be lower (but still not equal to zero) if an exchange is carried out within the boundaries of a firm instead of using the market mechanism. In firms, the price mechanism becomes suppressed and coordination is achieved as a result of the instructions and initiatives of an entrepreneur who holds centralized property rights to the produced good (Grossman & Hart, 1986; Hart, 1988; Hart & Moore, 1988; Coase, 1937; Williamson, 1975 & 1985). Firms are seen as a second-best solution besides the pure market mechanism. Since firms can be conceptualized as a nexus of contracts (similar to a market), firms and markets constitute the two extremes of a continuum of organizational types that contain a smaller or larger fraction of elements of both regimes (Jensen & Meckling, 1976).
As a general rule, *firms* are a more adequate and more efficient governance form for coordinating those transactions that involve great uncertainty about their outcome, which recur frequently, and which require substantial transaction-specific investments of time, money or energy (Williamson, 1975). Moreover, firms are suitable when the existing incongruence of goals between the contributing parties is moderately high (Ouchi, 1980), which is, for instance, the case for services as opposed to production work, *Markets*, in contrast, which cover flexible and short-term exchange relationships, are most efficient for the production of standardized goods with little performance uncertainty and with the involved parties’ high levels of opportunism tolerance (Ouchi, 1980). In the subsequent paragraphs, I will take a closer look at the formal and informal coordination that takes place within organizations.

### 2.3. Formal firm coordination

Prior literature differentiates between two grand types of firm coordination, *formal* and *informal* mechanisms. Formal firm coordination is defined as “*that kind of coordination among men [...] that is conscious, deliberate, purposeful*” (Barnard, 1938: 4), and it refers to an organization’s formal structure and formal hierarchy, both tracing back to Weber’s idea of a rational bureaucracy (Weber, 1978). The underlying assumption is that organizational systems can compensate for humans’ cognitive limitations, which result from their bounded rationality (Simon, 1976), and which indirectly guide and control their behavior.

Mechanisms of formal firm coordination aim at making the individuals’ behavior in the organization more foreseeable in the direction that is desired by the firm in order to eliminate behavioral uncertainties (Barnard, 1938; March & Simon, 1958; Thompson, 1967). Prior literature (mainly on contingency theory or the Aston studies) suggests a number of formal, structural coordination mechanisms, and the most important mechanisms are: (1) Centralization & decentralization, (2) standardization & formalization, (3) departmentalization & configuration, (4) goal setting & planning, (5) output & behavioral control. I will give a brief overview of these formal coordination mechanisms in the following paragraphs (for an overview see table 1.1, or: Martinez & Jarillo, 1989, Pugh et al., 1968; Kieser & Kubicek, 1992).
<table>
<thead>
<tr>
<th>No.</th>
<th>Coordination mechanism</th>
<th>Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Centralization (&amp; decentralization)</td>
<td>Concentration of decision rights at the top management (or distribution across lower levels).</td>
</tr>
<tr>
<td>2</td>
<td>Standardization &amp; formalization</td>
<td>The extent to which policies, rules, job descriptions etc. are written down in manuals or in other documents.</td>
</tr>
<tr>
<td>3</td>
<td>Departmentalization &amp; configuration</td>
<td>The formal grouping of activities within organizational units, labor division.</td>
</tr>
<tr>
<td>4</td>
<td>Goal setting &amp; planning</td>
<td>Systems that intend to guide and channel the activities and actions of independent units.</td>
</tr>
<tr>
<td>5</td>
<td>Output &amp; behavioral control</td>
<td>Bureaucratic control and feedback through files, reports, records.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Informal coordination mechanisms</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6 Organizational culture</td>
<td>Shared values, beliefs, rituals &amp; symbols,</td>
</tr>
<tr>
<td>7 Cognitive approaches</td>
<td>Individuals’ identical interpretation and mental models,</td>
</tr>
<tr>
<td>8 Routines</td>
<td>Complex, relatively automatic patterns of behavior, triggered by a small number of initiating signals,</td>
</tr>
</tbody>
</table>

Table 1.1: Characteristics of formal and informal coordination mechanisms.

**Centralization & decentralization**, In a hierarchical organization, individuals have different rights to perform control and supervision, goal and rule setting, decision-making or power, and they vary with respect to their access to critical information. Those individuals who are at the top have the most and the most important rights, since they are also supposed to have the best overview of their organization, and they should, therefore, be capable of making the best decisions (Galbraith, 1973; Mintzberg, 1979; Nadler & Tushman, 1988). Centralization and decentralization determine whether these rights lie in the higher or lower levels of the chain of command, and to how many people the rights are distributed.

In a *centralized organization*, all rights are concentrated with the top executives, while for a *decentralized organization* the opposite is true. In a centralized organization, the underlying logic is that the organization’s most capable and most experienced individuals specify ends, develop goals and route or filter information. Time-consuming discussions with their subordinates, who execute their commands, are not necessary, and
the subordinates should, at the same time, not question the decisions that were made at the top of the organization (Pugh et al., 1968; Simon, 1976; Lawrence & Lorsch, 1967a&b; Galbraith, 1973; Child, 1972a). Centralization is appropriate only if the people at the top of the hierarchy possess all the required information that is necessary for making their decisions.

**Standardization & formalization.** Standardization & formalization refer to “the extent to which policies, rules, job descriptions, etc. are written down in manuals and other documents, and procedures are established through standard routines” (Martinez & Jarillo, 1989: 491). The mechanism has received a variety of names including ‘standard practices’ (Simon, 1976), ‘paper system’ (Lawrence & Lorsch, 1967b), ‘formalization’ (Pugh et al., 1969; Child, 1973; Adler & Borys, 1996), ‘standardization’ (Thompson, 1967; March & Simon, 1958; Cyert & March, 1963), or (formal) ‘rules’ (Galbraith, 1973). Standardization can refer to the works input, output, or to working processes (Mintzberg, 1979).

The underlying logic is that written devices can reduce and contain all the information that is necessary in order to fulfill a certain task. Standardization aims at creating stable behavioral patterns across all members of an organization. Since the organizational members must not make time-intense case-by-case decisions whenever they occur, they can refer to what has already been decided and so act in a highly efficient way (Galbraith, 1973). Standardization and formalization anticipate what will happen in an organization. The mechanism can only effectively coordinate if an organization’s members are capable of fully anticipating the organization’s current and future situation, which is the case only for organizations under stable (internal or external) conditions.

**Departmentalization & configuration.** The third mechanism aims at achieving a “grouping of activities within organizational units, following the principle of labor division as a mechanism of organizational influence” (Martinez & Jarillo, 1989: 491). It describes the formal shape of the organization’s role structure (Pugh et al., 1968), which becomes visible through an organigram. An organization’s configuration provides formal information concerning the staff’s and line’s responsibilities or authorities. It includes the organization’s vertical span (number of formal hierarchical levels), as well as its
horizontal span of control (average number of direct subordinates) (Pugh et al., 1968; Kieser & Kubicek, 1992).

*Departmentalization* refers to how tasks that are similar or different are clustered in separate organizational divisions. It may be one-dimensional or multi-dimensional. Major forms of departmentalization include the *functional* organization (structured along functional areas, e.g. marketing, sales, R&D; e.g. Mintzberg, 1979); the *regional* organization (structured along markets, e.g. Asia, Europe, North America; e.g. Stopford & Wells, 1972; Lawrence & Lorsch, 1967b); the *multidivisional* organization (structured along product groups, e.g. print media, electronic media; e.g. Chandler, 1962); or the *matrix* organization (two or more of these dimensions are in use simultaneously; e.g. Galbraith, 1973). All of these structures aim at grouping workflow-interdependencies or process-interdependencies and they demand a clear hierarchical decision-making process in the organization.³

**Goal setting & planning.** The fourth mechanism refers to “*systems and processes […] that intend to guide and channel the activities and actions of independent units*” (Martínez & Jarillo, 1989: 491). It includes systems like strategic planning, budgeting, scheduling (March & Simon, 1958; Thompson, 1967) or goal setting (Galbraith, 1973). Coordination is achieved through specifications that refer to a certain period of time. Although plans and goals may be subject to frequent change, they anticipate future events and the more concrete the plans and goals are, the more they will be grounded in the assumption that an organization’s internal or external situation is stable and certain (Kieser & Kubicek, 1992).

**Output & behavioral control.** The fifth mechanism aims at coordinating organizational members’ activities by means of control and feedback. Coordination is grounded, *firstly*, in an impersonal form of output control, namely on the evaluation of files, reports or records that are submitted to corporate management, and authors refer to it as ‘bureaucratic control’ (Child, 1972a & 1973), ‘performance control’ (Mintzberg, 1979) or ‘output control’ (Martínez & Jarillo, 1989). *Secondly*, coordination is also grounded in direct, personal surveillance or on personal control (Mintzberg, 1979; Martínez & Jarillo, 1989), since people who occupy superior positions in the organization directly monitor and comment on their subordinates’ behavior. In general, this mechanism is more flexible
than anticipatory coordination mechanisms, but it assumes similarly that authority and knowledge are concentrated with the organizations’ executives.

Against the background of a broad range of formal coordination mechanisms the fundamental question of organization theory is ‘which coordination mechanism or organizational structure is most suitable and most successful for which kind of organization?’ Prior work in the tradition of contingency theory suggests that there is not just one best way for coordinating an organization. An organization’s appropriate structure depends, instead, on its specific context and the better the ‘fit’ between structure and context, the more efficiently the organization will perform (Galbraith, 1973). In this research tradition, contextual variables are associated with an organization’s environment (Burns & Stalker, 1961; Lawrence & Lorsch, 1967a&b), organizational size (Child, 1972a; Blau & Schoenherr, 1971; Pugh et al., 1969), production technology (Woodward, 1965), or organizational strategy (Chandler, 1962; Child, 1972b). Major findings indicate the following,

firstly, in stable and in rapidly changing markets, different forms of organizing are efficient. In stable markets, the best-suited form of coordination is bureaucratic, or ‘mechanistic’. It is characterized by a clear hierarchy, autocratic leadership, high degrees of specialization, standardization and formalization, and by highly centralized decision-making processes. In such a setting, decision-making refers to expectations that are well foreseeable. Dynamic markets with rapidly changing technologies instead, demand an ‘organic’ organizational structure that is characterized by flat or nebulous hierarchies, a democratic leadership style, a low degree of formalization and standardization and by a decentralized decision-making style, all of which enable the firm to make use of its members’ knowledge and experience, which are distributed across the organization (Burns & Stalker, 1961).

Secondly, organizational subsystems are confronted with different types of contexts. R&D departments, for instance, are located in a highly uncertain context that is characterized by unclear causal relationships and by ambiguous information, whereas for firms’ production departments the opposite is true. To the extent that organizational subtasks vary in their predictability, different structures (and different cognitive and emotional orientations) should be employed.
The appropriate coordination mechanism depends, therefore, on an organization’s degree of differentiation, with more strongly differentiated organizations demanding complex (personal, but still formal) forms of organizing, and less differentiated organizations requiring simpler coordination mechanisms, such as hierarchy or formalization (Lawrence & Lorsch, 1967 a&b; Hall, 1962). With increasing task uncertainty, however, personal (yet formal) coordination mechanisms, such as formal meetings or committees, substitute for impersonal instruments (e.g. plans or manuals, Van de Ven et al., 1976).

Finally, researchers found that large organizations as opposed to smaller firms are characterized by a greater need for coordination and formal organization structure. More concretely, organizational growth leads to more specialization, standardization and formalization and to the decentralization of decisions (Child, 1972a; Blau & Schoenherr, 1971; Pugh et al., 1969). The tendency to increasingly use formal and impersonal coordination mechanisms with growing organizational size leads, therefore, to an ‘impersonalizing effect’ (Van de Ven et al., 1976). These findings indicate that larger organizations are consequently less capable of adequately coping with uncertain, rapidly changing environments, since they primarily employ simple, impersonal, and formal coordination mechanisms. Such organizations have difficulties in responding to new situations that they have not faced before, and to which they have no ready-made responses.

Newer approaches to formal coordination instruments find that not only the degree of formalization is decisive, but also the type of formalization, and it can be either coercive or enabling (Adler & Borys, 1996). While coercive formalization substitutes rather than complements human commitment and thoughtfulness in organizations, enabling formalization is more open. It represents an organization’s memory, which captures lessons learned from prior experience, and which so codifies best practices (Walsh & Ungson, 1991; Levitt & March, 1988). While coercive formalization refers to what in the preceding analysis was described as formal coordination, enabling bureaucracy is rather informal and will be discussed in the following paragraphs.
24. Informal firm coordination

Prevailing in most organizations one finds not simply formal structures. Firms are additionally subject to informal coordination mechanisms that are more subtle, irrational and often implicit, since informal coordination instruments are effective beyond what is obvious and visible in organizations. Formal and informal coordination mechanisms have been found to serve as substitutes (Wilkens & Ouchi, 1983; Peters & Waterman, 1983), as counter-productive mechanisms (Wilkens & Ouchi, 1983), or as complements (Williamson, 1990; Barnard, 1938; Rüegg-Stürm & Gritsch, 2001; Rüegg-Stürm, 2003; Martinez & Jarillo, 1983; Poppo & Zenger, 2002).

In the latter line of thought, informal coordination has existed to complement formal organization in three significant ways: “One of the indispensable functions of informal organization in formal organizations […] is that of communication, […] Another function is that of maintaining cohesiveness in formal organizations through regulating the willingness to serve and the stability of objective authority. A third function is the maintenance of the feeling of personal integrity, of self-respect, and independent choice” (Barnard, 1938: 122).

Although prior literature has discussed a broad variety of partly overlapping informal coordination instruments, researchers have agreed on a set of common characteristics: Firstly, informal coordination mechanisms achieve unity of effort within an organization by means of a communal understanding that is shared between the organizational members. The informal coordination instruments differ with respect to how the members’ shared understanding is achieved, and prior literature has proposed the following instruments: Shared values, symbols & rituals (organizational culture: e.g. Schein, 1992; Smircich, 1983; Peters & Waterman, 1983; Rüegg-Stürm & Gritsch, 2001), identical patterns of interpretation and shared mental models (cognitive approaches: e.g. Bettis & Prahalad, 1994; Weick & Roberts, 1993; von Krogh & Roos, 1996) or overlapping, inter-linked and repeated activities and work processes (routines: e.g. Nelson & Winter, 1982; March & Simon, 1958; Pentland & Rueter, 1994).

The organizational members’ communal understanding of their world traces back to different types of shared organizational knowledge within the organization (Frost, 1997). For the purpose of this work, and along with Nonaka, I will refer to knowledge as
“justified true belief” (Nonaka, 1994: 15). Over time, organizational knowledge that is shared accumulates to an organizational memory (Walsh & Ungson, 1991) or to an organizational code (March, 1991) and so stores the individuals’ shared understandings.

Secondly, contrary to formal, structural coordination instruments, informal coordination mechanisms provide no clear answers to what needs to be done in a specific situation (as it is the case for plans or for programs). Informal coordination mechanisms rather define a (yet closed) framework within which the organizational members are supposed to act. In doing so, informal coordination mechanisms are able to provide the organizational members with some orientation and direction, and at the same time, these mechanisms serve as organizational control instrument since the members’ behavior is more predictive (Frost, 1997). The underlying basic idea is that the more an organization’s members share implicit, internalized understandings, the more alike they will react in newly occurring situations, and the less time-consuming case-by-case decisions are required. Informal coordination is consequently seen as an adequate instrument for achieving coordination in complex and uncertain settings, in which formal mechanisms fail, as for instance, in the case of R&D activities (e.g. Ouchi, 1980).

Thirdly, informal coordination mechanisms have a deep and embedded character, since values, cognitive structures and human behavior are relatively stable over time. They cannot simply be imposed on organizational members, but these mechanisms need both considerable time-investments and learning efforts to evolve and grow. Such processes require individual and collective efforts, since informal mechanisms must be shared and not simply be individual to coordinate an organization. Informal coordination instruments are consequently not costless, but create transaction- as well as social costs, similar to formal coordination instruments (Ouchi, 1980). I will give a brief overview of informal coordination mechanisms in the following and refer to (1) organizational culture, (2) cognitive approaches and, (3) routines.

Organizational culture. The idea of a shared organizational culture as informal coordination mechanism is grounded in the organizational members’ shared values, beliefs and rituals and on their shared understanding of symbols, all of which are deep and stable over time. Since the shared values, beliefs and rituals hold an organization
together, an organization’s culture gives its members enough guidance on how to behave in their organization and it can serve as a normative clue to provide some structural stability across functions and departments (Smircich, 1983; Schein, 1992; Peters & Waterman, 1983; Rüegg-Stürm & Gritsch, 2001). Fundamental values and beliefs are manifested in and become transferred through symbolic devices, such as myths, rituals, stories, legends, or through a specialized language. Management is, from this perspective, a symbolic activity (Smircich, 1983; Pfeffer, 1981).

As long as an organization’s members identify with the organization’s goals, their joint efforts will lead to both the fulfillment of individual and organizational interests (Wilkins & Ouchi, 1983; Ouchi, 1980; Rüegg-Stürm & Gritsch, 2001). An organizational culture fulfills several distinct functions that hold an organization's activities together: 
Firstly, it provides for a sense of collective identity, since individuals are committed to the organizational goals and opportunist behavior becomes unlikely, which gives the organization’s members a sense of safety and familiarity. Secondly, if opportunist behavior does occur, both honest and dishonest behavior of organizational members is likely to be discovered, and an organizational culture serves also as a control instrument.

Finally, stable values, beliefs and rituals legitimize and rationalize the individuals’ behavior as long as it is in line with the communicated culture (Smircich, 1983; Wilkins & Ouchi, 1983; Pfeffer, 1981; Peters & Waterman, 1983; Rüegg-Stürm & Gritsch, 2001). Authors consequently find that organizational culture is an efficient coordination instrument in those settings, which are characterized by high performance ambiguity and low opportunism, more concretely in technologically advanced industries where teamwork is common and where individual performance is highly ambiguous, such as in R&D settings (Wilkins & Ouchi, 1983; Ouchi, 1980).

**Cognitive approaches.** Approaches that are subsumed under this heading suggest that an organization’s members’ comparable interpretations and mental models serve as informal coordination mechanism. Authors have referred to this mechanism as ‘shared mental models’ (Espinosa et al., 2001&2002), ‘organizational cognition’ (Smircich, 1983), ‘collective mind’ (Weick & Roberts, 1993), or ‘dominant logic’ (Prahalad & Bettis, 1986; Bettis & Prahalad, 1994). Organizational groups, who were found to share their mental
models include top management teams (Prahalad & Bettis, 1986; Bettis & Prahalad, 1994), software development teams (Espinosa et al., 2001&2002) or airplane flight deck crews (Weick & Roberts, 1993), all of which are situated in highly uncertain settings.

Organizational members who are coordinated by means of mental models share not only implicit and explicit knowledge, but they also interpret it identically. In closely working together and in frequently exchanging their thoughts and views, the organization’s members develop comparable explanations and expectations of their surroundings. Mental models generate organizational coordination, since the people who make use of the models tend to conceptualize and categorize a certain event similarly, drawing on appropriate actions in a rapid and efficient way (Prahalad & Bettis, 1986; Weick & Roberts, 1993). At the same time, established cognitive models do serve as a simplifying filter mechanism in a complex, ever-changing world, since the models help individuals to focus on certain aspects, while others remain largely ignored (Bettis & Prahalad, 1994).

Cognitive coordination mechanisms have been seen as efficient and appropriate for those organizations that are situated in complex environments full of ambiguous information. Such information-rich, but interpretation-poor systems require that the organizational members respond to new challenges in a rapid and correct way. Empirical findings support this assumption and authors report on positive effects between individuals’ shared cognitive models and coordination in large-scale software development teams as well as on stronger effects for distributed as opposed to collocated teams (Espinosa et al., 2002). Authors have agreed that in an organization the development of shared mental models requires its members’ a team’s collective working- and team experiences and their frequent face-to-face encounters (Espinosa et al., 2001&2002; Prahalad & Bettis, 1986).

Organizational routines. Organizational routines are defined as “a relatively complex pattern of behavior (or the theoretical representation of such a pattern) triggered by a relatively small number of initiating signals or choices and functioning as a recognizable unit in a relatively automatic fashion” (Winter, 1986: 165). Authors have described routines to differing degrees by which individuals can personally influence their organizations’
activities. On one extreme, routines were seen as a fixed response to a given stimulus, which is characterized by the absence of search and in which individuals can only just react (March & Simon, 1958). On the other extreme, routines were seen as interactions that are not mindless or automatic, but which are rather effortful accomplishments and which require their members’ conscious activities (Giddens, 1984). A broad range of further authors’ interpretations of routines ranges between these two perspectives (e.g. Nelson & Winter, 1982; Pentland & Rueter, 1994).

To clarify the concept of routines, authors have referred to various metaphors that help to better describe the concept, such as grammar (Pentland & Rueter, 1994), genes or a sort of memory (Nelson & Winter, 1982). The authors have agreed that routines are repeated, social and inter-subjective organizational processes. Routines serve as a coordinating device that connect humans, resources and knowledge in an organization, since routines store operational patterns of the individuals’ behavior. Therefore, routines generate continuity and stability, since the organizational members’ interconnected behaviors become foreseeable, and consequently the individuals are supposed to behave in a way that is generally accepted by the organization of which they are a part. It is for this reason that routines simplify work processes and tend to relieve individuals from making fundamental decisions on reoccurring situations; routines economize on ‘knowing how’ (Nelson & Winter, 1982).

2.5. Firm coordination, organizational capabilities and new forms of organizing

Both formal and informal coordination mechanisms serve to connect the organization’s members, with knowledge residing in the organization and other resources that can all be described “as stocks of available factors that are owned or controlled by the firm. [...] These Resources consist, inter alia, of know-how that can be traded (e.g. patents, and license), financial or physical assets (e.g. property, plant and equipment), human capital, etc.” (Amit & Schoemaker, 1993: 35). Formal and informal coordination mechanisms are therefore organizational capabilities, which primarily aim at connecting and exploiting organizational resources. Organizational capabilities characterize “a firm’s capacity to deploy resources, usually in combination, using organizational processes, to effect a desired end,
They are information-based, tangible or intangible processes that are firm-specific and are developed over time through complex interactions among the firm’s resources” (Amit & Schoemaker, 1995: 35).

From a capability-based point of view, it is not a specific resource alone that creates organizational value; rather, it is how well a number of resources are combined (Eisenhardt & Martin, 2000), which is, at the same time, one major research question in the capability-based line of research (e.g. Kogut & Zander, 1996; Loasby, 1998; Conner, 1991; Conner & Prahalad, 1996; Grant, 1991). Organizational capabilities are alternatively referred to as 'combinative capability' (Kogut & Zander, 1992; Van den Bosch et al., 1999), 'architectural competence' or 'architectural knowledge' (Henderson & Cockburn, 1994; Henderson & Clark, 1990), 'resource recombination' (Galunic & Rodan, 1998), 'integrative knowledge' (Helfat & Raubitschek, 2000), or 'integration capabilities' (Grant, 1996a). In general, an organization’s capabilities characterize how well the organization is capable of coordinating its own activities.

Both formal and informal coordination mechanisms tend to achieve unity of effort in an organization by means of various simplifying mechanisms and by means of reducing an organization’s complexity, since both types of coordination instruments help its members to focus on certain aspects while others need not necessarily be considered in decision-making. Simultaneously, variability, novelty and search processes are reduced in the organization. This may lead to a competency trap as a result of the difficulties organizations experience with adapting to changing internal or external conditions (Levinthal & March, 1993; Levitt & March, 1988). Thus, the quality that, on the one hand, creates stability, orientation, efficiency and helps to exploit existing resources, can, on the other hand, also lead to inertial forces within organizations. These inertial forces prevent the organizations from fundamentally changing or from generating radically innovative output (Leonard-Barton, 1992).

Paradoxically, not only formal coordination instruments can induce organizations’ inertia, but also informal coordination mechanisms. Many informal coordination instruments are deeply rooted in the organizational member’s system of values (organizational culture), mental structures (cognitive approaches), or behavioral patterns (routines), all of which constitute a part of the human personality trait that they do not
tend to give up easily (Kotter, 1996). Informal coordination mechanisms are consequently one reason why humans' have difficulties adapting to new situations. When confronted with fundamental organizational changes or reorientations, deeply embedded informal coordination instruments are, therefore, often even more rigid and more difficult to change than their formal counterparts (Smircich, 1983; Bettis & Prahalad, 1994; Levinthal & March, 1993).

Taken together, coordination is, per se, a static and closed concept (Teece et al., 1997), while innovation and change are open concepts (Chesbrough, 2004). In a world faced with rapidly changing environments and that is characterized by processes of creative destruction (Schumpeter, 1934 & 1942), continuous change, learning, flexibility and innovative output are permanently required by organizations in order to remain competitive in the longer run.

Against this background and in order to guarantee for both openness and closedness, or innovativeness and coordinated organizational behavior, prior research has suggested a number of new forms of organizing that lie beyond the traditional market-firm continuum (Volberda, 1996; Daft & Lewin, 1993). Authors suggest, among others, an ambidextrous organizational design (Tushman & O'Reilly, 1996; Smith et al., 2002; He & Wong, 2002), the platform organization (Ciborra, 1996), the learning organization (Fiol & Lyles, 1985; Huber, 1991; Levitt & March, 1988), organizations as Jazz orchestras (Hatch, 1999), intra-firm and inter-firm (social) networks (Miles & Snow, 1986; Powell, 1990; Burt, 2000), communities of different kinds, including online communities or communities of practice (Benkler, 2003; Brown & Duguid, 1991) and the virtual organization (Ahuja & Carley, 1999; Wiesenfeld et al., 1999). All of these new organizational forms have in common that the task of coordination is no isolated task, but closely related to an organization’s task of generating innovation and change.

New forms of organizing aim at simultaneously guaranteeing for stability and adaptability, for coordination and innovation, for Schumpeterian and Pareto rents, or for exploration and exploitation (e.g. March, 1991; Volberda, 1996; Rivkin & Siggelkow, 2002; Levinthal & March, 1993). An organization’s capabilities of generating coordination and innovation are, therefore, two sides of the same coin, and they characterize an organization’s dynamic capabilities (Teece et al., 1997). In addition to this, coordination
and innovation compete, however, for scarce resources, with their generation signifying a trade-off (March, 1991).

From a theoretical perspective, the generation of dynamic capabilities is grounded in managerial and organizational processes (coordination, learning and reconfiguration) and this type of organizational capabilities is defined as “the way things are done in the firm […] or patterns or current practice and learning” (Teece et al., 1997: 518). Empirical studies confirmed that firms that simultaneously pursue an exploration and exploitation strategy are likely to achieve superior performance in terms of their sales growth rate (He & Wong, 2002). Prior literature has, however, to date not been able to clearly determine exactly how an organization should be designed in order to generate dynamic capabilities, Smith et al. (2002) suggested and empirically found an ambidextrous design to be realized by means of differently composed organizational subunits with some tightly structured units (exploitation), and some loosely structured units (exploration). Other authors suggested the position of a stabilizing CEO who has access to a rich flow of information and who widely searches for new pieces of knowledge. Alternatively, researchers suggested that an ‘ideal’ organization should be composed of a hierarchy with very capable subordinates and with frequent human interactions across functional departments for decision-making (Rivkin & Siggelkow, 2002).

In the remainder of this study I will present one new form of organizing that is successful, coordinated and innovative: Open source communities. To take a closer look at the peculiarities of this type of organization thus has the potential to reveal how these organizations manage to cope with the dual nature of coordination and innovation.

2.6. Research gaps

Prior contributions on coordination in and between organizations suggests a broad variety of coordination mechanisms and organizational forms; the authors, however, left some unanswered questions that will be dealt with in the remainder of this study. These research gaps are primarily connected to the challenges of ever-changing internal and external conditions that call for a Schumpeterian (1942) form of competition. Major research gaps are,
Firstly, formal and informal coordination mechanisms have so far been recognized both as complements and substitutes. Informal coordination mechanisms, however, have to date not been discussed as a sole type of coordination that is effective in organizations without any supporting formal mechanisms. Instead, informal coordination mechanisms were always discussed in combination with (at least some of) their formal counterparts. We do not yet know if informal coordination mechanisms alone are strong enough and manageable enough for achieving integration in an organization. If so, under what circumstances is their isolated use purposeful? These questions are most interesting, since new organizational forms (for instance, online communities) seem not to be grounded in any formal coordination mechanisms; notwithstanding, they are capable of bringing about a product that is the result of many peoples’ efforts.

Secondly, recent research is discussing new forms of organizing, which are supposed to be more suitable in meeting the challenges of today’s competitive requirements, such as a continuous generation of innovation and change or achieving a balance between exploration and exploitation (March, 1991). Many new forms of organizing go beyond our traditional understanding of firms, including networks, virtual organizations or online communities.

These organizational forms are unequal to organizations on the market-firm continuum, since they are not entirely profit-oriented. Since the coordination mechanisms that have been discussed in prior literature have exclusively been explored and tested in traditional organizations, we cannot say whether they even exist at all, or similarly, whether they are suitable for all types of new organizational forms, which have entirely new characteristics compared to traditional firms. Thus, the process of how coordination is achieved in new forms of organizing remains an unresolved topic. In addition to this, while established organizational forms are well captured through traditional organization theory, no encompassing and coherent theory exists for new forms of organizing. For this reason, Romanelli called the search for new forms of organizing one of the critical unanswered issues in organizational sociology (Romanelli, 1991). Research is consequently required to find out how coordination in new forms of organizing actually is accomplished.
Thirdly, in order to fill the above-identified research gaps, we will first have to take a closer look at how new forms of organizations actually function and at the type of peculiarities that make them so different from what we already know. Prior work has already been able to give initial insights into various new forms of organizing (e.g. Achtenhagen, 2001), but our knowledge about open source software communities still remains limited. Since many open source projects are both innovative and well-organized, gaining more insights into how they exactly function could also help to shed light on the concept of dynamic capabilities (Teece et al., 1997). While this concept is clear from a theoretical perspective, it still lacks an empirical foundation to clarify how organizations should in fact be designed to generate dynamic capabilities.

In the remainder of this dissertation I will provide for answers to these open issues. The first and the second issue, which ask whether any informal coordination mechanisms are capable of coordinating an organization without relying on any supporting formal mechanism as well as whether new forms of organizing require new forms of coordination, will empirically be analyzed in parts 2 & 3 of this work. In the remainder of this first paper, I will discuss the third issue from a theoretical point of view and present three peculiarities of (coordination in) open source projects. Throughout all parts of the dissertation, I will refer to the specific setting of open source communities. The following chapter will start with an introduction to the open source phenomenon and provide for some background knowledge about the projects’ origin and on the fundamental idea in which the phenomenon is grounded, both of which is necessary to fully understand how these projects are designed.
3. The Open Source Software Development Phenomenon

Open source software development communities (OSS), such as the Linux operating system, the Apache webserver, or the Sendmail email router constitute one new form of organizing that brings about many successful or innovative projects. Since open source projects differ broadly from what we know from traditional firms, they are characterized by a number of peculiarities that, at the same time, open new options for humans’ cooperation and for the products they generate. In order to understand the peculiarities that are closely connected to open source communities, I will present some fundamental background knowledge about the open source phenomenon and on the ideas that led to the phenomenon’s initiation.

I will simultaneously refer to the phenomenon as ‘open source software development projects’ or ‘-communities’. The following paragraphs will briefly discuss (1) what open source software development is, (2) the open source origin and licenses, (3) and open source communities’ significant influence on the global software- and hardware industries.

3.1. What is open source software development?

Open source software (OSS) is freely available for all to be used, and at the same time, it can be modified or improved by anybody with the appropriate skills to do so. Interested developers work collaboratively, voluntarily and without renumeration on specific software on internet-based open source software development platforms. Independent from commercial firms the contributing individuals generate complex and often strikingly innovative programs.

Open source software projects cover a wide array of software types, ranging from spamfilters, office and text formatting tools, web browsers, encryption software to operating systems or computer games. The projects’ size, stages of development and success are similarly different to some very famous and successful projects to which tens of thousands of developers contribute, and many small projects that are used and coded
by a single person (see e.g. www.sourceforge.net). Most projects were initiated by an individual or a small group of developers who had a software idea that fit their specific needs or interests. After they coded a first rough version, they set up an internet domain and a mailing list. They invite others to download the program’s code for using the software, or to participate in writing code, bug tracking or in discussions.

A typical project exists exclusively on a common virtual platform and the contributors’ communication takes place by means of internet-based devices. Most critical for their entire communication are, firstly, the project’s homepage where the project’s aim, frequently asked questions (FAQ) or other general information on the project can be found. A second crucial tool is the development mailing list, on which the contributors’ discuss their code modifications and other community-related issues. The code repository characterizes a third tool through which the contributors to an open source project communicate, and in which the developed program along with some documentation on the individuals’ coding work is being stored.

At every point in time, interested individuals can access a project’s communication tools and the information that is being stored on a project website. A project’s members usually do not know each other in person and they may be located in different countries worldwide from where they contribute across different time zones. If the contributors so wish, they may remain completely anonymously and reveal nothing but a nickname, such as ‘thelema’, or an email address, as for instance, ‘Ferrariman356@...’.

### 3.2. Open source software origin and licenses

The origin of software engineering dates back to the 1950 and 1960s, when computers were mainly found in the research laboratories of American universities, where they served research purposes. During that time, a small group of people who pursued similar interests in computer technology established their own language and behaviors for collaborating. They were able to exchange these ideas over the ARPANET, the internet’s predecessor. Members of this community are often referred to as ‘hackers,’ a notion defined by Stallman as “someone who loves to program and enjoys being clever about it” (Stallman, 1999: 53). One central feature of their shared behavior was the free
exchange of software patches or of complete programs. By mutually exchanging
software, they were able to review, modify and further develop it according to their
personal needs, since at that time software programs were available only in a very
limited number (Raymond, 1999a).

A fundamental prerequisite for their unlimited software exchange was that the
source code of their programs was open and readable. Programs consist of source code,
sequences of instructions that a computer will execute, plus brief documentation from the
developer that will help all succeeding readers to understand the software. In order to
properly run a program, a computer transforms the code into a binary version, which
consists of sequences of numerical digits (zero and one) that humans hardly can
interpret. The availability of source code is, therefore, crucial to fully understand or to
modify software.

Along with the rise and commercialization of computer technology during the
early 1980s, a number of commercial firms started to restrict the free access to their
software and so withheld their programs’ source code, such as, for instance, for the UNIX
operating system. To the original hacker community, the firms’ activities signified a
strong violation of their norms of close cooperation and of free knowledge exchange:
“The rule made by the owners of proprietary software was, ‘If you share with your neighbor, you
are a pirate. If you want any changes, beg us to make them.’” (Stallman, 1999: 53). Richard
Stallman, a developer at MIT’s Artificial Intelligence Lab, felt additionally attacked in his
fundamental rights of personal freedom. Stallman started, in response, firstly, to develop
a complete and free UNIX-compatible operating system and mobilized his fellow
developers to join his efforts. Playing by words, he named the emerging operating
system GNU for ‘Gnu’s not Unix’ (Stallman, 1999).

Secondly, Stallman recognized that it was necessary to protect the freely available
GNU software to prevent the program from the commercial actors’ privatization or
commercialization, and he made use of existing copyright law. He invented a license that
guaranteed the software’s current and future free use for all. Stallman’s GNU GENERAL
PUBLIC LICENSE (GNU GPL) is often referred to as ‘copyleft’, “Copyleft uses copyright law,
but flips it over to serve the opposite of its purpose: Instead of a means of privatizing software, it
becomes a means of keeping software free.” (Stallman, 1999: 59). According to the GNU GPL
LICENSE, free software guarantees every user four basic freedoms, namely to run a program, to study and modify it, and the rights to copy and redistribute copies of the program. The license does not allow, however, to add any restrictions to the software.

Copyleft explicitly forbids the privatization of future derivatives from software that was developed under this license, and it prevents the programs from a concentration of property rights to the software by any owner. Software in the public domain, in contrast, can be modified and made proprietary. Today, a number of comparable free software licenses are in use, but most licenses are less restrictive than the GNU GENERAL PUBLIC LICENSE, since they allow for the return appropriation from the software’s derivatives (see Perens, 1999).

Finally, in 1985, Stallman founded the FREE SOFTWARE FOUNDATION (FSF), a tax-exempt charity, aiming to further strengthen the principles of free software and thus defending humans’ fundamental rights of personal freedom, “…Think of free speech, not free beer” (Stallman, 2001: 7; Stallman, 1999 and 2001; Free Software Foundation, 1991). At that time, firms were largely reluctant to integrate free software into their business models. The notion ‘free software’ was misleading and it held too strong of a connotation of software that is free of costs or, more importantly, free of returns. Ten years later, in 1998 a group of leading free software developers met against this background to evoke the OPEN SOURCE INITIATIVE, which aims at a stronger promotion of the software’s practical ends, putting less emphasis on the philosophical meaning of personal freedom that is associated with open source software.

The initiative established the new notion ‘open source software’ in order to avoid the dual meaning of ‘free’ in the sense of liberty and price. By and large, open source software satisfies similar requirements as free software albeit less restrictive, since it allows for a privatization of the programs’ code modifications (Perens, 1999; DiBona et al., 1999). In the remainder of this work, I will focus on the term ‘open source software’ to refer to all types of ‘free software,’ ‘libre software,’ or ‘open source software,’ and define it along with the open source definition (the exact wording of the open source definition is available in the appendix).
Against this background, open source software can be characterized by three critical features that distinguish it from software developed in firms, Firstly, open source software is coded by a virtual community and a network of users who are, at the same time, the software’s developers. Secondly, open source licensing agreements allow for indirect, but not for direct return appropriation, and finally, the software’s crucial element, its source code, is open and freely available to all.

3.3. Influence on the global software- and hardware industries

During the few years since open source communities’ initiation in the mid-1980s, the public interest in open source software projects grew rapidly, as the estimated number of projects, users, and developers that are currently in place, shows. SOURCEFORGE, for instance, a host for open source software projects that are sponsored by VA SOFTWARE Corporation took up their services in 1999 with one project and four users/developers. As of November 2004, the site hosts some 90,000 projects, daily adding more than 60, and to which some 950,000 registered users/developers contribute, including a daily increase of more than 700 (SourceForge, 2004).

Open source software projects’ rapid establishment becomes additionally clear through the success stories of some very popular projects, such as in the case of the LINUX operating system. In 1987, Andrew Tannenbaum developed MINIX, a simplified version of UNIX, which was mainly a teaching tool used for University computer science courses. As a result, many students suggested improvements of MINIX, most of which Tannenbaum refused to implement, Linus Torvalds was one of the frustrated MINIX users and on October 5th 1989, he posted a message to the MINIX mailing list under comp.os.minix,

“… I’m working on a free version of a Minix-lookalike for AT-386 computers. It has finally reached the stage where it’s even usable (though may not be depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02 … but I’ve successfully run bash, gcc, gun-make, gun-sed, compress, etc, under it,”

(Linus Torvalds, LINUX founder)
Linus Torvalds, who was at that time a student at the Helsinki University, called the new operating system LINUX, put together by the terms Linus and UNIX. Starting in 1989, with one developer, one user, and 10,000 lines of code, Torvalds has been able to continuously attract interested co-developers (Torvalds, 1999; Raymond, 1999a+b; Kroll, 1999).

In 1994, LINUX was able to compete on stability and reliability with commercially developed systems, as it runs without crashing, freezing or having to be rebooted, and it is known to run extremely fast and efficiently. In 1998, LINUX consisted of 1.5 Million lines of code; it had 7.5 Million users and some 10,000 developers (Torvalds, 1999; MacCormack & Herman, 2000). And today, LINUX is supposed to be in use by approximately 14 to 18 million registered users, although the correct number of users cannot be determined exactly, and it is even supposed to be several times higher (Linux Counter, 2004).

LINUX is the second most widely used operating system worldwide (28.5% market share) following MS WINDOWS (49.2%) (Netcraft, 2003) and it is the first operating system that was entirely developed in the open mode. Although LINUX is an amazing success story and consequently by no means reflects the average open source project, it is an example (next to some other very successful projects, such as the APACHE webserver software that runs on about 60% of all servers worldwide; Netcraft 2003) of the potential inherent in software that is developed beyond company boundaries.

As a consequence of the rapid growth and diffusion of the open source community, commercial players on the software market integrated open source software into their corporate strategies, which shows that they perceived open source software as a serious, yet different competitor on their respective markets. Steve Ballmer, CEO of MICROSOFT, called LINUX their ‘threat number one’ (o.V., 2001). Two internal Microsoft documents, known as the ‘Halloween Documents’ supported Steve Ballmer’s statement and proposed MICROSOFT’s intention to use the free development regime for their own purposes (Valloppillil, 1998 a&b). Additionally, MICROSOFT runs a public campaign against freely developed software (Perens, 2001).
Hardware firms, such as IBM, Dell, Sun Microsystems, or Apple, in contrast, equip their computers on a regular base with the Linux operating system, and they publicly announced their support of free software projects (O’Reilly, 1999), which proves open source software’s high quality. In addition to that, commercial players initiated open source software projects for their own purposes such as, for instance, Apple with their operating system Mac OS X named ‘Darwin’, or Netscape with their web browser ‘Mozilla’ (Hamerly et al., 1999). To date, however, none of the commercial firms has been able to repeat the success of Linux with their open development projects. One reason for this might be, that we do not yet exactly know how these projects work in detail, and under which conditions it makes sense for commercial corporations to take over the open regime for their purposes.

Other ventures during the 1990s started to generate indirect returns from the freely developed software, mainly by initiating innovative business models, One interviewed software entrepreneur claimed, for instance,


"Without open source software, our company would not exist."

(CEO, entrepreneurial software company, Switzerland)

The business model of the firm in question is grounded upon a combination of consulting, content management of open source software and of support. Another example is the Linux distributor Red Hat Software, which is based on a similar business model, and which was incorporated in 1995. Red Hat offers convenience to the freely available Linux operating system, since it combines deployment, training, consulting and technical support for the users’ internet infrastructure (Young, 1999).

Although many business models that are grounded in open source software are still in an infant stage of development, the above-mentioned two examples illustrate that some ventures that are grounded on open source software have been seen as credible enough to receive external funding. This impression is similarly reflected in a WR Hambrecht study which expects future revenues from the market for Linux products and services to rise with a compound annual growth rate of about 90 percent (Pantel, 2000). These expectations are to date, however, not reflected in the NASDAQ quoted prices of Linux distributors such as Red Hat. The quoted prices were continuously declining.
since the end of the year 2000, which also shows the lack of knowledge and experience on freely developed software and on business models connected to it.

The above-illustrated numbers and anecdotes indicate that software developed in open source projects proved to be successful against commercial competitors in the computer- and software industry, which is often seen as one of the most complex and most volatile industries, “The most dynamic and fast-paced industry in the world – computing” (Brown & Eisenhardt, 1998: ix), which turns these projects into a phenomenon that significantly influenced global economic activity. In comparison to other communities or new models of organizing, open source software development is far more elaborate, having reached a far greater number of contributors, and thus exerting a significantly greater influence on economy and society.

It is for this reason that open source software development projects are an interesting field to study new forms of organizing and it is worth taking a closer look at these communities. In the following, I will present three peculiarities of open source projects and illustrate why these communities are, on the one hand, so different from the traditional organizational model, and what, on the other hand, makes them interesting for those types of work that aim at generating some creative or innovative output.
4. Three Peculiarities of Coordination in Open Source Software Development

In the following, I will discuss three peculiarities of open source software projects. Many of these communities are strikingly successful or innovative, and at the same time, they do not appear chaotic or unorganized (Markus et al., 2000). In terms of coordination, open source projects differ largely from what we know from firms, which could be a reason for their suitability for the generation of innovative work. It can thus prove useful to shed more light on the characteristics of open source projects.

I will focus on the following three peculiarities of open source projects: (1) Open source projects generate a public instead of a private good, and formal coordination instruments would not be accepted by the project contributors; (2) in open source projects, the total amount of coordination is reduced when compared to firms, since the contributors to open source communities focus exclusively on exploratory work, whereas supportive or administrative tasks play no crucial role in this setting; (3) open source projects allow for stability and volatility simultaneously, since they attract two types of contributors who fulfill different roles in their communities.

4.1. Public good and the failure of traditional formal coordination mechanisms

Open source software development communities differ from firms with respect to the output they produce. Neoclassic economic theory distinguishes between two core types of goods, *private goods* and *public goods* (Samuelson, 1954). While firms produce private goods to which they hold all property rights, open source communities are characterized by the production of public goods. The category to which a produced good belongs influences how coordination of the process of product generation can be achieved. Both types of goods and their influences on coordination in open source software communities will be discussed in the following.
**Private goods.** Only a limited number of users can consume a specific private good, and these consumers have acquired such products on the market along with the right to use, modify or consume the good. All other consumers who are not in charge of the product have no right to do the same. Private goods are exclusive and rivalrous in consumption as, for instance, all products we can buy in a shop. A firm, defined as a collection of non-human assets, holds the ownership of all residual control rights to the goods it has produced (Grossman & Hart, 1986; Hart & Moore, 1990; Hart, 1989). At time of production, all property rights to use or change the good, to transfer it to a second actor, and to keep the resulting returns from the good, remain therefore with the entrepreneur (Furubotn & Pejovich, 1972).

Since all benefits that flow back from the produced good are exclusive, the producer will be the only recipient and he or she will have the greatest incentives to produce or to innovate a specific good. Not all products are exclusive by their very nature, and spillovers may occur. To the entrepreneur, spillovers signify a loss of returns and in order to avoid the loss, he may use legal protection mechanisms, for example, licenses, patents, copyrights or trade secrets. For intangible goods, the above-mentioned protection mechanisms fail to entirely catch up the illegal activity, as the free exchange of music or DVD files over the internet illustrates.

Since all property rights to the originally produced good remain with the entrepreneur, he has the central authority to determine and monitor what will be produced, how and where this will be done, and who will perform what kind of work in the production process. The entrepreneur's legitimacy to do so leads to the design of an explicit company architecture with a formal hierarchy and with clear principal-agent relationships that enable for an exchange of work and knowledge against money. These relationships will structure, organize and coordinate the workflow of an average company that, for instance, produces software, such as Microsoft, as the following example will illustrate.

At Microsoft, software is developed in small ‘feature teams,’ ranging from 3 to 8 people in size. By and large, these teams work independently of all other software development teams in their firm, but they must stick to tight schedules that detail their project’s milestones (planning, development and stabilization) by numbers of weeks.
During the development process, about 20-30% of software specifications might change, mainly during the initial stages of project development. In the final stages of development, only few and minor modifications are seen as justified, while radically innovative ideas to the software are not accepted by the company. The feature teams, therefore, devote only little time and few resources to purely inventive or intellectual exercises that bring no direct financial returns, but which could be crucial for future value generation (Cusumano & Selby, 1995).

For Microsoft, recruiting of new software developers to the feature teams signifies a key task and it takes place along a number of pre-defined criteria, such as technical talent, sense for business, or a strong commitment to Microsoft. Once at Microsoft, employees have the chance to climb the vertical promotion ladder mainly if their predecessors leave their posts (Cusumano & Selby, 1995). In sum, this short anecdote illustrates that software developers at Microsoft produce a private good and during the innovation and production process the programmers must stick to explicitly given rules, structures and to a formal company hierarchy, which all guide and coordinate their activities.

**Public goods.** Public goods, in contrast to private goods, are free to all and no individual can automatically be excluded from their consumption, no matter how many people already possess, use or consume the good. Public goods are non-rivalrous and non-exclusive (Samuelson, 1954) as, for instance, nature or streetlights. Public goods can be subject to freeriders, consumers who do not contribute to the burden of the products’ provision, a phenomenon known as ‘tragedy of the commons’ (Hardin, 1968; Lloyd, 1833).

The property rights to use, change or transfer a public good and to keep the returns from it cannot be withheld by one single entrepreneur. Spillovers may occur frequently, and any transaction costs are too high for earning the entire control over a public good. Rational commercial producers have consequently no incentive to produce or to innovate this type of good, because the returns from such products do not flow back to the entrepreneur exclusively. Since they may be subject to under-provision, public goods are most often produced by public hands (Olson, 1971).
Open source software falls into the category of pure or mixed public goods, as explicitly defined by open source licensing agreements. No central authority has the legitimacy to formally determine who does what, when, how or where, and no principal-agent relationship in the traditional sense exists in the context of an open source project. Additionally, no formal hierarchy or organizational structure can be in place that translates an entrepreneur’s authority to the organization or which coordinates the overall project activity.

With regard to who contributes how to an open source project, no explicit or implicit working contracts are in place that define the duties or rights of project members, and no explicit incentive structure does exist, since contributors to a community volunteer their time, Proactive recruiting measurements cannot take place, and a certain project cannot rely on a stable body of workforce or on the contributors’ frequent participation. An open source software project depends, instead, on volunteers and neither the individuals’ educational background nor their skills or personal knowledge need be revealed to any other member of the project. Since contributors come and go, an open source projects’ boundaries are fluid and can at no point in time clearly be determined in an open source community. There is also no control of when (during a day, a week, a year, …) project members need to contribute, how long their efforts will take, or if a piece of work will find its completion at all.

Similarly, no one has the right to determine what the relevant tasks and preferences are while working in the context of an open source community. Contributors to a certain project also have the rights to decide by themselves to what subject and how much they wish to contribute. No formal processes split up the total mass of work (of which the boundaries might be nebulous in and of themselves), and no formal rules of cooperation or knowledge exchange exist between the voluntarily contributing developers. And finally, since the open mode of working over a virtual platform does not require any personal face-to-face contacts, electronic devices enable all project contributors to determine for themselves where a piece of work will be accomplished. If contributors so wish, they can remain in entire anonymity as, for instance, in the TikiWiki open source project that is outlined below. Table 1.2 summarizes the preceding discussion.
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Closed Source Development (Company)</th>
<th>Open Source Development (Project)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of good</td>
<td>Private good; Rivalrous in consumption, excludable,</td>
<td>Public good; Non-rivalrous in consumption, non-excludable,</td>
</tr>
<tr>
<td>Property rights</td>
<td>Centralized to owner or managers; Subject to patents, trade secrets, copyrights, for buying and using the software (source code), Consequences; Direct return appropriation.</td>
<td>Diluted; Open source licensing models, allow at least for viewing, using, modifying, redistributing the software (source code), but not for direct return appropriation.</td>
</tr>
<tr>
<td>Boundaries</td>
<td>Rather clear,</td>
<td>Blurring, changing and unclear.</td>
</tr>
<tr>
<td>Freeriders</td>
<td>(Theoretically) no freeriders do exist,</td>
<td>Freeriders do exist,</td>
</tr>
<tr>
<td>Owner – employee relationship (principal – agent)</td>
<td>Traditional principal – agent relationship; Work and specialized knowledge against money. This allows for influencing, defining, controlling and monitoring as to …</td>
<td>Questionable who is principal and who is agent, no exchange of money against work. As a consequence no central determination as to …</td>
</tr>
<tr>
<td>... Who works how?</td>
<td>Active recruitment, stable workforce, working contracts, skills and backgrounds of employees are known, workers are known.</td>
<td>No active recruitment, no stable workforce, no working contracts, skills and backgrounds of employees are not necessarily known, workers may remain in complete anonymity.</td>
</tr>
<tr>
<td>... When:</td>
<td>Timeframe and deadlines can be determined,</td>
<td>Timeframe and deadlines cannot be determined,</td>
</tr>
<tr>
<td>... On what:</td>
<td>Central definition of relevant tasks, formal work distribution,</td>
<td>Definition of tasks and formal work distribution do not exist,</td>
</tr>
<tr>
<td>... Where:</td>
<td>Mostly common workspace, frequent face-to-face contacts,</td>
<td>No common workspace, no or few face-to-face contacts, virtual platform,</td>
</tr>
<tr>
<td>Coordination through</td>
<td>Market; Price mechanism, Firms; Property rights, principal-agency relationship, formal structure and hierarchy.</td>
<td>?</td>
</tr>
</tbody>
</table>

Table 1.2: Property rights, principal-agent relationship and consequences for coordination in open and closed source software development.
The TikiWiki software development project (www.tikiwiki.org) generates an open source content management system and groupware that can be used to create all sorts of web applications, sites or portals (e.g. newsletter, blog, image gallery, etc.). As outlined on the project website, the TikiWiki mission is fairly broad and rather product- and community-oriented than market-oriented, and financial returns play no role within the context of the project. The TikiWiki mission reads,

“To be a powerful, leading, complete, user friendly, stable and secure full featured open source web application which includes (but is not limited to) cms & groupware features; To have fun and friendly community where people can easily contribute & enhance Tiki; World domination.” (TikiWiki, 2003).

Contributors to the TikiWiki project are located worldwide and 13 randomly selected and interviewed project members report to be residents of 10 different nations on all continents, which makes frequent personal encounters or realtime (internet) meetings almost impossible. Proactive recruitment of new developers does not take place for the TikiWiki project. If they wish to contribute code, interested contributors are, instead, invited to submit an email, or to contact one of the project administrators. In contrast to a commercial firm, the project contributors’ skills, personalities or names must thereby at no point in time be revealed. Although the project website displays a developer profile page, only 32 of 158 core developers reveal some limited information on themselves as of November 2003. For instance, the contributor with the nickname ‘redflo’ submitted the following:

“Personal Information: Login name: redflo; Real name: Florian Gleixner; Email address: redflo at users.sourceforge.net; Site membership since: 2001-03-27 12:23; User ID: 183 182. Resume: Born, working will die some time. Inbetween having some fun :-).”
(redflo, contributor to the TikiWiki project)11.

For a contribution to the project, no deadlines exist, and the only milestones for TikiWiki are project releases. Changes to the project are consequently allowed at any time, and no one determines what should be modified.
As a consequence of the differences between an average firm and an open source community, traditional formal coordination mechanisms on the market–firm continuum are bound to fail in open source software projects due to two reasons. On the one hand, since open source software is free to all, neither a market nor a price (and contract) exists which can coordinate the project’s activities. Similarly, no bundled property rights and no principal-agent relationships are in place that legitimate some central entrepreneurial authority to formally set up a number of coordination mechanisms, such as organizational structure or an explicit hierarchy. Since traditional formal coordination mechanisms are grounded entirely on this precondition, they make no sense in the open source software development setting, and no contributor would ever stick to a formal rule. Contributors to the FREENET open source project (www.freenetproject.org) correspondingly describe their work as follows:

“Yes... we have had project plans; they’ve been universally ignored. The problem with having any form of hard-core plan is that you cannot tell people to do things because they’re volunteering their time. And so any form of set-in-stone plan really necessitates some form of authority and you can't have authority over somebody who is really a volunteer. You can try to persuade them, that they should do something and that it’s a good idea to do it, but you can’t just tell them to do it.”

(Ian Clarke, founder of the FREENET project)

On the other hand, even if someone had the legitimacy to determine formal coordination mechanisms, coordination in the traditional sense were problematic in an open source project, since the required information for centralized decision-making or planning is not available at any point in time. Open source projects are subject to extremely uncertain conditions that are caused by permanently changing situations (Dемil & Lecoq, 2008), whereas uncertainty is defined as “the difference between the amount of information required to perform the task and the amount of information already possessed by the organization” (Galbraith, 1973: 5). Uncertainty is, therefore, a function of the diversity of necessary inputs, outputs, and the level of the task’s goal difficulty.

Such a characterization of open source projects comes close to Cohen, March and Olsen’s (1972) definition of an ‘organized anarchy,’ organizations in highly ambiguous situations. These organizations can be described by uncertain and fluid membership and
by ill-defined problems and solutions that come together by accident like pieces of waste in a garbage can. Planning or formal, centralized coordination processes make no sense in such organizations, since many causalities between the elements remain nebulous, and the organizational members’ activities are grounded in frequent trial and error processes, “Where goals and technology are hazy, and participation is fluid, many of the axioms and standard procedures of management collapse” (Cohen et al., 1972).

In changing situations, the number of exceptions becomes so large, that no bureaucratic rules are capable of capturing the reality’s complexity. Purely bureaucratic, formal coordination mechanisms, however, fail when tasks become highly unique or ambiguous (Ouchi, 1980). For organizations that permanently deal with innovation and environmental complexity, formal bureaucratic coordination mechanisms are therefore inadequate (Burns & Stalker, 1961). Since informal coordination mechanisms provide rather for a framework in which the organization’s members are supposed to act instead of giving clear advice for what needs to be done in a certain situation (unlike e.g., formal plans or programs), this type of coordination seems to be more appropriate for open source projects (see also the argumentation on informal firm coordination in paragraph 24).

The line of thinking illustrated above is confirmed by Lawrence and Lorsch’s (1967a) finding that the greater the certainty of an organization’s subsystem’s (sub-) environment in diversified organizations, the more formalized the system’s structure will be, Martinez and Jarillo (1989) in their meta-analytical study on firms in a multinational context found additionally that the number of published papers that refer to informal coordination mechanisms increased since the 1980s. Their findings refer to a time span in which the firms’ internal or external situation turned to become more uncertain than it was before. Organizations increasingly tend to use informal coordination mechanisms when confronted with uncertainties, I conclude and propose:

P1a: Successful open source software development communities will rather rely on informal than on formal coordination mechanisms.

P1b: Successful open source software development communities will make less use of traditional formal coordination mechanisms than unsuccessful communities or than firms.
4.2. Reduced need for coordination in open source communities

Organizations differ with respect to how much coordination is necessary until their output, or final product reaches the end user. A second peculiarity of open source projects is that compared to firms, they have generally a reduced need for coordination effort.

In general, software innovation is a non-routine activity, since many software projects are one-of-a kind with no prior prototypes, and many tasks in the software development process are inherently unpredictable. Any specifications that pretend to predict or fully plan the software development process are invariably incomplete (Kraut & Streeter, 1995). In order to operate correctly, software development does, however, require a precise integration of the necessary components and modules which is often a challenging task, as Curtis and colleagues (1988) found to be the case in their empirical study. Through their study of 19 software development projects in 9 firms, Curtis and colleagues conclude that coordination and communication breakdowns are one of the three most salient problems in software development projects.12

Many software development projects in both settings, open and closed source software development, are additionally very large and far beyond the ability of a small group to create the software by themselves. Large projects, instead, require the unity of effort of many individuals. Coordination is consequently a crucial task in both closed and open source software development, (Crowston & Kammerer, 1998; Kraut & Streeter, 1995; Curtis et al., 1988; Faraj & Sproull, 2000).

Since the general architecture of open source software development differs broadly from software development under the closed regime, the two organizational forms are also in need for different amounts of coordinative effort. Organization and contingency theories are capable of giving some hints on what determines the amounts of coordination necessary in a specific context. Such hints are enough to broadly compare the open and the closed modes of software production and to give tendency statements on how much coordination these forms of organizing will require for the production of some comparable output. Organization and contingency theories suggest the following variables that influence how much coordination a specific organization will require.
These variables are thereby not independent of one another, but correlate positively: The degree of task diversity in an organization, the degree of differentiation in an organization's subsystems, the size of an organization, and the degree of task or subsystem interdependence.

The underlying logic is that complex organizations consist of a number of differentiated subsystems, each of which pursues various subtasks of the overall organizational goal. Differentiation means the number of an organization’s subsystems, or “the state of segmentation of the organizational system into subsystems, each of which tends to develop particular attributes in relation to the requirements posed by its external environment” (Lawrence & Lorsch, 1967a: 4). An average organization will group tasks that are most similar in the same subsystem (e.g. in divisions or in functions), while tasks are most different between an organization’s subsystems.

Diversity is not just limited to the tasks to be fulfilled in an organization, organizational subsystems also vary with respect to their members’ orientation towards others, or to the individuals’ goal orientation, as Lawrence & Lorsch (1967a) found in their seminal study on the chemical processing industry. Heterogeneity of any kind creates itself instability in an organization that, in turn, requires more coordinative effort as compared to more homogeneous organizational tasks or subsystems. As a general rule, the more subsystems an organization has, and the greater the difference between the subsystems, the more coordinative effort will be necessary to produce some coherent output. Besides, the more variation in an organization’s tasks, the more coordinative effort is necessary for the organization (Lawrence & Lorsch, 1967; Perrow, 1967; also: Coase, 1937).

An organization's size is positively correlated with its degree of subsystem differentiation and subsystems’ task variability. Larger organizations are found to be more differentiated and to have more variability between their tasks than smaller organizations, yet with a declining rate. As a consequence, the larger an organization becomes, the more coordination it will require (Blau & Schoenherr, 1967; Child, 1972b; Pugh et al., 1969).
The extent to which an organization requires coordination additionally depends on how strongly its subsystems’ tasks are interdependent, which characterizes “the extend to which unit personnel are dependent upon one another to perform their individual jobs” (Van de Ven et al., 1976: 324). Task interdependencies can be of various kinds. Among others, scholars refer to the matching of interdependent goals (Cyert & March, 1963), decisions (March, 1999; Cyert & March, 1963), resources (Pfeffer & Salancik, 1978), strategies (Chandler, 1968), values, interpretations and cognition (Schein, 1982; Smircich, 1983; Prahalad & Bettis, 1986; Bettis & Prahalad, 1994), workflow and processes (Thompson, 1967; Galbraith, 1973; Malone & Crowston, 1994; Malone et al., 1999), or capabilities and knowledge (Grant, 1996, Kogut & Zander, 1996). In general, the stronger the interdependence between organizational subsystems, the more coordination is necessary.

A number of general statements can be made on how much coordinative effort is required for the production of software in firms as opposed to software that is generated in open source projects, without the need to take a closer look at the specific process of software development. Such statements can be grounded on the design of the entire organization and its divisionalization (into specific functions) that do or do not exist in each of the two settings discussed here. I assume that an open source project and a firm produce the identical piece of software, yet in different organizational settings and surrounded by different environments. All statements reflect tendencies that result from a comparison of the two forms of organizing, closed versus open source software development. In the following I will briefly outline the situation of both types of organizations.

**Closed source software development in firms**, Large companies in general, and more specifically software firms, are characterized by a large degree of differentiation that is created on various company layers. As described by Curtis, Krasner & Iscoe (1988, see also figure 1.3), these organizational layers are (1) the individual, and (2) the software development project, in which people directly work on the software development task. In the project, coordination refers mostly to the management of knowledge interdependencies, since software innovation is, first and foremost, knowledge work (Faraj & Sproull, 2000). In addition, a firm’s layers cover (3) the entire company with its functions such as marketing, sales, production, or logistics, etc., and finally, (4) a firm’s...
firm is influenced by its surrounding *business environments* with customers, suppliers or other co-contractors who are only indirectly involved in the software development task.

The involved environmental parties' goals or interests *differ* broadly and the differences are especially great between a firm's various functions, between a firm and its stakeholders, or between the firm's specific environments (Lawrence & Lorsch, 1967). Each of the involved parties is crucial for a firm's proper functioning. As a consequence, the goals that are pursued by a company's various functions are of a conflicting nature, since they compete for the identical resources such as specialized knowledge held by individuals, money, and time. Their use is often characterized by a conflicting relationship of the parties involved, and an efficient use of these resources is crucial for firms as profit-seeking entities, and a firm's goals and tasks are, at the same time, highly *interdependent*. Similar things apply for the relationships between the firm and its stakeholders and for those between the various stakeholders with conflicting interests (e.g., banks, customers, shareholders, etc.).

Curtis and colleagues (1988) have studied firms that worked on large software projects. They report that breakdowns were likely to occur at organizational boundaries (between the firm and its external stakeholders), and that coordination across these boundaries was extremely important for the projects' success. One central boundary constitutes the firm's relationship with its customers, since the software developed is supposed to fit their needs. For both sides, producer and customer, it is difficult to understand the other's point of view, since they do not entirely share the identical knowledge. For customers, it is difficult, if not impossible, to understand the requirement specifications that are written by the producers' requirement analysts (Crowston & Kammerer, 1998). On the other hand, software designers were found to have difficulties interpreting their customer's needs as indicated by their requirement statements (Curtis et al., 1988).

Software development firms claim that designing a system that fits their customers' needs, and specifying a system that can actually be programmed by the developer, constitute two of three major problems of large software development projects (Crowston & Kammerer, 1998). In addition to this finding, other studies report similar coordination problems that occur at the external boundaries of software development.
firms. If multiple firms are involved in the process of software development, their coordination is even more complicated as compared to single-company projects (Curtis et al., 1988). A brief characterization of MICROSOFT further illustrates these arguments.

**Figure 1.3:** The layered model of software development in firms and in OSS communities: Differentiation, interdependence and size.
Adapted from Curtis, Krasner & Iscoe (1988).

MICROSOFT pursues a standard setting strategy, and they aim at serving the mass-market and to maximize their own profits. The company is structured horizontally along four divisions (platforms, applications, administration, sales & support) and vertically, is the company is structured along several explicit hierarchical layers. About 75% of the MICROSOFT personnel are located at their headquarters in Seattle, WA, where most of the software development activities take place. The employees’ collocation enables for their frequent formal or informal personal encounters and for ongoing discussions or knowledge transfer (Cusumano & Selby, 1995).

Only about 30% of the Seattle-based employees are directly involved in the process of software development as program-managers, software developers and testers. 17% of the employees, in contrast, work exclusively on administrative tasks (finance and administration), and about 50% of the Seattle-based personnel works on a mixture of
both types of tasks (customer support, marketing & sales). These figures illustrate how much of a firm’s efforts become absorbed by coordination, organization and their translation into administrative work, while only a fragment of the firm’s efforts directly flow into the software development work (Cusumano & Selby, 1995). For open source projects, the situation is different, as will be illustrated below.

**Open source software development.** Software developed in open source communities, in contrast to that developed in firms, is characterized by a smaller degree of task or subsystem differentiation, task variability and interdependence. Open source software communities focus almost exclusively on the innovative task of software development and on related activities, such as website maintenance or, to a limited degree, on project documentation. Other functions which are crucial for a company, for instance, human resource management, finance, production, etc. simply do not exist, since an independent project that relies on voluntary contributors, does not require these functions. An average open source project is therefore less differentiated than an average firm.

The contributors to an open source project can additionally concentrate their efforts on a single resource, their knowledge, while other resources are either entirely unimportant or do simply not exist in the context of their software development work. Time, money, opportunity costs or a specific location, for instance, strongly determine a firm’s every day work. In the context of an open source project, these resources may be crucial for the personal decisions that an individual contributor makes, but they are unimportant for the project’s progress. Every individual is free to decide for himself or herself on how much he or she is willing to donate to the project. The project contributors can focus their entire efforts on the software (knowledge) creation and recombination task. This means that they are confronted with a lower level of task variability than their colleagues who work for a profit-oriented firm.

Moreover, open source communities exist widely independent from any stakeholder groups that are situated external to the projects, such as customers or banks. Although open source projects do have users, these persons do not have the identical rights as a firm’s customers have. Developers to an open source project may consider the
wishes and comments that the users have for the software, but all project contributors are ultimately free to work on whatever they find interesting or challenging (as long as it is legal). An open source project consequently does not have to react to any market pressures, since no financial transactions, obligations or any other interdependencies exist between the project and any other external parties. A layered model of an open source project’s architecture and its degree of subsystem and task differentiation, variability and interdependence, is, therefore, only limited when compared to that of an average software development firm (figure 1.3). The following example of the TikiWiki open source project (as of December 2003) will further illustrate these arguments.

The TikiWiki open source software development project has no explicit organization structure with no project divisions existing that are similar to those of a firm. The project is grounded upon 300 software modules and on 354 files in the code repository. The project members contribute to the software development process through self-selection of the tasks they wish to complete, and every developer’s personal interests, goals and methods for doing so are respected at all times. For the project, a number of different contributor types can be distinguished, but the project website refers explicitly to 158 developers who have write access to the code repository and to 4 administrators.

The latter do not exclusively devote their efforts to administrative tasks (as the term indicates), but they range among the project’s 10 most active developers. As of December 2003, three of the four administrators, who can be found on the website, had contributed 59% of all code changes and 41% of all lines of code to the entire project (TikiWiki, 2003). Coordination in TikiWiki is no planned task that absorbs a lot of organizational resources. In a private email to the researcher, one maintainer describes coordination in TikiWiki as follows,

"We are not well organized in the common sense, but more like a biological organism in an ecosystem." (Contributor to the TikiWiki project)

Open source software projects in sum require a lower level of coordinative effort when compared to firms, since they do without many functions and tasks that determine a firm’s every-day work, this given that the two organizations produce the identical piece of code. Open source projects rely on a reduced level of subsystem differentiation, of task
variability and of interdependence of their work that surrounds the actual process of software development. For the generation of the identical piece of software, firms do require more functions in the organization, with more people working on a greater number of tasks that vary broadly, these being interrelated with a greater number of external parties than open source software development projects. I summarize the preceding discussion and propose:

P2: All else being equal, software created in open source software development communities requires *less coordination* until it reaches the end-user than software created in firms.

### 4.3. Simultaneous realization of stability and volatility:

**Dynamic capabilities**

Stability and volatility or, exploration and exploitation (March, 1991), are crucial concepts which characterize dynamic capabilities (Teece et al., 1997) and which are called for by numerous researchers (e.g., March, 1991; Volberda, 1996; Teece et al., 1997). *Social network theory* deals with a corresponding pair of concepts, namely with *diversity* and *closeness* (Burt, 2000), both of which are said to enable for competitive advantages, with each of the concepts generating them for different reasons. In doing so, social network theory refers to different types of ties between people, namely *weak* and *strong ties* that have different characteristics, Open source software communities’ particular composition of core developers and peripheral members simultaneously enables for both diversity and closeness, which signifies a third peculiarity of this type of organization I will characterize in the following.

*Diversity & weak ties. Creativity is defined as “the production of novel and useful ideas in any domain” and it is a starting point for innovation, or “the successful implementation of creative ideas within an organization” (both: Amabile et al., 1996: 1155). The rationale for both concepts is similar, new things result from using old ideas in new combinations, new ways or places (Sutton, 2002; Schumpeter, 1934; Weick, 1979). Social network theorists claim that more creative and more innovative solutions to any task*
come into existence when people who have access to different pieces of information merge their knowledge and so create a large number of knowledge combinations. The access to different knowledge sources is found to be most valuable in those fields of activity where people are engaged in activities in which the generation of novelty is crucial (Rodan & Galunic, 2004).

A network, the set of people to whom an individual or an organization is directly or indirectly linked, can provide for such sources of knowledge. Although networks cannot replace knowledge in an organization, they affect the knowledge’s flow into or within the organization. In order to generate a flow of widely different pieces of knowledge, social network theory finds that a large number of unique and weak ties are more suitable than a number of comparably stronger, but redundant ties (Burt, 1992 & 2000; Granovetter 1973). Weak ties are of rather superficial or casual character, and the individuals have only little emotional investments in these type of ties. Moreover, weak ties are short-term in duration, composed of individuals usually sharing a low frequency of contacts. This type of tie is thus also less reliable and more uncertain than stronger ties (Aldrich, 1999).

Weak ties allow for an access to widely dispersed pieces of information, and should allow for the quick transfer of knowledge between the involved individuals. This type of tie between humans can either be direct or indirect, and the latter type is characterized by the effectiveness of brokers, individuals who facilitate contacts between otherwise unconnected humans who, in turn, have access to different sources of information (Burt, 1992 & 2000; Granovetter 1973).

Empirical work provides evidence for the above-mentioned line of thinking and demonstrates that diversity on each of the team, organizational or inter-organizational levels of analysis leads to the generation of innovative output. Sources of diversity are thereby the work group members’ backgrounds (e.g. age, occupation, nationality, gender, etc.) (e.g., Mellewight & Kloninger, 2003; Ancona & Caldwell, 1992); a moderate degree of work group or organizational turnover (e.g. March, 1991; Katz, 1982; Ancona & Caldwell, 1992; Bantel & Jackson, 1989; Virany et al., 1992), or contacts between individuals that are boundary-spanning in nature (e.g. Hansen, 1999; Hargadon & Sutton, 1997; Baum et al., 2000; Stuart & Podolny, 1996; Rodan & Galunic, 2004; Liebeskind et al., 1996).
Hargadon & Sutton (1996 & 1997) illustrate, more concretely, how knowledge
diversity can generate new ideas and new products on the organizational level. They have
studied the case of IDEO, a Silicon Valley-based product design firm that retrieves and
stores existing design solutions from a number of industries and newly combines the
ideas in their internal brainstorming sessions. The outcomes of this process are new
product designs. Other studies on the inter-organizational level report on a higher
probability of innovation or patent output from firms that established alliances with
partners outside their technological area (Stuart & Podolny, 1996), or which have
multiple kinds of alliance partners at the time of startup (Baum et al., 2000). On the team
level, Ancona & Caldwell (1992) found, additionally, that R&D teams which had
established a greater number of contacts to various corporate functions internal to their
own firm brought about more innovative solutions than teams with a smaller number of
contacts. Besides, Clark & Smith (2002) found for top management teams that their
external networks increased organization-level innovation in their firms under study.

Other empirical findings suggest that the value of working groups’ R&D functional
diversity (Ancona & Caldwell, 1992) and top management teams’ age heterogeneity
(Mellewigt & Kloninger, 2003) is positively related to the groups’ capability of bringing
about innovative output and for initiating strategic change. Working groups’ age and
greater average tenure diversity was found to positively influence their groups’ innovation
and performance output (Ancona & Caldwell, 1992). Working groups with, on average, a
shorter tenure on average (Bantel & Jackson, 1989; Mellewigt & Kloninger, 2003) or, with a
higher education of their members (Bantel & Jackson, 1989), were additionally found to be
capable of bringing about more innovative results than it was the case for groups with
the adverse characteristics. All of these examples illustrate that diversity in knowledge
gained through weak ties could positively influence the organizations’ capability for
generating innovations of any kind. At the same time, weak ties bear the danger of a lack
in communication and coordination between the involved individuals, which can both be
achieved more successfully through strong ties (Burt, 2000).
**Closeness & strong ties.** Closeness and strong ties between a network’s members is characterized by durable, longer-term oriented relationships and through frequently repeated patterns of communication between the individuals. The contacts are more reliable, more trustful, and more predictable than weak ties (Aldrich, 1999). In a network of strong ties, the involved individuals tend to be connected to the identical individuals, or to humans who are collocated. Their contacts are likely to be redundant, and the individuals have access to the identical (instead of different) pieces of knowledge, At the same time, frequent contacts may lead to an assimilation of peoples’ bodies of knowledge, since the transfer of knowledge is enhanced through strong ties (Burt, 2000 & 2002; Granovetter, 1973).

These findings are most of all relevant for knowledge that is complex, such as tacit, non-codified knowledge, or for strongly interdependent knowledge, which requires humans to already have an overview of the larger system of which they are a part (Hansen, 1999). Weakly-tied interactions, in contrast, are adequate for the integration of codified, or of modular pieces of knowledge. In general, the stronger the ties between two individuals, the more alike their knowledge will be (Granovetter, 1973).

*On the one hand,* strong ties undermine knowledge diversity in organizations and so constrain the creation of new things. *On the other hand,* close and frequent interaction between people enhances their capability of coordination (Reagans & Zuckerman, 2001, see the literature review on informal coordination in paragraph 2.4.). Closeness therefore generates stability, but it tends to be a static concept (Burt, 2000). In order to generate innovative output, however, neither type of tie and interaction is sufficient for an organization; indeed both types are required simultaneously for the generation of both coordination and innovation (Reagans & Zuckerman, 2001) This, also, characterizes dynamic capabilities, “the firm’s ability to integrate, build, and reconfigure internal and external competences to address rapidly changing environments” (Teece et al., 1997: 516).

Open source communities manage to cope with the seemingly contradictory qualities of closeness and diversity, since they combine the best of both worlds. Unlike firms, the functioning of open source projects is grounded upon two distinct types of project contributors, core developers and peripheral project members. The members of these two groups possess and use different types and different amounts of knowledge
that is relevant for the specific project of which they are a part. The most obvious
criterion in order to distinguish between the two contributor types is that core developers
have direct access to a project’s code repository (CVS), while peripheral project members
do not. The latter type of contributors can only indirectly contribute code to a project by
submitting their contributions to the community’s mailing list. Moreover, members of the
core developer group have strong ties to members of their own group, while members of
the peripheral contributor group have rather weak ties to all the other members in the
community inside and outside of their own group. I will briefly outline both groups’
characteristics.

Core developers & strong ties. In an average open source project, a small circle of
core developers are interconnected through close, strong relationships that are long-termed
nature, which require considerable investments of time, which have a high frequency of contacts and involve considerable emotional intensity and cohesion. The
FREENET open source project will serve as a suitable example to illustrate this concept.14

For the FREENET project, two types of contributors can be found. One member type
represents a small circle of core developers, and a second type is a larger group of
peripheral project contributors. In the year 2000, a total of 356 contributors were
subscribed to the project’s developer mailing list of which 30 individuals (8.4%) ranged
as core developers, and 326 (91.6%) as peripheral contributors (von Krogh et al., 2003b).
A comparable relation between the number of members of these two groups has similarly
been found for other communities. Zhang & Storck (2003), for instance, report that 90% of
the contributors to the online community which they had studied ranged as peripheral
project members, The core developers’ characteristics differ broadly from those of the
peripheral contributors.

In FREENET, the average core developer longevity in the project in 2000 was 185.97
days (sd, 131.56), seven times longer than the peripheral members’ longevity. One core
developer additionally recalled his average time investment in the project in an interview,

“Probably I average about an hour, between an hour and two hours a day, [...] This
summer, when we were trying to implement the key types and everything else, I was
probably working between six and eight hours a day on it. [...] That really was my summer job and I was probably spending six or eight hours a day, seven days a week.”

(Core developer C, The Freenet project)

Furthermore, the members of open source communities’ core developer groups were additionally found to be responsible for large fractions of the coding work, which also illustrates the considerable time investments to the projects to which they contribute (Koch & Schneider, 2000; von Krogh et al., 2003b; Mockus et al., 2000 & 2002; Lakhani & von Hippel, 2002). The core developers’ time investments are similarly reflected by a high frequency of contacts on the major communication tool, the developer email list. In the FREENET project, in 2000, a total of 11,210 email messages, summarized in 1,714 message threads, appeared on the development mailing list. The 30 core developers alone contributed a total of 6,751 (60.22%, sd, 459.54) emails, which makes an average of 225 emails per core developer in 2000 (von Krogh et al., 2003b), 16.5 times as many emails compared to the average peripheral member. The core developers tend primarily to communicate with the members of their own group. During an interview, one FREENET core developer described his emotional involvement and cohesion to the other members of the core developer group as follows,

“Everybody knows me already and I have earned the respect of the rest of the core developers [...] I would say that the people I most closely associate in Freenet are definitely the core developers A and B, and a couple of others, the client [core] developers, core developer I and X. There is a few more of them.”

(Core developer C, The Freenet project)

Core developers, however, perform different work than do peripheral members. They control and develop the code base, while peripheral members repair and report defects (Mockus et al., 2000 & 2002)(see also table 1.4).

Taken together, the core developers build a relatively coherent and stable stock (DeCarolis & Deeds, 1999) of an open source project’s knowledge. Members of this group share strong ties, and since they frequently exchange their personal knowledge and, at the same time, generate most of a community’s knowledge, they should know similar things about their project. The core contributor group is thus comparable to a firm’s
stable workforce of employees. This group, therefore, has the function of an organizational memory (Walsh & Ungson, 1991) that provides for stability and coherence in the virtual setting.

<table>
<thead>
<tr>
<th>Core developers</th>
<th>Peripheral members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong network ties within their own group, weak ties between groups</td>
<td>Weak network ties within and between groups</td>
</tr>
<tr>
<td>Long-term project affiliation</td>
<td>Short-term project affiliation</td>
</tr>
<tr>
<td>Stocks of knowledge</td>
<td>Flows of knowledge</td>
</tr>
<tr>
<td>Gatekeepers and integrators</td>
<td>Broker positions</td>
</tr>
<tr>
<td>Sameness, stability, coherence, coordination, closure</td>
<td>Variation, change, newness, diversity</td>
</tr>
<tr>
<td>Work on central parts of a project</td>
<td>Work on the project periphery</td>
</tr>
</tbody>
</table>

Table 1.4: Comparison of core developers and peripheral project members in open source software development.

*Peripheral contributors and weak ties.* Unlike profit-oriented firms, an open source community consists of a second type of contributor, peripheral members. Peripheral project contributors build a large body of people who have rather *weak ties* to the other members of their own group, and to members of the core developer group. Their ties can be characterized as being *short-term oriented*, with only *little time investment* in the communities to which they contribute through a *small frequency of contacts* and only *weak (if at all) emotional investments or cohesion* with their colleagues. This is illustrated by the following anecdote on FREENET’s peripheral members.

In the FREENET open source project, the peripheral members’ *average project longevity* in 2000 was 24.84 days, which is about one eighth of the core developer’s longevity on the project. On average, members of the peripheral contributor group report on only *little time investment* in the project with the peripheral contributors continuously changing, as becomes clear by means of the following remarks provided through an open questions survey that was sent to peripheral contributors of the FREENET project,
“I probably only spent a few hours total in my life [on Freenet].”
(Peripheral member #9, the Freenet project)

“Reading mostly, not a lot of time, probably a month or two at an hour or so a day."
(Peripheral member #11, the Freenet project)

“As I recall there were 20 - 30 messages a day on the mailing list, so I would have spent
about 15 - 20 minutes on average reading them.”
(Peripheral member #7, the Freenet project)

The peripheral contributors comparatively small time investment is mirrored in
their low frequency of contributions to the FREENET developer mailing list. In 2000, each of
the 326 peripheral members of the project contributed 13.6 emails to the list on average
(in total 4,449, von Krogh et al., 2003b), which is about one seventeenth of the average
core developers’ effort. Other studies report on comparable figures, Dempsey and
colleagues (1999) analyzed the LİNX application and utility community in a longitudinal
study, which excludes core developers and therefore focuses exclusively on the LİNX
peripheral members. Their results indicate that most peripheral authors contributed only
once or twice to the project, and the authors analyzed a 5-year period between 1995 and
1999. In addition, peripheral community members report a weak emotional involvement
with respect to their respective projects, and they seem to share only a small degree of
cohesion with the other members of their community. Peripheral members to the FREENET
project consistently report,

“My involvement in Freenet was very limited, I’m certainly not any sort of contributor. I
never made it a priority.”
(Peripheral member #12, The Freenet project)

“I don’t know any of the developers (and I forget the founder’s name right now,) I’ve never
had a role in Freenet.” (Peripheral member #1, The Freenet project)

Peripheral community members were, furthermore, found to fulfill tasks that core
developers do not wish to take over, such as functionality testing, bug reporting or
defects repairing, all of which can be fulfilled by contributors who have no deep
understanding of the project’s entire software architecture (Mockus et al., 2000 & 2002;
Figure 1.5: Stability and volatility through stable and changing membership of core developers and peripheral project contributors.

An average open source project profits from the changing body of peripheral members in a twofold way. On the one hand, the universe of potential peripheral members of an open source project is vast, if not virtually endless, which ensures a continuous flow (DeCarolis & Deeds, 1999) of new knowledge into a project. On the other hand, peripheral project contributors to open source software projects are not only permanently changing, but due to the projects’ worldwide distribution and impersonal, anonymous character, their members will have external networks that are redundant only to a very small degree (through their friends, colleagues at work or school, families, etc.). In open source communities, peripheral project members have the roles of knowledge brokers who bridge structural holes between their personal contacts and the remaining project members, mainly the core developers. The peripheral contributors to an open source project are responsible for variation in a certain community, and since they continuously throw new ideas into a project, they are able to prevent the rather stable group of core developers from being trapped in their own and coherent lines of thinking (figure 1.5).
Together, the two contributor types simultaneously combine volatility and stability in an open source project. Thus, a balance of a community’s number of core developers and peripheral contributors should be crucial for a project’s success.

I do summarize the preceding discussion on strongly-tied core developers and weakly-tied peripheral project contributors in the following propositions:

P3a: All else being equal, the larger the accumulated body of peripheral members in an open source software development community, the more innovative and the less stable it will be.

P3b: In open source software development communities coordination will be stronger within the core developer group than within the peripheral member group, or than between members of the core developer and the peripheral contributor group.

P3c: The relationship between an open source software development community’s average member longevity (core developers and peripheral contributors) and its success will be invertedly U-shaped.
5. Conclusions

The aim of this study was threefold. The first aim was to present an overview of previously identified organizational coordination mechanisms both formal and informal. The study’s second aim was to present open source communities as one new type of organizing and to give some background information on the open source phenomenon. The study’s third aim was to discuss three peculiarities of coordination in open source communities and to draw conclusions on the projects’ to serve as a new mode for organizing innovative work. In order to reach these goals, I discussed open source communities from the point of view of a number of theoretical perspectives and referred to organization theory, social network theory, or economic theory.

The discussion suggests the following. Firstly, in open source software communities, formal coordination mechanisms are bound to fail, since their output is a public instead of a private good and no contributor would accept any formal rules or guidelines to guide a project. Secondly, compared to firms, the full need for coordination in open source software development communities is reduced, since this setting does not rely on functions that characterize a typical profit-oriented firm. Thirdly, unlike firms, the average open source community is composed of two types of contributors. These types are few core developers with a stable membership, and many peripheral members with a fluid project membership. This structure allows for a simultaneous realization of, on the one side, coordination & stability and of innovation & volatility, on the other. The reason for this is that open source projects draw on both stable elements and on a continuous stream of external knowledge, which make the characteristics of an open organization and dynamic capabilities.

With these findings, the work advances existing bodies of literature in both organization theory and strategic management, which ask for how and how well organizations are coordinated, and which search for new forms of organizing that can adequately cope with today’s requirements. In discussing the peculiarities of coordination in the open source setting, the paper could additionally concretize the currently rather abstract concept of dynamic capabilities, Nevertheless, the work opens issues for future research,
*firstly*, although I could illustrate some anecdotes that provide for evidence to the proposed conclusions of this work, the study still remains theoretical and, therefore, it can be generalized only to a limited extend. Future studies should take a closer look at communities in general, and more specifically at open source projects in order to find out more about how coordination is achieved in this setting, and about how coordination is related to the communities’ capability of generating innovative output.

*Secondly*, although the community mode of organizing software development is promising, and various business models are being created around open source software, the organizational model still seems to be mysterious to firms. One interesting question for future research is: ‘How can profit-oriented organizations make best use of the community mode of organizing?’ This question is especially interesting, since open source software development is based on voluntary contributors who receive no direct financial returns from their community-related work, and firms are generally not familiar with this characteristic of communities.

Any firm that is planning to intervene in a community’s functioning to make use of their work and ideas could be interpreted as an offense to the contributors’ unpaid efforts, and it bears the danger of running counter to the open source philosophy. The projects’ character could consequently be subject to unintended changes such as, for instance, through changes in the contributors’ motivation to contribute (from intrinsic to extrinsic, e.g. Osterloh & Frey, 2000) or, changes in the quality or quantity of the work that the project members donate to the community. It could be that a firm’s intervention into the system of an open source project leads to the community’s collapse.

*Thirdly*, communities that are grounded upon similar principles as open source software development projects can also be found in other, mainly knowledge-intense fields of study. Compared to open source software development projects, most of these communities are smaller in size, or they are less successful. Future work should not only think of how to make use of the open mode for organizing work in a profit-oriented setting, but also of how learnings from successful open source projects could be transferred to other, non profit-oriented communities in order to find out for which circumstances this organizational model can be suitable.
Finally, a small number of cross-sectional studies on open source software communities have found that open source projects differ largely with respect to their size, their success, the produced type of software or other variables (Ghosh, 2002; Krishnamurthy, 2002). Only few projects were found to be extraordinarily successful, or to have a large number of contributors. Critical questions are, therefore: ‘What distinguishes successful open source projects from less successful ones? Why are some projects capable of attracting many contributors, while others are not?’ And, most importantly, ‘How can an open source software development project be kept alive in the longer run? How can it continuously attract new peripheral members?’ And, finally, ‘How can a project be made interesting for core developers to have them work on the software in the longer run?’
Endnotes Part 1

1 A part of the figures on the FRENET open source project (chapter 43, of this paper) emerged out of a research collaboration between the University of St. Gallen and MIT, and the following persons contributed to collect and calculate the data; Eric von Hippel, Karim Lakhani, Georg von Krogh, Petra Kugler, Sebastian Späth (in alphabetical order). All interviews with the FRENET core developers did also emerge out of this research collaboration and were led by the identical people.

2 From a knowledge-based view on the organization, firms are generally seen as amore efficient governance mechanism for the coordination and production of certain goods such as, for instance, tacit knowledge. The reason therefore is that for these goods neither a market nor a price do exist. Firms are more adequate for the generation, transfer, and protection of their knowledge, what is at the same time one source of sustainable competitive advantage in an environment that is faced with permanently changing conditions and that demands continuous innovation (e. g. Liebeskind, 1996; Foss, 1999). It might therefore be that a traditional, economic perspective is not enough to capture how organizations can best be organized, which calls for new forms or organizing beyond the market-firm continuum (Spender, 1996).

3 The literature suggests a broad variety of different, overlapping organizational structures, Mintzberg (1979), for instance, suggests five organizational forms, which he names (1) simple structure, (2) machine bureaucracy, (3) professional bureaucracy, (4) divisionalized form, (5) adhocracy.

4 The literature on organizational resources and capabilities is vast, and so are the constructs’ conceptualizations, definitions, or terms that name the capabilities. Authors use the concepts mainly in a similar, overlapping vain, yet with more or less differences in focus, I will refer to resources and capabilities as defined in the body of this paper.

5 The dual nature of capabilities, of which the benefits face a trade-off between coordination & efficiency versus newness & innovation are similarly reflected in the two types of rents that capabilities are supposed to generate; Pareto and Schumpeterian rents, Pareto rents (Klein et al., 1978), which come to existence if an organization has the potential to perform a specific task more efficiently than other organizations. This is, for instance, the case through firm-specific and specialized capabilities, which create a ‘fit’ between what a firm does and possesses (resources) and how well it performs it (capabilities), Pareto rents cannot guarantee for long-term rents, which are achieved through entrepreneurial, or Schumpeterian rents. Schumpeter saw innovation as the key economic phenomenon, which gives an impulse to economic growth, since it induces an economic disequilibrium. According to Schumpeter, organizations that are confronted with an ever-changing environment should continuously renew themselves in order to remain competitive in the longer run (Schumpeter, 1934 & 1942; also Nelson, 1996). Firms can generate Schumpeterian rents, when they take risky decisions and innovate in an uncertain environment.

6 Teece et al, define dynamic capabilities as, “the firm’s ability to integrate, build, and reconfigure internal and external competences to address rapidly changing environments” (1997: 516), Eisenhardt & Martin (2000: 1107) interpret capabilities as “the firm’s processes that use resources—specifically the processes to integrate, reconfigure, gain and release resources—to match and even create market change”.

7 The notion ‘hackers’ is today used with an ambiguous connotation. Its popular meaning characterizes the illegal use of computer systems, whereas within the software development community the notion signifies „someone who loves to program and enjoys being clever about it“ (Stallman, 1999: 53). The popular meaning of ‘hacking’ is instead called ‘cracking’: “Hackers build things, crackers break them” (Raymond, 1999).
The number of Linux users includes only users who have registered with the Linux counter, which is estimated to represent only between 0.2 and 3% of all actual Linux users. The total number of Linux users worldwide is consequently supposed to be much higher (www.counter.li.org) and any estimations of Linux’ market share show great differences in number. The netcraft web server survey presents for example the following shares of the operating systems market worldwide: 1. WINDOWS (49.2%), 2. LINUX (28.5%), 3. SOLARIS (7.6%), 4. BSD (6.3%), 5. Other non-UNIX (2.5%), 6. Other UNIX (2.4%), 7. Unknown other systems (3.6%) (www.netcraft.com/survey/).

As of October 2003, Netcraft (2003) reports on the following shares on the market for web servers: 1. APACHE (64.52%), 2. MICROSOFT (23.54%), 3. SUNONE (3.48%), 4. ZEUS (1.72%).

The responding TIKIWIKI contributors claim to be residents of the following countries: USA (3), Australia, Argentina, Belgium, United Kingdom, Canada, France, Germany, India, Uruguay.

Redflo gives some further information about the programming languages he knows, including the level of his programming skills (“want to learn — competent — wizard”) and on how deep his experience with these languages is (“<6 Months —6 Mo—2yr —2.5yr”).

Curtis, Krasrer & Iscoe (1988) found the following three fields of problems to be most salient in firm software development: (1) The thin spread of application domain knowledge; (2) fluctuating & conflicting requirements of the software developed; (3) coordination & communication breakdowns.

Interdependencies within and between organizations or individuals are additionally and synonymously categorized as sequential (Thompson, 1967) or flow dependencies (Malone et al., 1999), which occur “whenever one activity produces a resource that is used by another activity” (Malone et al., 1999: 432); secondly, as pooled (Thomson, 1967) or sharing dependencies (Malone et al., 1999), which refer to organizational subsystems that all use identical resources; thirdly, as fit dependencies, in which multiple actors “collectively produce a single resource” (Malone, 1999: 432); or finally, as reciprocal interdependencies, in which the “output of each contributor becomes the inputs for others” (Thompson, 1967: 54).

All figures and citations on the FRENNET open source software development project were gathered through telephone interviews with members of the core developer group, by means of an open questions’ survey filled in by peripheral members and through publicly available records of the community developer mailing list. All data were stored in a separate database. For additional information about the data gathering methods, see the methods parts in part 2 & 3 of this work.

Interestingly, (virtual) communities that generate innovative output can also be found in fields other than software development. Many online communities have in common, that they can primarily be found in knowledge-intense and creating fields of interest, such as: biology/ medicine (HUMAN GENOME PROJECT, eg, www.shgc.stanford.edu or www.dhg.de), classical music production (ORPHEUS CHAMBER ORCHESTRA, www.orpheusnyc.com; Hackman, 2002), architecture (VIRTUAL HOUSE, www.virtualhouse.ch), literature (PROJECT GUTENBERG, http://gutenberg.net).
Part Two

Coordinating through Dominant Knowledge:
Evidence from the Freenet Open Source Project
Coordinating through Dominant Knowledge:
Evidence from the Freenet Open Source Project

1. Introduction

How organizations coordinate and integrate their specialized and distributed efforts lies at the heart of management and organization theory. Organizational coordination in general is defined as the “process of achieving unity of effort among the various subsystems in the accomplishment of the organizational task” (Lawrence & Lorsch, 1967: 4). Coordination refers to the integration of an organization’s decentralized or specialized resources, competences or its individual members’ knowledge, and it is consequently a critical organizational capability (Teece, Pisano & Shuen, 1997; Hoopes & Postrel, 1999; Grant, 1996b). Organizational coordination determines how resources, skills or ideas become combined and transferred into a final product. Traditional views on organizational coordination are strongly grounded in organizational structure and they refer primarily to hierarchical forms of organizing along the market-firm continuum.

Recent work has increasingly sought new forms of organizing which allow for continuous innovation, learning and adaptation to the firms’ specific, often fast-changing environments (e.g., Volberda, 1996; Ilinitch et al, 1996). Those organizational forms that have been found to meet the demands of today’s knowledge economy best often go beyond our (traditional) understanding of organizations on the market-firm continuum, such as in the case of networks (Nohria & Eccles, 1992; Powell, 1990) of communities of practice (Wenger, 1998), or of virtual teams (Ahuja & Carley, 1999).

Open source software development projects (OSS) are one new form of organizing. They characterize communities of voluntary contributing software engineers who manage to collectively develop a new product. Unlike most firms, open source software projects exist exclusively on a virtual platform. Open source software development is promising, since some projects manage to produce software that is able to successfully challenges its commercial counterparts. The Apache open source web server, for
instance, served about two-thirds of the shares of the corresponding market in 2003 (Netcraft, 2003). Open source projects, in addition, have characteristics that foster learning (von Krogh et al., 2003) and intrinsic motivation (Lakhani & Wolf, 2001; Osterloh et al., 2003), both of which are prerequisites for the generation of innovative work.

Open source projects, similar to other new forms of organizing simultaneously challenge our taken-for-granted understanding of how organizations coordinate their distributed and specialized efforts, and how they manage to bring about a collective product. Since open source software is a public good, traditional (formal) coordination mechanisms we know from organizations on the market–firm continuum such as the price mechanism, central property rights, formal organizational structure or an explicitly given hierarchy, make no sense in the open source setting. Open source projects, however, by no means appear chaotic, but are strikingly disciplined (Markus et al., 2000), which indicates that some sort of coordination must be in place.

Against this background, this study aims at finding answers to the following sets of questions; first, how is coordination produced in open source communities as a new form for organizing innovative work without the need to rely on traditional coordination mechanisms that we know from organizations on the market–firm continuum? Is there any substitute for the traditional forms of coordination, that include the price mechanism, central property rights, or a formal organization structure? If so, what is it, and what does the substitute look like? Secondly, how does this substitute come into existence in the first place; how does it evolve over time?

In order to answer these questions, this work employs an inductive and qualitative case study design in the tradition of grounded theorizing (Glaser & Strauss, 1967; Strauss & Corbin, 1990). Empirical data on the complex single case of the Freenet open source project have been gathered between October 2000 and November 2003. Data sources included qualitative interviews with the project’s core developers, an open questions survey filled in by peripheral project members, plus a supporting quantitative analysis of publicly available project documents.

Briefly summarizing, my findings suggest that an informal project-specific implicit and explicit knowledge base that is shared by the project members serves as a substitute for traditional formal coordination mechanisms. Project-related knowledge, which is often
referred to as ‘hacker culture’, plays only a minor role in this process, I find that the shared knowledge base, which I term ‘dominant knowledge,’ covers three dimensions, namely the overall project direction, its technical standards and rules that guide the community’s behavior. Dominant knowledge has its origin in the early stages of a project’s inception, Project-specific knowledge goes through a four-step evolutionary process to become dominant and to coordinate a project. It transcends two boundaries, since it wanders from an individual to an organizational level (of analysis), and from a single-time to the repeated use of knowledge. Key mechanisms in this process are knowledge creation, capturing & replication and transfer, which may lead to an overall coordinating dominant knowledge base. Early project members were additionally found to profit from a ‘first knowing advantage.’

The study empirically and theoretically contributes to existing bodies of work. Empirically, the study investigates the FRENET open source software development community, an innovative project initiated in late 1999, and for which no templates or external frames of reference have existed. Theoretically, my work will result in a framework that answers the above-mentioned research questions and that contributes to prior streams of literature. On the one hand, it contributes to organization theory, since the study explores organizational knowledge as an informal coordination mechanism which is effective without the need for relying on formal coordination instruments. Additionally, the paper contributes to prior work on innovation management, since it explores an organizational model that is suitable for the uncertain and complex task of generating innovations, In doing so, this study contributes also to the growing body of literature on open source development.

The paper is structured as follows, the second section gives an overview of prior literature on open source software development from a management perspective, and it shows how little we know about a how coordination is achieved in this setting. Thirdly, I will outline the methods of this inductive and theory-generating work, Next, the findings of both a rather descriptive first-order and a more general second-order analyses will be presented and compared to well-known informal coordination mechanisms. And finally, I will conclude by discussing what dominant knowledge is not, and by presenting some promising avenues for future research.¹
2. Literature Review:

Open Source Software Development

At the time this study started (mid-2000), almost nothing was known about the open source software development phenomenon. Since then, the subject has attracted considerable attention from scholars who have looked at the topic from a broad range of backgrounds, including economics, management & organization theory, sociology or psychology. Researchers have analyzed open source projects from different theoretical stances, such as motivation theories, institutional economics, or collective action. On the opensource.mit.edu website, a host for the open source research community, for instance, scarcely 10 papers could be found on the subject as of July 2000. As of October 2004, the site recorded more than 150 papers or articles on open source software development. Much of this work, though, is laid down in working papers, many of which are still in an infant stage of development with only a limited number of articles having been published.

Prior contributions to the emerging stream of literature on open source software development have focused on a handful of topics, and their authors have contributed to two major perspectives. Firstly, prior work has taken in an external perspective from which authors have asked ‘Why does the open source phenomenon exist?’ Secondly, prior work has referred to an internal perspective and has asked ‘How, why, when do open source projects function?’ Each of these perspectives has contributed to sketching a raw picture of the open source phenomenon, and in the following I will present a brief review of the respective literature.

To describe the open source phenomenon authors have used a broad range of different terms, but they have referred to the identical phenomenon and mode of organizing. Among others, the terms in use are ‘commons based peer production’ (Benkler, 2003), ‘voluntary product development community’ (Shah, 2003), ‘private–collective invention model’ (von Hippel & von Krogh, 2003), ‘knowledge ecology’ (Lanzara & Morner, 2003), ‘gift economy’ (Raymond, 1999; Zeitlyn, 2003), or ‘bazaar governance’ (Raymond, 1999; Demil & Lecocq, 2002). I will refer to the phenomenon as ‘open source software development project’ or ‘open source community.’
2.1. External perspective

From an external perspective, researchers have explored why open source software projects exist at all, and why some projects manage to be very successful against their commercial competitors. From this perspective, open source software development has been widely acknowledged as a pure or mixed public good (von Hippel & von Krogh, 2003; Johnson, 2000; Bessen, 2001; Edwards, 2001b) to which a large number of volunteers freely provide input at their private expenses (von Hippel & von Krogh, 2003). The production of a public good bears thereby the danger of negative effects that are generated by freeriders, which might create inefficiencies for the entire organization. Software, however, is a complex good, for which the conventional wisdom of free-riding inefficiencies is turned upside down, and open source projects have the potential to outperform companies’ profit-oriented software production (Bessen, 2001).

Open source projects’ characteristics can, at the same time, generate positive effects by means of low production costs and by their wide information networks, which have the potential to increase the software’s quality and innovativeness. In comparison, closed source software that is developed in firms suffers from information disadvantages, since firms rely on a smaller number of developers and users, who make suggestions for improvements to the software (Johnson, 2000; Bessen, 2001; Kuan, 2000; von Hippel, 2001; von Hippel & von Krogh, 2003; Benkler, 2003). While closed source software may consequently be subject to market failures, open source projects have the potential to generate software that would probably never have been developed in a corporate environment (Johnson, 2000). In general, the open mode of organizing seems to be most promising for the production of either information or culture, provided that the physical capital that is required for its production is widely distributed (Benkler, 2003).

Harhoff and colleagues (2000) have used a game-theoretic approach and they have found that it even pays for self-interested agents with complementary capabilities to freely reveal their information and their innovations to others. Their finding has been found to be true even when the actors are confronted with direct rivals. As a consequence of the positive effects of the open mode of organizing, open source projects have a great potential to turn into an established institution, similar to traditional markets, firms, or hybrid forms of organizing. Communities, consequently, open up new options for
organizing work beyond the pure production of software (O’Mahony, 2002; Benkler, 2003; Dalle & Jullien, 2000; Kuan, 2000; Khalak, 2000).

All contributions that have approached the open source phenomenon from an external perspective have focused on the projects’ environment. Prior work of this tradition has therefore interpreted the projects themselves as ‘black boxes’. Work in this tradition has consequently failed to analyze in detail how, why, and when central internal processes of open source projects take place, and if they do exist at all. These contributions have generated useful insights into the social desirability of open source software, its rapid success and existence in general. However, the authors have been unable to give any insights into how open source projects function in detail, and they have failed to exactly mirror how coordination is achieved in this setting. In order to fully understand and make use of the open source organization model, the latter insights are still a necessary prerequisite.

In addition to that, prior work that has analyzed open source software production from an external perspective suffers from methodological deficiencies. Some authors’ work has remained entirely theoretical and without any empirical evidence to test their quantitative economic models, their statistical experiments, or their computer simulation approaches at all (Johnson, 2000; Khalak, 2000; Bessen, 2001). Others have not empirically tested their model (Harhoff et al., 2000; Benkler, 2003), or they have not revealed their data sources or analysis methods to the reader (Dalle & Jullien, 2000; Kuan, 2000). They remain exclusively theoretical and give only limited insights into the phenomenon of open source software.

In order to fully grasp, analyze and understand a new phenomenon during its initial stages of discovery, research requires to deeply dig into the phenomenon’s empirical reality and compare the emerging theory with what can be empirically observed. The current early stage of discovering open source projects calls therefore for a second, internal perspective to the phenomenon, to which this study aims at contributing.
2.2. Internal perspective

The second category of research in the field of open source software development takes in an internal perspective and it explains how, why and under which circumstances these projects function. Prior contributions have taken a closer look at three areas of sub-problems, and prior work has primarily been grounded in empirical studies. The three areas of interest are, firstly, what motivates and incentives programmers to voluntarily contribute to a project? Secondly, what characterizes the community of contributors to an open source project? Finally, how are open source projects organized, and how is coordination produced in this setting? In the following, I will briefly outline prior contributions to these areas of interest.

Motivation & incentives. Prior work on the first area of study has looked at what motivates programmers to voluntarily contribute to an open source project, without being paid for their efforts. Prior studies have suggested that the following reasons play a crucial role in motivating the contributing individuals and can be divided into intrinsic and extrinsic motivational factors. Intrinsic motivators include the pure enjoyment of programming, having fun and challenging their own creativity (Lakhani & Wolf, 2003; Lakhani & von Hippel, 2002), as well as the contributors' learning effects (von Krogh et al., 2003a; Hertel et al., 2003). Extrinsic motives include cases such as gaining a good reputation within the entire open source community (Harhoff et al., 2000; von Krogh et al., 2003a; Moon & Sproull, 2000; Weber, 2000), directly making use of the improved or newly created software (Hertel et al., 2003; Moon & Sproull, 2000; Lakhani & Wolf, 2003), and finally, using the public exposure gained from the software to bolster one's career position (Lerner & Tirole, 2000; Moon & Sproull, 2000). Taken together, the individuals' motives to contribute to an open source community build four different clusters of contributor types2 (Lakhani & Wolf, 2003).

Prior studies have suggested that the strongest driver for contributing and for spending time on open source projects is the individuals' enjoyment-based intrinsic motivation that is effective when a person feels creative during his or her work. Extrinsic motivators (e.g. community reputation, status, beating closed source software development), in contrast, are less important (Lakhani & Wolf, 2003), and the advantages
of contributing outweigh the perceived disadvantages, such as the contributors’ time investments, or the lack of payment for their work (Hertel et al., 2003). One reason therefore is the contributors’ access to synergetic effects inherent to so-called communal resources (reputation, control over technology and learning opportunities), which can only be accessed through collective work. How much a developer will get from contributing to an open source project differs by involvement. The more a contributor is willing to invest, the more he or she will personally get in return for the project (von Krogh et al., 2003a).

Developer community. Authors who have analyzed the second area of interest have described the characteristics of the open source project community of contributors, their behaviors and their relationships to each other. The voluntarily formed communities create positive external effects on the market and their output remains systematically undervalued. The communities tend to attract people who seek socio-psychological and hedonistic (in contrast to monetary) rewards, which is especially true for young adults and teenagers who wish to position themselves in society, who have only little money, but who still have a long span of life to live ahead of them. For the open source communities to be kept alive it is thus key to maintain their intrinsic rewards and so attract those contributors who are intrinsically motivated (Benkler, 2003).

Prior work has reported on two major types of contributors to open source projects; firstly, (core) developers, and secondly, lurkers or peripheral contributors. Authors have suggested a variety of notions to name these contributor types, such as ‘developer group’ and ‘interested readers’ (Hertel et al., 2003), ‘committer’ and ‘short-term participants’ or ‘long-term participants’ (Shah, 2003), ‘developer’, ‘contributor’ and ‘lurker’ (von Krogh et al., 2003a), ‘kernel developer’ and ‘obscure developer’ (Dalle & Jullien, 2000), or ‘developer’ and ‘maintainer’ (Moon & Sproull, 2000). By and large, all of the authors have referred to similar types of contributors.

Core developers often represent only a small circle of a project’s contributors who are responsible for large fractions of the coding work (Koch & Schneider, 2000; von Krogh et al., 2003b; Mockus et al., 2000 & 2002; Lakhani & von Hippel, 2002). For the Linux kernel development, Ghosh (2002, and Ghosh et al., 2002) found, for instance, a
Gini coefficient of 0.79 to be effective. Core developers, though, perform different work when compared to peripheral members. Core developers control and develop the code base, while peripheral members repair and report defects (Mockus et al., 2000 & 2002). In open source software projects, an extreme concentration and a widespread distribution of work are, therefore, aspects that are simultaneously effective (Ghosh, 2002).

Von Krogh and colleagues (2003b) have further distinguished between joiners, newcomers and developers in the core developer group, and they find that membership to this group is relatively stable. Joining the core developers, though, is not costless. It requires a ‘feature gift’, new code modules or features (rather than improvements of existing codes) that a newcomer donates to an ongoing project. Open source projects were therefore also referred to as ‘gift economies’ (Raymond, 1999; Zeitlyn, 2003), in which no such thing as free gifts exist. Instead, receiving a gift creates an obligation of reciprocity to the receiver who will be expected to do the same in the future.

Peripheral project members, or lurkers, make the majority of many projects’ contributors. Zhang & Storck (2003) have found, for instance, that 90% of their focal online community’s participants were peripheral members. Since the members of this group fulfill tasks that developers wish not to take over themselves, such as functionality testing, bug reporting or the repairing of defects, peripheral project members constitute a crucial part of open source projects (Mockus et al., 2000 & 2002; Nonnecke & Preece, 2000; Edwards, 2001). At the same time, lurkers are full community members (as opposed to outsiders), since they share some kind of identity, communication, repertoires and a sense of engagement with the other project members (Zhang & Storck, 2003).

Organization & coordination. Prior work in a third field of interest on open source projects has focused on their organizational structure and it seeks to answer how the globally dispersed contributors achieve unity of effort of their individually executed work. Prior work on open source communities has strongly argued for the importance of understanding what leads to integration in this setting (e.g. Lerner & Tirole, 2001; von Hippel & von Krogh, 2003), which is one of the major differences between commercial and open source software production (Mockus et al., 2002), and which could therefore be one critical reason for the success of open source projects. Work in this area of interest has
therefore sought to find an answer on how coordination takes place in the absence of traditional organizing mechanisms that are employed by firms or by the market, such as property-rights, clear agency-relationships, or the price mechanism.

Prior work has found that open source projects constitute a new model of organizational governance, grounded in voluntarily contributing workers (Markus et al., 2000), and although the communities seem to lack a high degree of formal organization (Tuomi, 2000), open source projects appear to be surprisingly disciplined (Markus et al., 2000). The communities’ disciplined character has been traced back to a variety of informal coordinating tools and elements, which may substitute formal coordination. Primarily, authors have referred to a unique culture that is effective in the total of all open source contributors, and to the project's task modularization as coordination instruments in open source projects.

Prior work has identified, firstly, a unique culture across the entire community of developers to all existing open source projects as one coordination mechanism that is effective in this setting. Authors have termed it 'hacker culture' (Bonaccorsi & Rossi, 2003; Markus et al., 2000; Moon & Sproull, 2000), ‘rational culture’ (Yamauchi et al., 2000), ‘Unix culture’ (Tuomi, 2000), ‘gift culture’ (Raymond, 1999; Zeitlyn, 2003), or ‘ideology’ (Stewart & Gosain, 2003). They have all assumed that members of the open source community share a common set of norms, beliefs, and values that all contributors know, and which they are supposed to follow.

Norms, values and beliefs of the hacker culture refer to the following aspects: (1) The belief that open source software produces better code than software generated in firms, which is a result of faster bug fixing, the avoidance of forking, or of information sharing and of helping others in the community (Stewart & Gosain, 2003), (2) the use of a concurrent versioning system (CVS), (3) and the open disclosure of code and documents (Elliott & Scacchi, 2003), (4) a common language and a common notion of validity (Bonaccorsi & Rossi, 2003), (5) or self control and social control (Moon & Sproull, 2000). A shared hacker culture can additionally be characterized by abundance of resources rather than scarcity, since the contributors’ social status is determined by what one gives away (Raymond, 1999). Programmers who are committed to this culture have additionally
been found to make their behavior logically plausible to others, which leads to better decisions and to finding technologically superior solutions (Yamauchi et al., 2000).

Lakhani and Wolf’s finding (2003) that 83% of respondents to their survey that has been sent to SourceForge developers, have claimed to strongly or somewhat identify with the hacker community (Lakhani & Wolf, 2003) seems to support the assumption of a common community culture. Additionally, many members of the open source community have been found to be involved into several ongoing projects, yet not necessarily at the same time. Ghosh (2002; Ghosh et al., 2002) has found, for instance, that 72% of the developers who responded to a survey were involved in 1 to 5 projects at the time of being asked, and 28% of the developers were involved in 6 or more projects. As a consequence, the open source contributors should be fairly familiar with the community’s hacker culture.

However, in one of the first deductive studies on the open source subject, Stewart & Gosain (2003) tested for the claimed relationship between a shared hacker culture and organizational coordination, with the latter having been operationalized as organizational effectiveness. The hypothesized direct effects of community members’ adherence to open source values, norms and beliefs and effectiveness as a dependent variable did not find any support by the study’s results. However, some indirect influence by the mediator variable, trust between the community members, was confirmed by the study. These findings indicate that a shared culture in the total of all open source developers might not be capable of providing for coordination in a specific project. Open source projects are, therefore, too different with respect to their sizes, their development stages, project aims or success stories (see e.g. SourceForge, 2003), and the rules or norms of the unique hacker culture are too general to provide for orientation in a certain project’s daily progress.

Supporting this assumption, Shah (2003) has found that the aspects that constitute acceptable behavior within the entire community differs by project, and Mockus and colleagues (2002) found strong differences between the projects’ task interdependencies or the sizes of the projects’ modules by way of a comparison of the Apache and Mozilla projects. A unique hacker culture consequently seems to be capable of achieving integration and orientation between the members of all open source projects, and it may
serve as some sort of ‘entry ticket’ to a specific project, but it fails as a sole mechanism to coordinate a specific project. Research on coordination in open source projects should therefore dig deeper into the specifics of selected projects.\textsuperscript{4}

Authors have suggested \textit{code- & task modularization} as a second type of coordination mechanism prevailing in open source projects. Modularity characterizes the breaking up of a larger system into its distinct parts that can be designed or produced independently of the other parts, but which can still function as a whole (Baldwin \& Clark, 2003). Through a system’s modularity, developers located in distributed settings can be granted a maximum of freedom to work on one system at a time (Moody, 2001; Benkler, 2003). Besides that, modularity enables the comparison of various (competitive) solutions to a development problem, and finally for high-quality code development (Moon \& Sproull, 2000). Since open source developers have the option to pick only a small number of modules on which they wish to work, modularity leads to a reduction in the complexity involved with large projects such as, for instance, in the APACHE webserver project, where developers obtain a ‘code ownership’ of small fragments of the code they created or consistently maintained (Mockus et al., 2000 \& 2002).

The interpretation of open source projects as modular systems explains the benefits that result from splitting up tasks in a distributed setting. It does not, however, answer the question of how unity of effort between the modules can be achieved and how these modules are to be coordinated. Coordination seems to be reciprocally related to modularity, and the latter cannot fully explain how coordination is generated in a certain project. Coordination, therefore, rather characterizes a necessary second step after a system has been modularized, and its effectiveness may require some other mechanism to take place. Consequently, coordination in open source software development remains an unresolved topic.

Furthermore, prior work has suffered from fundamental deficiencies with regard to the empirical foundations or methodological proceedings. A striking part of empirical studies have focused on a few prominent, large and successful open source projects including the \textsc{gnome} project (Koch \& Schneider, 2000), the \textsc{apache} project (Shah, 2003; Lanzara \& Morner, 2003; Mockus et al., 2000; Lakhani \& von Hippel, 2002; von Hippel, 2000), or the \textsc{linux} kernel development (Hertel et al., 2003; Lanzara \& Morner, 2003;...
Ianacci, 2003; Shah, 2003; Lee & Cole, 2002; Weber, 2000; Tuomi, 2000, Dempsey et al., 1999). These studies are obvious examples of the potential inherent in the open source model, but an exclusive concentration on these few and outstanding projects suffers from a threefold bias.

First, Linux, Apache or GNOME are by no means average open source projects in terms of their success stories and sizes. For instance, Krishnamurthy (2002), in a study of 100 mature projects, found that the average number of contributors to open source projects is four, and that only few projects generate a lot of discussion on their mailing lists (see also Ghosh, 2002; Ghosh et al., 2002). We can assume that very large projects are subject to different challenges, processes or dynamics than are comparably smaller open source communities, Secondly, contributors to these exceptionally large and successful projects might become biased by way of a large number of studies, surveys, or interviews conducted on their communities. Furthermore, they might give answers that they could have previously learned from prior publications on their respective project. Finally, not only the contributors’ answers but also their behavior within a project could be subject to changes in favor of prior published work.

These large and successful projects are thus not representative of thousands of smaller communities, and exclusively concentrating on them could lead to some over-evaluation and over-generalization of a narrow part of open source development. While such a proceeding makes sense during the early days of establishing a new field of research, current work should move on and expand empirical studies to mid-sized or to smaller open source projects. This is necessary in order to build a foundation for the future comparison of open source projects and for a broader generalization of findings and propositions on this form of organizing.

In addition to that, other authors have grounded their findings on anecdotal evidence instead of on in-depth case studies (Benkler, 2003; von Hippel & von Krogh, 2003; Osterloh et al., 2003; Lerner & Tirole, 2000), or on projects that reflect no proper open source software development communities, but which are hosted by profit-oriented companies (Elliott & Scacchi, 2003; Zhang & Storck, 2003). Furthermore, these authors only concentrate on one special type of project contributor as, for instance, core developers (von Krogh et al., 2003b), project administrators (Stewart & Gosain, 2003), or
peripheral members/ lurkers (Nonnecke & Preece, 2000; Zhang & Storck, 2003). Other authors, in addition to that, ground their findings on no empirical data at all and remain entirely theoretical (Zeitlyn, 2003; Demili & Lecoq, 2002; Bonaccorsi & Rossi, 2003).

During an early stage of researching open source projects, however, it is necessary to refer to in-depth empirical data into the analysis to fully capture and focus on exactly those dimensions that are capable of appropriately illustrating the new phenomenon. Anecdotal evidence as opposed to in-depth studies might therefore be too superficial to grasp the relevant mechanisms that are in place. In order to gain a complete overview of a certain open source project it is therefore key to study its entirety of members, their roles and their relationships, instead of focusing on a small part of the community, and to provide a mere snapshot of their project activities.

Prior contributions to the newly emerging research stream of open source software development have in sum been able to justify why these projects exist and why some projects have proved to be successful against commercial software development in a highly competitive setting (external perspective). Prior work has also been able to define valuable elements of this young mode for organizing innovative work (internal perspective), and both perspectives helped sketch a rough picture of fundamental processes and the basic architecture of open source projects.

Prior work has not, however, been able to explain how coordination is generated in open source projects. A complete and coherent model of how open source communities coordinate their distributed efforts in absence of traditional mechanisms is still missing. We do not yet know if any substitute mechanism is in place at all, and if so, what it is, how it comes into existence, or how it evolves over time. Such an understanding is crucial to paint a complete picture of open source projects, as a special case of the community model of organizing, and which could be a promising new mode for the coordination of innovative work. Only by fully understanding the open source phenomenon and by uncovering the mechanisms, elements, causalities, and dynamics that underlie this organizational model, can crucial steps be made in the direction of, firstly, a theory of community (or, open source project) organization, and secondly, to the proactive design and management of the new model: “If open source software really does
pose a major challenge to the economics and the methods of commercial development, it is vital to understand and evaluate it” (Mockus et al., 2002: 311).

In order to fill the above outlined research gaps and to gain an insight into how coordination takes place in the open source setting, this work employs inductive grounded theorizing and it studies the single case of an innovative open source project. In the following chapter, I will give an overview of the methods that were employed for this study.
3. Methods

The literature review of the preceding chapters revealed that only little is known about coordination in new forms of organizing in general, and more specifically on how coordination is achieved in open source software development projects. Due to the subject’s novelty, I employed a research strategy based on data from an exploratory single case study and on analysis methods that follow the idea of positivist grounded theorizing (Glaser & Strauss, 1967; Strauss & Corbin, 1990). The following paragraphs will give a more detailed overview of the case study as a method of inquiry, of the case sampling, data sources and data collection, and of the data analysis.

3.1. Exploratory single case study design

For the purpose of the present study, I employed an exploratory, qualitative single case study method of inquiry based on an embedded design.

**Exploratory research.** The issue of coordination in open source projects is an anomalous phenomenon, which according to prior organization theory must not exist or which must at least not be successful. Existing theory may, however, not be capable of fully grasping and explaining the new phenomenon, since established concepts and frameworks are grounded in assumptions that may not be applicable to the open source setting, and they tend to pre-structure and pre-define the problem space in question. A new phenomenon may yet go beyond what we have already defined and it requires us to re-think established boundaries and theories. Seeing a new phenomenon through old glasses could consequently overlook actual new structures, concepts or causalities that are essential to fully capture the uncovered phenomenon to date (Christensen et al., 2002).

During the early stages of a new phenomenon’s discovery, *exploratory research* as opposed to deductive research, is therefore more appropriate to develop a deep understanding of the phenomenon, its elements and causalities (Glaser & Strauss, 1967) which constitute the building blocks of new, or improved theory (Christensen, 2002;
Sutton & Staw, 1995; Whetten, 1989). Since almost no prior knowledge exists from which this study on coordination in open source software development is able to depart, I adopted an inductive, theory-building research framework.

**Data types.** Coordination in the open source setting, on the one hand, is a complex phenomenon, since it refers to a social context into which multiple individuals are involved. On the other hand, open source projects are a dynamic, boundary-less setting in which coordination might be a temporary, changing task. These characteristics make coordination in open source projects a phenomenon that is probably located beneath the surface of clearly observable facts that can be captured at first sight. Purely quantitative data or a cross-sectional analysis of multiple open source projects are therefore not be suitable to mirror how coordination takes place in this setting.

Qualitative data are in essence richer and deeper than those produced by purely quantitative methods. They enable a research process of the phenomenon in question in its entire context, and they prove to be sensitive to grasp the phenomenon’s evolutionary character (Denzin & Lincoln, 2000; Marshall & Rossman, 1995). For the purposes of this study, I will primarily analyze qualitative data. In addition to that, quantitative data serve as a supplement to verify and validate the qualitative findings. All data refer to one entire year of analysis (2000), and all quantitative calculations were based on SPSS 11.0 and on EXCEL.

**Case study method.** For the purposes of this work, I focused on the case study method as an empirical approach, which “investigates a contemporary phenomenon within its real life context, especially when the boundaries between phenomenon and context are not clearly evident” (Yin, 1994: 13). The case study method is suitable for all situations in social sciences where little is known about a temporary phenomenon, and which are too complex to be captured by the design of another method, such as it is the case for this work.
The case study method aims at retaining the holistic and meaningful character of the phenomenon to be explored. It aims at generating constructs and variables, and stipulates causal links that are relevant and practically useful. The case study method is consequently the first necessary step for any further empirical investigation of the young issue of coordination in open source software development (Yin, 1994; Eisenhardt, 1989; Miles & Huberman, 1994). Due to the topic’s newness and complexity the study focuses on a single (pilot) case on the FREENET open source project (www.freenetproject.org), which serves as a frame of reference for future investigation of coordination in this setting.

**Case sampling.** The sampling of a case is crucial, since it defines the universe from which the evidence draws, it controls for extraneous variation and helps set the limits for generalizing the findings (Eisenhardt, 1989). In line with theoretical sampling reflections (Glaser & Strauss, 1967), I decided to sample the FREENET open source software development project due to a variety of reasons.

**Firstly,** FREENET was only rarely studied in prior work (e.g.: von Krogh et al., 2003 a&b) and it is a mid-sized project to which a total of 356 contributors were subscribed to in the year 2000. In addition to that, FREENET is an entirely innovative project of peer-to-peer software and it has no templates similar to other open source communities frequently studied (Oram, 2000) such as, for instance, LINUX where UNIX OS served as a technological template.

Although the FREENET contributors have an idea of what they want to reach with their work, they do not exactly know how they can get there. Since a great number of project contributors are involved in the collective efforts and no template exists of how they should or how they could work together, it is necessary that the project members somehow coordinate their work. It is likely that some sort of coordination is effective in their project that this study aims at revealing, On the one hand, these characteristics of the FREENET project are critical to answer the study’s first research question (How is coordination achieved?), on the other hand, and in contrast to many open source projects studied in prior work, FREENET reflects a unique example of an innovative, extremely
uncertain project that serves as a critical pilot case on how informal coordination in an innovative, ambiguous context takes place (Stake, 1995; Yin, 1994).

Secondly, FREENET is a young project. It was launched in late 1999 on the basis of the project founder’s masters thesis in computer sciences (Clarke, 1999), which he made public on the internet and to which Ian Clarke invited others to reflect upon, to discuss and to contribute to his ideas. For the purpose of this study, data were gathered to sketch FREENET’s first year of existence in 2000. At that time, the project was still young enough to trace back critical events that led to the establishment of some project coordination. At the same time, FREENET had already had enough time to evolve, to mature, and to foster the mechanisms in place. Both of these characteristics are crucial to answer the second set of research questions (How did coordination come into existence in the first place, how does it evolve?).

Finally, access to a field of study constitutes a critical feature for empirical research. In comparison to other communities, open source software development projects in general offer unique opportunities with regard to data access, since a great part of the underlying data is stored, documented and publicly available via the internet. Direct contact to a project’s developers as, for instance, through interviews, remains difficult. For the purposes of this study, first personal contacts to the FREENET contributors were established via email and then further maintained. With one core developer personal contacts could be maintained during the entire process of study. This contact made possible frequent informal discussions about the findings during all stages of the work, which further strengthens the findings’ construct- and internal validity.

In order to analyze how FREENET’s content and processes are coordinated, the project contributors are of central concern and they will serve as multiple units of analysis for this work. How the contributors create interdependencies and connections between their activities, their strategies, or between their conversations can indicate how coordination is generated in the project. At the same time, an embedded single case study design with multiple levels of analysis has assisted in concentrating the focus during the stages of data collection and data analysis that were filled with an extensive volume of data (Yin, 1994).
3.2. Data sources & collection methods

For the case study method of inquiry authors do encourage multiple sources of evidence (Charmaz, 2000; Yin, 1994; Eisenhardt, 1989; Stake, 2000), which enable for a triangulation of data (Yin, 1994), since they generate multiple perspectives on the object in study. Diversity of data types provides for a stronger substantiation of emerging concepts and ideas, and it improves the case’s explanatory power as well as its internal validity and construct validity (Eisenhardt, 1989). For the present study, I focused on the following sources of evidence: Telephone interviews with project contributors, an open questions survey, publicly accessible project data, and supplementary interviews with entrepreneurship and software industry experts.

Telephone interviews. Firstly, fourteen qualitative telephone interviews were conducted in two rounds with ten of Freenet’s total of 30 core developers. Telephone as opposed to personal interviews enabled the interviewees to remain widely anonymous which is, at the same time, one of the major characteristics and goals of the Freenet project. All interviewees were identified from the project’s official developer list that is accessible on the project website (www.freenetproject.org or freenet.sourceforge.net). The listing separates core developers from peripheral project members, and the former have direct access to the project’s code repository, the concurrent versioning system (CVS), while peripheral members have no direct access to it. The broad categorization of core versus non-core developers/peripheral members will be maintained for the remainder of this study.

Each interview took between one and two hours and was recorded and transcribed to facilitate data analysis. All interviews were semi-structured, with guidelines defining the overall structure and broad categories of interest. On the one hand, this procedure made it possible to remain as open and flexible as possible to inductively find out central as opposed to irrelevant categories. On the other hand, the interviewees’ answers to identical questions guarantee a suitable comparison of arguments to the Freenet contributors.
A first interview round was conducted between October, 2000, and January, 2001. During this stage, I aimed at achieving a general overview of the Freenet project aim and project evolution, of contributors to Freenet, and of the coordination subject in general. I asked, for instance, about the contributors’ personal backgrounds, about their decision-making processes, about their avenues of conflict resolution, communication, common procedures, or about how cohesion was achieved in the project. In a second interview round, preliminary insights were deepened, concretized and verified between March and November of 2001.

Open questions survey. One major result of the first and second interview rounds was that great differences between core developers and peripheral project members existed in Freenet. It became clear that it was not sufficient to exclusively focus on the project’s core developers as key informants. As a consequence, an open questions survey served as a second data source (and interview surrogate), which was sent to peripheral project members. In contrast to the project’s core developers, during the period studied, the peripheral members had at no point of time direct access to the Freenet code repository, the CVS. Corresponding with the idea of theoretical sampling, this multi-step procedure helped fill gaps in the previously collected interview data and to refine emerging theoretical ideas (Glaser & Strauss, 1967; Charmaz, 2000).

The target group encompassed all 326 peripheral members to Freenet in the year 2000, although they did not necessarily contribute simultaneously (the same day, month, etc.). I searched the email addresses of 154 (47%) randomly selected contributors out of the body of emails in the 2000 development email list. Next, I contacted every individual through a personalized email by briefly explaining the project and asking for their support. By opting for this method, I was able to respect the anonymity of their email addresses (e.g., obfuscation@...), thereby guaranteeing all contributors complete anonymity of their responses, if they wished for this to be the case. In addition to this, they could choose between being contacted by telephone, email, or online chat. Of the 154 addresses contacted, 69 (45%) proved to be invalid, leaving 85 valid contacts.
In a second step, all peripheral members who showed interest in the study were sent a second email to which an open questions’ file was attached into which they could directly type their answers and comments before they returned it. By and large, the file covered the same questions that were asked during the interviews with the FREENET core developers, although the questions asked were more focused on topics that proved to be critical. If the contacted peripheral members had not responded to the second email after 14-20 days, they were sent a carefully and politely formulated reminder email to which the survey file was attached again. 20 project members returned the survey and their answers to it, and one additional member gave his comments by means of an online chat. All responses ranged from 3 to 9 pages of comments. Most of the respondents showed great interest in the study and offered additional support either by phone or by email, if required. As soon as a presentable working paper was available, interested peripheral members were sent some preliminary findings for further validation. Listed in the appendix one can find the first contact email to peripheral project members, a list of all interview partners and respondents, interview guidelines, as well as the survey questions.

All contacts and responses took place between September and November of 2003, which represents an extended period since the time when the peripheral members submitted their contributions to FREENET in the year 2000. One might think, however, that their responses could suffer from a potential memory bias. Notwithstanding, this bias has was eliminated at its maximum by way of a variety of measurements,

(1) all respondents were explicitly asked only to provide answers to questions to which they could remember the situation in 2000, and leave out other questions where they couldn’t remember exactly, (2) Their comments proved to be extremely consistent over the participating contributors, which further validated their responses, (3) By typing in their comments into an open questions’ file, the peripheral members were under no time pressure and they had the chance to carefully recall their activities in 2000. If necessary, they have also been able to reflect on their responses and so avoid the danger of mistakes through spontaneous answers. Against this background, and in comparison to other studies on open source projects that have used a similar method (e.g. Stewart & Gosain, 2003: response rate 37.5%), a response rate of 24% is a satisfactory result. 
Archival data. Thirdly, publicly available documents and project member conversations that are stored in the Freenet development mailing list, served as additional, supporting sources of information. The most important publicly available sources included the Freenet project website with a frequently asked questions (FAQ) listing, the project founder’s master thesis (Clarke, 1999), which originally initiated the project, journal and newspaper interviews with (core) project members, a working paper on the project (Clarke et al., 2001), and the Freenet developer mailing list.

The Freenet developer mailing list archives the contributors’ public email conversations that are related to the technical project development. The discussion on the list centers on software releases, bug fixes or code patches. In the Freenet project, three other email lists are in use (user support, announcements, technical discussion), albeit less frequently and with less influence on the project’s coordination as compared to the development list. Between January 1st and December 27th, 2000, which was the time frame of analysis, 11,210 emails were submitted to the development mailing list. All emails were stored in a separate database that indicated the sender, time and date of sending, direct responses to the email and the body of the message. The database was able to simplify search, comparison and access to the developers’ discussion, and it allowed for a quantitative analysis of the stored data.10

In comparison to other conversation instruments that were used for the Freenet project, such as private email and IRC (Internet Relay Chat), the email list has first priority for the discussion of project-related topics. One core developer noticed that IRC “is even more informal compared to private email […] and also less committal, […] Discussions [on IRC] center around life, work and challenges that are often even not related to Freenet,” (Core Developer H). Discussions on IRC or in private emails, though, were of less importance for this study and I did not take them into further consideration.

Supplementary interviews. Due to the novelty of the open source phenomenon, 7 qualitative interviews with software industry and entrepreneurship experts served as a fourth and supporting data source for this study. Through these interviews, I was able to gain a more complete overview and impression of the differences and similarities between open
and closed source software development in general, on coordination in both contexts, and on the current or even future relevance of the topic in question.

The interviewees additionally discussed central questions and challenges inherent in the open source software development and described if, how, and when they used openly produced software or contributed to projects themselves. Interview partners were 2 consultants to entrepreneurial ventures, 3 CEOs, 1 COO and one software engineer, of which all were employed by new software ventures in Switzerland or in Germany. All interviews were semi-structured and took place in February 2002, either face-to-face, or by telephone. The interviews lasted between 1 to 3 hours and they were recorded and transcribed.

The interviewees agreed on the reliability, low costs, and independence of open source software from major commercial players, as well as on the phenomenon’s great significance and relevance for their own companies. All members of the software ventures have additionally reported to use open source software for their own purposes and they have stressed that they returned code and their experiences to the community whenever their time schedules allowed them to do so.

3.3. Data analysis in the tradition of grounded theorizing

Data analysis followed the principles of grounded theorizing, which traces back to Glaser and Strauss (1967). Their approach is purely inductive and starts from scratch, which is applicable to a setting for which only little prior knowledge exists, such as for the subject in question for this study. The process of data analysis defined its goal as gaining new insights on coordination in the FREENET open source project, and as coherently integrating these insights to new theory. The study’s output is a consistent model of how coordination in this setting is generated, how it comes into existence, and how it evolves over time. It is the first of its kind in this field of research. For this purpose, the grounded theorizing method provides a systematic approach to inductive data analysis, since it consists of a set of clear guidelines and steps from which to build explanatory frameworks (Charmaz, 2000) that will be outlined in the cases following.
For this study, data analysis went through several phases of discovery from unstructured towards structured patterns in the data. By means of this process, constructs, categories and dimensions continuously emerge from the data, as was similarly described in prior work (Glaser & Strauss, 1967; Strauss & Corbin, 1990; Eisenhardt, 1989; Christensen et al., 2002). The initial stage of data analysis started with a description of the FREENET case and it referred to central project members, to their (coordinated) cooperation, or to the project’s evolution over time.

To look beyond initial impressions, the analysis then aimed at generating and integrating concepts through the process of coding that “represents the operations by which data are broken down, conceptualized, and put back together in new ways. It is the central process by which theories are built from data” (Strauss & Corbin, 1990: 57). During this process the researcher searches for patterns in the data, which indicate concepts, categories, properties or dimensions. The uncovered regularities then become reconnected and put into a greater context to uncover causal conditions for the phenomena observed in the data11 (Glaser & Strauss, 1967; Strauss & Corbin, 1990).

Since it was obvious that coordination in the setting of the FREENET open source project was no familiar or traditionally used mechanism, I searched for categories that were not clear at first sight. The analysis correspondingly followed the guiding question ‘What lies beyond the obvious?’ looking rather for implicit as opposed to explicit categories. In order to achieve this, the analysis made use of the principle of constant comparison on a number of theoretical and/or empirical levels,

on a first level, a comparison of empirical data took place, since data were gained from multiple sources of evidence and collection methods. This step in the data analysis included the comparison of the views, actions or experiences of FREENET project members in terms of whether they felt that enough coordination was taking place in the project or not, and if so, where, how and between whom. The findings revealed that not all project members had the same perceptions of coordination in FREENET, and that there was a strong correlation between early project members and core developers who felt well coordinated, and between later entrants and peripheral members who felt poorly coordinated.
On a second level, the comparison constantly took place between and within emerging theoretical categories, including the FREENET member types (core developers vs. peripheral members; early vs. later project entrants), knowledge inherent to the project and to individual members (amount; much vs. little knowledge; types: project-specific vs. project-related knowledge; overall vision, technical standards, community rules), and time (project stage: early vs. advanced; time investment: great vs. small). These fundamental categories indicated that coordination in the FREENET setting is a knowledge-related and (evolutionary) process.

On a third level, the emerging theory was constantly compared to empirical data, which was possible as a consequence of an overlapping and longitudinal process of data collection, coding and analysis that took place between October, 2000, and December, 2003. The resulting iteration between inductive and deductive data analysis finally converged into a definition of the new construct ‘dominant knowledge’. All comparisons were carried through until theoretical saturation was reached, the point at which additional incremental learning from the empirical data is minimal, since the emerging phenomena had already been discovered (Glaser & Strauss, 1967).

On a final level, the resulting dominant knowledge construct was compared to concepts that were already described in prior literature to find similarities, overlapping instances, or conflicts between these constructs. This final step further enhanced the study’s internal validity and generalizability (Eisenhardt, 1989).

In the following I will present the case study analysis, case evidence and the findings to which the analysis has led, I will, firstly, present a rather descriptive first order analysis of the FREENET case, which is structured along the questions that guided this work. The first order analysis summarizes facts and relationships as observed in the available documents or as reported by the interviewed project members. Secondly, I will present a more generalizing second order analysis, which results from the comparisons within the FREENET case data. The second order analysis moves away from the specific context of the focal project and results in a sample of general constructs, dimensions, and their proposed causalities. Such procedure and presentation is in line with comparable inductive work (e.g. Gioia & Chittipeddi, 1991).
4. Case Study Freenet: First Order Findings

I will start with a brief overview of some FREENET project basics and then describe a number of processes that lead to some coordinated project output.

4.1. Freenet project basics

The FREENET project (www.freenetproject.org) started in 1999 on the conceptual base on the project originator’s masters thesis (Clarke, 1999), which he wrote at the University of Edinburgh. Ian Clarke published his ideas on Freshmeat.net, inviting others to contribute to the project. FRESHMEAT is an open source community website where people release and exchange software. The FREENET software is being published under the GNU GENERAL PUBLIC LICENSE (‘copyleft’), which allows for the free use, modification, and redistribution of the code, but not for its proprietary use. Copyleft ranges among the most restrictive open source licenses, since it guarantees for strong software protection (Stallman, 1999).

FREENET is a file publishing system in the tradition of a peer-to-peer network, in which each participating computer (node) has equivalent capabilities and responsibilities. A central server, such as required for the traditional client-server architecture, is not necessary. The project’s major goals are anonymity and efficiency. FREENET guarantees for anonymity, since each decentralized node in the network retains a copy of a published file before it passes the file to the next requesting node. As a consequence, it is difficult for the author or the requester to determine where a file is being stored, who authored it, or to remove the file from the network to practice censorship. FREENET achieves efficiency as the system ‘learns’ over time where to locate files or where to search for the requests (Clarke, 1999; Clarke et al., 2002; Clarke et al., 2000; Freenet, 2000).

In the year 2000, a total of 356 individuals were subscribed to the developer mailing list (von Krogh et al., 2003b), and for the purpose of this study their subscription to the list serves as a ‘knockout criterion’ to be considered as a project member (contributor) or not. 326 (91.6%) of the contributing individuals were peripheral project
members who have no direct access to the project’s code repository, the CVS, whereas 30 (8.4%) individuals ranged as core developers with direct CVS access.

Project members are located all over the world and most of the individuals had never any personal or telephone contacts to any other project contributor. Some members reveal their true names; others remain in entire anonymity and contribute by using a nickname (e.g., thelema), or an anonymous email address (e.g., Ferrariman356@...). The project exists exclusively in a virtual context, proving traditional informal coordination mechanisms that require frequent personal contacts, such as organizational culture (e.g., Smircich, 1983; Schein, 1992), cognitive approaches (Prahalad & Bettis, 1986; Bettis & Prahalad, 1994; von Krogh & Roos, 1996), or routines (March & Simon, 1958; Nelson & Winter, 1982), to be an invalid option to coordinate Freenet.

Freenet can, at the same time, be considered as active open source project with considerable traffic. In 2000, a total of 11,210 email messages, summarized in 1,714 message threads, appeared on the development mailing list, with an average of 6.5 messages per thread (sd, 10.4) and an average weekly message traffic of 211 (sd, 90.61) and 32.34 (sd, 16.75) new message threads. In 2000, a total of 1,248 code commits were added to the code repository, with an average of 24 (sd, 18) commits per week. By December 2000, the commits added to an accumulated number of 1,800 lines of code, with 54,000 lines of code modified over the year (von Krogh et al., 2003b). However, the large number of project members and the extensive traffic require that coordination must somehow take place in Freenet to bring about some coherent output, which was simultaneously confirmed by individual project members (see below in the following paragraphs).
4.2. Coordination through shared, project-specific knowledge

4.2.1. Coordination despite the lack of traditional mechanisms

FREENET is the first software project of its kind, with no project architecture paralleling FREENET’s, which gives the project a unique and innovative nature (Oram, 2000). The code developed for FREENET is almost exclusively new and it could not be imported from other projects or code bases. As a consequence, the FREENET developers cannot exactly say what the final software product will look like, nor do they exactly know how to get to their nebulous end, since the code is permanently changing. One core developer remarked,

“We are doing something completely new. So we really do not have much of a roadmap to go after. We do not really know what comes next. [...] As for I know there is absolutely nothing like it anywhere else. And I think that one could call it innovative. That difference definitely gives new challenges compared to something that is well understood.”
(Core developer B)

Since the FREENET software is a network, not just an independent program, compatibility issues play a crucial role and code changes may have multiple effects. The program’s very core (its node) is inter-tangled and any options to clearly modularize the code that enabled the individuals to work independently on an isolated piece of code are very limited. Modularization requires clear-cut code interfaces, which were still lacking during FREENET’s early stages in 2000, since the coding of software interfaces is a fairly unpopular task. Close coordination among the project members’ individual efforts is thus required to bring about a coherent project output,

“The node itself is so complicated and, unfortunately, so inter-tangled that it’s not easily modularized and to do development on Freenet requires an understanding of fully everything that happens in the Freenet node.” (Core developer C)

Due to an open source project’s lack of centralized property rights and due to its voluntarily character, the project contributors cannot refer to traditional formal coordination mechanisms we know from organizations on the market–firm continuum,
and which could give hints on who does what, when, or how. Even if such mechanisms existed, planning or a rigorous structuring of the FREENET project would not make any sense due to the project’s evolving, uncertain and permanently changing nature. In line with these theoretical reflections, project members reported on lacking or on ill-suited traditional coordination mechanisms,

“There was no organizational structure, no hierarchy, and little planning.”
(Peripheral member #7)

“We don’t use formal docs at my job, so I had only vague ideas of what management docs even looked like. The management books I was trying to read seemed increasingly ill-suited for a small project staffed entirely by remote volunteers.” (Peripheral member #20)

Despite the lack of traditional formal or informal coordination mechanisms, FREENET appears not to be chaotic or disorganized. The project’s success indicates, instead, that the project members are capable of bringing about some coherent, widely acknowledged software output. Firstly, since its inception in late 1999, more than 1.5 million users downloaded the software and so demonstrated their interest in the project. Secondly, compared to the vast majority of open source projects (Krishnamurthy, 2000; Ghosh, 2002), FREENET could rapidly attract a large number of project members, who serve as a potential body of code contributors. Finally, since its initiation, the FREENET project has enjoyed considerable media attention (e.g. Oram, 2000; Koman, 2000; Kahney, 2000; Lipschultz, 2000; Cave, 2000), which indicates a great public interest in the project.

4.2.2. Core developers and peripheral members

Nevertheless, coordination does not cover all project members. The empirical data revealed that two different types of developers are prevailing in FREENET. One type of contributor feels that the project is well coordinated and these contributors know what to work on, and they know how to do it. They describe coordination in FREENET as ‘organic’ or as ‘self-organizing.’ As a result, their work becomes well integrated into the project. This group comprises in essence the project’s core developers and early project contributors. Also, the core developers are directly working on the project’s node, which cannot entirely be modularized and it is exactly here, where coordination is required most,
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Peripheral members (less coordinated)</th>
<th>Core developers (well coordinated)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coordination in Freenet:</strong></td>
<td>“I found coordination rather loose, To me it was at times not clear who was really in charge.” (Peripheral member #5)</td>
<td>“How do we coordinate it - I guess it’s a very organic process,” (Core developer A)</td>
</tr>
<tr>
<td><em>Feel or feel not coordinated, know what, when, how to do</em></td>
<td>“The only problem is that you have little control over what will actually get done, but this is no one’s fault but your own because if you really wanted to you could do it yourself,” (Peripheral member #10)</td>
<td>“I guess the whole thing self-organizes;” (Core developer A)</td>
</tr>
<tr>
<td></td>
<td>“Despite my lack of understanding [about what people were doing and how they were doing it], I never felt that it was the developers’ fault.” (Peripheral member #15)</td>
<td>“This structure is a kind of natural thing that evolved,” (Core developer A)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“I think we would work just as well together as a physical group than spread over the internet, But it just does not matter…” (Core developer C)</td>
</tr>
<tr>
<td><strong>Knowledge about project-specific rules in general</strong></td>
<td>“I don’t know what Freenet’s rules are;” (Peripheral member #1)</td>
<td>“All of these things are obvious ways to do things […] this is proven by the fact that numerous people independently say this is the way we should do things…” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“I can’t think of any rules or norms that guide Freenet, but I can point the Apache project’s rules regarding voting…” (Peripheral member #10)</td>
<td>“A lot of the things that have been influencing Freenet are things that we kind of knew – in a sense everyone knows that we need to have some-thing like this,” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“I don’t think there were any rules of behavior, written or unwritten, early on in the project,” (Peripheral member #6)</td>
<td>“I think one has to recognize that one simply should not do certain things,” (Core developer H)</td>
</tr>
<tr>
<td></td>
<td>“Not that I noticed [any specific rules or norms which guided Freenet]. Pay attention, speak reasonably, and you’ll garner the respect of your peers,” (Peripheral member #2)</td>
<td>“There are rules and opinions on two different levels, One is the generally acknowledged rules and opinions of the entire [Freenet] community. And then, everyone has his own private opinions;” (Core developer H)</td>
</tr>
<tr>
<td></td>
<td>“The only rule that I can think of is that you value the goals of Freenet,” (Peripheral member #6)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>“I wasn’t involved long enough to know [any rules],” (Peripheral member #9)</td>
<td></td>
</tr>
<tr>
<td><strong>Integration of project members’ efforts into the project</strong></td>
<td>“Basically, I wrote a proposal about how to link Freenet to the CCQ system, sent other emails around this idea, took part in discussions around a perspec-tive Freenet client (proposing my idea), I ultimately wrote a ‘Freenet wiki’ but I’m pretty sure that wasn’t used by anyone, I occasionally wrote general emails about Freenet client interface questions, I also wrote a long text about ‘Freenet and Xanadu’ which also wasn’t used or published anywhere,” (Peripheral member #4)</td>
<td>“Core Developer X is the easy example and I would say maybe four or five other people have fundamentally changed parts of the design,” (Core developer C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“There are people like core developer Y and they are working most with the client, And core developer C and I have been doing the core stuff,” (Core developer B)</td>
</tr>
<tr>
<td>“The other contributors generally evidenced little interest in my ideas. Perhaps they were correct considering how difficult the simple implementation was, I don’t believe that the Freenet project contains a single line of code written by me.” (Peripheral member #4)</td>
<td>“There are generally around four who are doing most of the development work.” (Core developer A)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Core developers, peripheral project members and coordination in Freenet: Case evidence.

For the second type of contributor the opposite is true. They feel poorly coordinated and some individuals cannot say how, or if at all, coordination in FREENET takes place. They feel poorly informed about what is going on in the project, or they do not know what to work on in detail. Although they suggest code patches to the FREENET code base, their committed work is often rejected or even ignored by their colleagues. Additionally, they they have no clue as to the reasons of their dismissal, causing some contributors even to give up their efforts and dropping out of the FREENET project. This type of contributor encompasses, by and large, the project’s peripheral members and later project entrants (table 2.1).

4.2.3. Coordination through taken-for-granted, shared knowledge

What is the reason for these differences in behavior between the well-coordinated core developers and the less coordinated peripheral members? The case data revealed that shared knowledge about project-specific rules and standards plays a central role as a discriminating variable in this process and it implicitly guides the contributors’ efforts. Along with Nonaka, I will interpret knowledge as “justified true belief” (1994: 15), and refer to both tacit and explicit (Polanyi, 1966) types of knowledge. Members of both the peripheral project group and the core developer group report the following.

Peripheral project members and people who only recently joined the project report that they know these project-specific rules only in part, or that they do not know them at all. Peripheral project members in general have either a very broad understanding of the project, but not on FREENET’s very details, and they consequently do not know enough to complete a certain piece of work that could coherently fit into the project’s whole.
Alternatively, they have a detailed understanding of a very small part of the project and focus on rather isolated tasks, such as the command line client, or a makefile, but they do not understand the project's overall causalities or relationships and they are not capable of locating their contributions in the code base.

Core developers and early project contributors, instead, share a strong agreement that is often taken for granted on informal, project-specific rules. FREENET's core developers agree on the project's overall orientation and focus, on the project content and on how work processes within the developer community should take place. On behalf of the agreement on these project-specific rules and standards, they know what to strive for, what to do, how to do it, and how to behave with respect to their fellow project members, which finally results in a somewhat coordinated project output. They refer to a shared project-specific knowledge base of how to complete their work, which is taken for granted and never subject to any controversial discussions or being questioned at all.

4.2.4. Three types of knowledge

The core developers refer to three different types of knowledge that guide their activities. Firstly, they refer to behavioral rules and processes within the project community (how do the developers work together?). Secondly, they refer to the project's overall direction, strategy and vision (where should the journey lead to?), and thirdly, they also refer to the project's technical standards (what is worked on, how, and how well?). I will give examples of all three knowledge types below. Peripheral project members, in contrast, have difficulties in specifying FREENET's rules and processes at all.

Community rules. Informal rules guide the project members' behavior and communication within FREENET as a first type of knowledge. This knowledge guides how the developers work with each other in a virtual and dispersed setting, and in settings lacking formal guidelines or structures. These rules are recalled differently by core developers and by peripheral members.
The well coordinated core developers report, for instance, on specific roles, which the members of this group take up by themselves, and they can also discern which other roles their colleagues fulfill. One core developer, for example, works almost exclusively on a Windows installer for the Freenet software (core developer H), and a second member of the core group is a recognized cryptography expert (core developer X). The core developers term their specific roles ‘specialization’, although nobody explicitly determined who does what, when, or how. Members of the less coordinated peripheral contributor group report instead that they do not take over any special roles within Freenet. Additionally, they cannot describe any roles that other developers take up, since such roles do not appear obvious to them. Some individuals name Ian Clarke, the project founder, as an outstanding spokesperson; they thereby refer to explicit and obvious knowledge within the context of the project. Subtler, unspoken roles in Freenet, however, are not visible to the members of the peripheral group.

As a second example for informal knowledge about behavioral rules within the Freenet community, the members of the core developer group illustrate how they know what to work on. They describe a process of self-selection in order to distribute their coding tasks, which no central authority guides or coordinates. Their behavior reflects that the contributors, who volunteer their time, talent and effort, would not accept any centralized, authority-driven plans or task distribution. It is obvious to them who selects what kind of work according to their specific preferences, their individual knowledge and according to the availability of tasks. They identify tasks by themselves and announce their efforts on the developer mailing list either before or after the tasks’ completion. The group of peripheral Freenet contributors, in contrast, reports on difficulties in identifying tasks that fit into the project architecture. They complain about the lack of feedback from their fellow contributors, or if some feedback is given, they do often not understand the message.

As a third example, the Freenet core developers report on an informal hierarchy, which is never made explicit, but which nonetheless prevails in the project. The core developers expect all project contributors to respect the hierarchy and to behave in accordance with their relative rankings, which requires the contributors both, firstly, to recognize the very existence of hierarchy, and secondly, to locate themselves within the actual ranking. Those individuals who have the deepest understanding of the project,
who work most, or who joined the project in a very early stage of development, seem to occupy the highest ranks, which is obvious to the core developers. Individuals who are at the top of the hierarchy have a final say in decision-making, and they profit from a sort of immunity when acting, which means that for the identical activities they become less criticized compared to lower ranked individuals. The following body of an email submitted to the development mailing list on August 24, 2000 illustrates how seriously the contributors expect every project member to follow the informal rules that guide Freenet:

“Peripheral member #35, you are a man leading two lives. In one life, you work at a respectable software company, pay your bills on time, and help your landlady. Take out her garbage. In your other life, you go under the hacker pseudonym ‘ peripheral member #35’. You have committed virtually every newbie computer crime we have a law for. One of these lives has a future... and the other does not. Time for you to make a choice... do you leave this list peacefully, or do we pack it?” (Poster to the Freenet development mailing list)

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Periphery members (less coordinated)</th>
<th>Core developers (well coordinated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(No) Recognition of roles &amp; specialization in Freenet; (no) specific roles taken over</td>
<td>“I don't know any of the developers.” (Peripheral member #1)</td>
<td>“I think we recently have specialized.” (Core developer B)</td>
</tr>
<tr>
<td></td>
<td>“No specific roles, The only major role I noticed was Ian's, as spokesperson, Other than that, most others involved were very flexible.” (Peripheral member #6)</td>
<td>“Core developer H is responsible for the Windows installer.” (Core developer C)</td>
</tr>
<tr>
<td></td>
<td>“I had no role in the project other than to bother people with questions.” (Peripheral member #17)</td>
<td>“Core developer B is pretty much the only one who is making structural and architectural changes.” (Core developer H)</td>
</tr>
<tr>
<td></td>
<td>“I just read a bit about it and posted a few messages, I never got really involved.” (Peripheral member #13)</td>
<td>“Core developer X is unique, [...] He’s such an expert in the area and he never says anything except for in that topic [cryptography].” (Core developer C)</td>
</tr>
<tr>
<td>(No) Recognition of task distribution and self-selection</td>
<td>“I have found that most of my suggestions and contributions tend to not get any feedback, so it is frustrating at times to see your thoughts fall on deaf ears. It may be in fact that they are silly suggestions but some negative feedback after some moments of true consideration would be more satisfying than none at all.” (Peripheral member #21)</td>
<td>“The code is publicly available, People do not have to ask, they just check it out and see if there’s something they can do, Or, we will get them on the [mailing] list and start listening to what people are talking about until they find something that they can do. Because of that we’ve been criticized a couple of times, I think there has been enough clarity.” (Core developer B)</td>
</tr>
</tbody>
</table>
"I left the mailing lists after I realized I had nothing tangible to contribute to the project, I felt I was wasting my time correcting obvious misunderstandings of computers, software, or politics."  
(*Peripheral member #7*)

"When something interests me I just go off and start coding it."  
(*Core developer C*)

"It gets back to the idea that people are to work on what they’re interested in."  
(*Core developer A*)

"There is good self selection,"  
(*Core developer A*)

"Since it’s a self selecting process, the people who are involved tend to do more."  
(*Core developer J*)

"It’s like ‘I’ll do this and I’ll do this’. Basically people throw ideas onto the mailing lists and then people say ‘I’ll do that, I’ll do that’. You’ve got these tasks floating around and then people take tasks that they want to do,"  
(*Core developer A*)

<table>
<thead>
<tr>
<th>(No) Recognition of some sort of hierarchy in Freenet;</th>
</tr>
</thead>
<tbody>
<tr>
<td>(No) recognition of what it takes to become a core developer</td>
</tr>
</tbody>
</table>

"I don’t think it was a hierarchy in Freenet, except maybe Ian having final say.”  
(*Peripheral member #4*)

"No hierarchy, but everyone has always respected others’ code contributions, especially when it comes to making changes to others’ code.”  
(*Peripheral member #6*)

"I didn’t find much [of a hierarchy in Freenet].”  
(*Peripheral member #2*)

"Ian was the top.”  
(*Peripheral member #3*)

"Can’t tell.”  
(*Peripheral member #1*)

"No hierarchy, but some who were in the centre, others not.”  
(*Peripheral member #5*)

"I had sometimes the feeling that some were kept out from core. But this is just a feeling.”  
(*Peripheral member #5*)

"I saw some people struggling to get recognized as developers.”  
(*Peripheral member #10*)

"There is something of a social pecking order.”  
(*Core developer H*)

"There is no need for a democracy, because there is not that many people really involved.”  
(*Core developer B*)

"It’s an entirely informal hierarchy. It’s never anything laid down explicitly.”  
(*Core developer H*)

"There are people whose voice counts more than others’.”  
(*Core developer H*)

"No one really decided it [who is in the core group or not]. It just became who understood, because there is a group of core people who understand everything about Freenet and write code for it and know how to change the software. Then there is a group of people who […] may not understand the code completely.”  
(*Core developer C*)

"Some other implicit rules would be not to make sweeping changes as a new developer […] that would be a grave faux pas.”  
(*Core developer J*)

Figure 2.2: Knowledge about behavioral community rules: Case evidence.
The core developers are additionally aware of what it takes to become a core developer with direct access to the FREENET code repository. They know that it is essential to demonstrate a deep understanding of the project, and to make a specific contribution to the code base (von Krogh et al., 2003b). Members of the peripheral contributor group, in contrast, have difficulty describing the informal hierarchy at all. Again, some peripheral project contributors are able to name the (obvious) project founder Ian Clarke at the top of the project, but they cannot refer to any other higher-ranked individuals. Peripheral members can additionally not reason what it takes to become a core developer. For case evidence and sample citations, see table 2.2. I summarize the preceding discussion in the following proposition:

P1: Shared, implicit and explicit project-specific knowledge about behavioral community rules leads to coordination in open source software development.

Technical standards. A second type of unspoken knowledge about project-specific rules are technical standards, which guide FREENET's coding process, and which are different for every specific project. This type of knowledge represents rules and requirements that define what is worked on, how, and how well the work should be done in FREENET, as the following examples will illustrate.

A broad and unspoken agreement exists among FREENET's core developers on what needs to be done in the coding process, such as project priorities or alternative coding solutions which could be integrated into the code repository ranges. What needs to be done is not explained to individual contributors, but is rather seen as self-explanatory. The core developers report to agree to a large extent on most of the important pieces of code without the need for any controversial discussions. The coding solutions seem obvious to them, as they have a broad overview of the code, its programming standards, and its structure and architecture. Their work fits, though, well into the project's code base.

Many peripheral members, in contrast, tend not to complete any coding work at all. Even if they would like to provide code for the project, they do not know what to work on in detail, or to which standards their work should strive. Partly, peripheral
contributors suggest code patches to the Freenet code repository, but their work is mostly rejected or ignored, and the peripheral members cannot explain why this is the case. Some members of the peripheral group reported that they dropped out of the project as a consequence of such experiences, which were frustrating to them or, they left Freenet, since they had the feeling that they could not contribute any value to the project.

Members of the core developer group agree, secondly, on standards that separate a good programmer from a mediocre programmer. Although they have difficulties in explicitly defining an outstanding programmer, the core developers describe standards and rules that indicate the individuals' programming capabilities, such as that the code must be readable, understandable, well-organized and well-documented in the code repository. In Freenet, all developers are implicitly asked to provide for the respective high-quality software code in order to maintain the project's high standards. Developers who stick to these rules get a lot of good reputation and their fellow project members tend to listen to them. Although these rules are effective in many open source projects or in software development as a whole, their concrete realization differs across projects and organizations. One contributor recalled,

"Every project has its own standards for how to submit patches, the way source formatted and/or documented, etc., the way questions are discussed, and so forth."

(Peripheral member #16)

Peripheral project members, instead, mostly do not recognize outstanding developers in the project at all. Only few core developers are capable or willing to fulfill the high technical standards prevailing in the project. As a consequence, only a small circle of core developers emerges, consisting of members who do most of the coding work in Freenet with respect to quality (the most important work) and quantity (most of the work). They are familiar with the code and could have worked in any area of the project. For case evidence see table 2.3. Summing up I conclude with proposition 2:

P2: Shared, implicit and explicit project-specific knowledge about technical standards leads to coordination in open source software development.
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Peripheral members (less coordinated)</th>
<th>Core Developers (well coordinated)</th>
</tr>
</thead>
</table>
| Knowledge about project-specific rules; Technical standards | “If you’re not familiar with the code, so that you’re replicating stuff unnecessarily, then you’re doing something that doesn’t fit in with the rest of the code, and the work that you do isn’t going to be useful.” *(Core developer J)*  
“The design patterns are different in all software projects.” *(Core developer H)* |                                                                                                                                                                                                                              |
| (No) recognition of a good programmer       | “It is usually quite simple to see one [a good programmer] on the code, Good programming has many aspects, But if you are working together, one of the most important aspects is that the code is readable and understandable.” *(Core developer B)*  
“It rapidly becomes apparent who knows what they’re talking about and who doesn’t.” *(Core developer A)*  
“Good code has to be well and clearly programmed, So if others read this, they have to understand the piece of code that was committed, And in that case it has to be documented in a really good manner,” *(Core developer H)* |                                                                                                                                                                                                                              |
| (No) recognition of what to code            | **“There is always some level of technical struggle about what to do next or who will do something,”** *(Peripheral member #10)*  
“I’m still hazy on a few details – for example, are TCP connections kept alive for a while after transferring a file, or is the connection closed and the public key kept, or is a new connection established…” *(Peripheral member #17)*  
“I aimed at producing a client that ran on top of the basic Freenet architecture, I offered this as a possibility and developed a sample client based on a program I had developed separately, Freenet was far too slow for such an approach, I don’t know whether this was a problem with the architecture or with my approach,” *(Peripheral member #4)* | “It was sort of fairly obvious that this [a technical issue] was kind of inefficient, […] When I think about the history of the project, the major events that I think about are pieces of code that are being implemented and most of the time they were fairly uncontroversial in a sense.” *(Core developer I)*  
“We haven’t had many surprises [in the code] that have said much of anything that has confused us or looked worse than we thought.” *(Core developer F)*  
“All of these ideas [technical proposals] are sort of obvious ways to do things, […] I think this is proven by the fact that numerous people independently say this is the way we should do things,” *(Core developer I)* |
“I was working on a C++ client at one time for Freenet, I stopped working on it after about a month or two on the project. This was because there were other clients coming out that were far more along with in development and worked decently, [...] Seeing these much farther ahead and with me having limited time to play catch up, I stopped the development, This was basically a survival of the fittest idea.” (Peripheral member #3)

“Most of the times we have problems there isn’t really a big argument, [...] I think we’re lucky in that respect.” (Core developer C)

“A lot of the things that have been influencing Freenet are the sort of things that we kind of knew, in a sense that everyone knows that we need to have something like this.” (Core developer I)

Only few do coding work

“I did very little, Another developer and I were about starting on a GUI interface for Freenet in C or C++, Development on the core protocol was moving so fast that just the two of us couldn’t rewrite it quickly enough to catch up.” (Peripheral member #3)

“All of the core developers were familiar with the code and could have worked on any area.” (Peripheral member #17)

“In fact I would say that 99% of the code was still written by core developer A, myself and core developer B.” (Core developer C)

“What was interesting, and what is still interesting is that while the kind of ideal open source development would be that you have tens or even hundreds of people all signing on to work on a project, what actually happens, or what happened in Freenet, is that a small number of 2 or 3 people emerge who work [on the code]. And the other people are very much spectators.” (Core developer A)

“I think probably 3 or 4 of them do almost all of the [coding] work.” (Core developer A)

Table 23: Knowledge about technical standards; Case evidence.

**Overall direction.** Knowledge about FREENET’s overall direction, vision and the informal rules connected to it characterize a third type of specific knowledge with respect to the project. The fundamental ideas and goals that the project aims at achieving, were initially laid down in Ian Clarke’s master thesis. The work gave rise to the FREENET project’s initiation and it serves as a very broad frame of reference for the contributors’ efforts. Since the FREENET subject and aims are essentially innovative, many aspects of the project remain unspoken, since they cannot easily be put into words. Although at the time of initiation, the project’s aim and mission were clear to Ian Clarke, he could not clearly articulate his ideas in order to make his vision understandable to others.
Moreover, Ian Clarke had difficulties structuring his own thoughts, which becomes clear in his description of the initial time of Freenet,

“I was saying ‘My God, this is such an obvious thing to do, [...] In fact, if you were to go to that website [where I posted the project in the first place] you may even be able to find the posting just explaining, probably not explaining well, because at that time I didn’t really refine my ability to explain what I was trying to achieve, but I invited people to read my thesis...’” (Ian Clarke, project founder)

Ian Clarke’s initial ideas and thoughts are well recognized by the Freenet core developers, and implicitly, his ideas provide for enough guidance to the members of this group to contribute to the ongoing project. At the same time, Ian’s vision and the overall project direction are open enough to give the core developers enough freedom to search for and to find project specifications that fit into the challenges that emerge from the ongoing and innovative process of Freenet’s progress, Members of the core developer group share a clear understanding of where the Freenet journey will lead to in the longer run. Implicitly, the contributors are asked to deliver work that is in line with Ian’s initial ideas and which is a path dependent extension of Ian’s early philosophical goals or of the project architecture and design. Work other than that becomes ignored and marks a contributor for future (negative) judgment. Freenet’s overall direction, however, is unspecific in nature, since it gives a rather negative definition of what the project aims not to be rather than specifying exactly what Freenet is. Additionally, over time the project permanently evolves through modification or progress of the contributors’ work.

Members of the peripheral contributor group, in contrast, interpret Freenet’s overall direction as ‘ad-hoc’ and ‘unclear’. They report to know best the explicitly stated parts of the project’s initial direction such as ‘anonymity’ or ‘efficiency’. Some peripheral contributors even violate the project’s most fundamental goals, since they are not familiar with Freenet’s vision, One of Freenet’s very core ideas, for instance, is not to make files permanently disposable in a permanent storage, This is necessary to guarantee for anonymity and for freedom of speech in Freenet, two of the project’s key goals. Nevertheless, the core developers report on peripheral contributors who continuously suggest the coding of a permanent storage, which comes close to what core developer J terms a ‘kiss of death’,

105
“The ‘kiss of death’ for Freenet development participation is saying ‘Freenet needs to have permanent storage.’ They’ll sometimes make a very eloquent, well-designed pitch for how to do this, why to do it, etc. By crossing that line that has been crossed so many times, they unknowingly mark themselves as taboo. [...] It’s definitely the main way to get kicked. I don’t mean this person is physically shown the door, that they’re dropped from the mailing list or anything, but it’s a good way to get the cold shoulder from the rest of the development group. Your ideas won’t be as well accepted, if you spend a lot of time restating design issues.” (Core developer J)

For case evidence and sample citations see table 24. I summarize the proceeding discussion in the following proposition:

P3: Shared, implicit and explicit project-specific knowledge about the overall project direction leads to coordination in open source software development,

<table>
<thead>
<tr>
<th>Characteristics (No) recognition of an overall direction/ vision</th>
<th>Peripheral members (less coordinated)</th>
<th>Core Developers (well coordinated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>“With Freenet there was an overriding sense that privacy and anonymity had to be maintained at the cost of almost everything else in the project.” (Peripheral member #15)</td>
<td>“In following the idea of Freenet we all have an idea about what it should be and how it should work, [...] We have something of a unified vision so that it’s going to be like, but not entirely. I would sometimes code stuff up as existence proof, or to show people that something could be useful. A lot of times that is just pushing my vision of what Freenet is. A lot of times the places I’ve gone to, as far as what I want to code, have been related to what my vision is for Freenet, [...] I’ll go and code stuff like that to push my opinion of how things ought to be.” (Core developer F)</td>
<td></td>
</tr>
<tr>
<td>“The real main explicit rule is anonymity, it is of utmost importance in this project.” (Peripheral member #19)</td>
<td>“...Ad-hoc structure and direction.” (Peripheral member #10)</td>
<td>“I think everybody respects this vision, because everybody who discusses Freenet on a regular basis is amazed by this vision, [...] They follow the same vision,” (Core developer H)</td>
</tr>
<tr>
<td>“Ian’s ideas for the original project were the main guidelines. That was for the core code. Past that anyone could do what they want to extend that idea.” (Peripheral member #3)</td>
<td>“Once in a while, I do notice that some nuggets of my suggestions does seep into the direction that the core devs take, I am never sure whether that was someone coming up with the thought independently or whether I had any influence.” (Peripheral member #21)</td>
<td></td>
</tr>
</tbody>
</table>
Path dependent development in line with the overall direction

<table>
<thead>
<tr>
<th>Path dependent development in line with the overall direction</th>
<th>Knowledge about the overall project direction: Case evidence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Much as I dislike vision statements, I have to admit that this seemed to work, My thinking, as well as the occasional discussions with others, started to be much better focused. In retrospect, what worked about this goal was not so much what it included, but what it forbade, [...] That by itself cut out a lot of the grand digressions, and kept us on track.” (Peripheral member #20)</td>
<td>“Most of the things Ian wrote initially are unchanged, but there’s a lot more now. He was pretty unspecific about a lot of things, like ‘Oh well, I have this closeness metric for keys,’ and didn’t really describe how that would work, I’m not if everything he said is still true, but probably most of it,” (Core developer F)</td>
</tr>
<tr>
<td>“This overall vision is still prevailing and in Freenet it becomes never questioned at all, Still, that is fairly easy to achieve, as it is a very broad vision.” (Core developer H)</td>
<td>“The basic architecture of Freenet has basically changed not at all, since Ian first began the project,” (Core developer I)</td>
</tr>
<tr>
<td>“So Freenet is primarily concerned with anonymity, and we’ll sacrifice anything else to make sure that the anonymity is available.” (Core developer F)</td>
<td></td>
</tr>
</tbody>
</table>
4.2.5. Project-specific knowledge versus project-related knowledge

It is striking that the knowledge for understanding what is going on in Freenet, as well as what individuals could do, or where they could locate their efforts in the community is project-specific as opposed to project-related knowledge. Project-specific knowledge emerges directly out of the context of a certain project, and it characterizes justified true beliefs on the ‘what’ (content) and ‘why’ (causalties), and of what is going on in the project. Although the general content of this knowledge might be valid for various open source projects or for software development as a whole, its concrete realization differs between projects and organizations. One peripheral contributor illustrated the difference between project-related and project-specific knowledge as follows:

“All open source projects I participated in had their own iconoclastic rules. Some have strict code layout guidelines, others have many levels of review before you can get your code in, and others are anarchistic heaps. Some use IRC as the main method of discussion between the devs, some meet in person, some use mailing lists, some use yahoo groups (although this last group strikes me as slightly unprofessional), Some favor correctness, some favor efficiency, Some common rules: Before you ask, check the web archives of the mailing lists, Before you code, ask if it’s already been done, or if someone else is working on it, Before you commit, make sure it works.” (Peripheral member #1)

Project-related knowledge, which is indirectly linked to a certain open source project, plays only a minor role as discriminating variable between well coordinated and less coordinated project members. This finding becomes clear through the peripheral contributors’ and the core developers’ comparable levels of project-related knowledge about computer sciences and about software engineering, on programming languages and about the open source philosophy in general (‘hacker culture’).

In the group of peripheral Freenet members, who are less coordinated, all replying individuals reported to have fundamental knowledge about software engineering. All but one have an academic degree in computer sciences, software engineering or in a related field (mathematics, electrical engineering, etc.). Alternatively and/or additionally, they followed a professional career in this field, working as programmer, IT manager,
database technician, systems administrator, etc. Some peripheral members claimed that they have worked and played with software since their early youth. Only one of 20 individuals had no profound knowledge about programming languages, the remainder of this group being familiar with several, mostly between 5 and 10 different languages. One peripheral contributor answered,

“I know the programming languages C, C++, Lisp/Scheme, Python, SQL, Prolog, ML, Matlab, Maple, bash, M4, Basic (commodore and quickbasic), Pascal (Wirth’s version), Modula-Z. […] I’m sure I’m forgetting some languages.” (Peripheral member #1)

All peripheral project contributors reported that they already had this knowledge before they entered FREENET. 16 out of the replying 20 peripheral contributors additionally reported that they contributed to other open source projects besides FREENET, their involvement ranging from initiating their own project to more moderate contributions. All peripheral project members were attracted by FREENET’s technical challenges, its philosophy or the core idea of anonymity and freedom of speech by the use of the developed software, or simply because they wanted to contribute to any open source project. All reported that they had learned about FREENET on Slashdot.org, mostly in early to mid 2000. The members of this less coordinated group are assumed to be familiar with the fundamentals of programming and with the overall open source philosophy and its netiquettes.

For the well coordinated group of interviewed core developers, the record is comparable to that of the peripheral members’ with all of them studying or working in the described fields of computer sciences (or in related natural sciences), all knowing several programming languages and all but one having gained experiences with other open source projects.

Project-related knowledge has, thus, not been able to help the members of the peripheral group gain understanding of the project in its entirety, and it is not able to coordinate their contributions. It is too broad to create detailed understanding of the evolving, innovative, and unique characteristics of FREENET. Therefore, project-related knowledge can rather be interpreted as an ‘entry ticket’ for contributing to any open
source project, since it provides the necessary but not sufficient prerequisites to do so, which is also reflected in two developers’ remarks,

“Without a decent understanding of computer sciences, the project wouldn’t have made much sense.” (Peripheral member #16)

“I think that understanding programming languages and software engineering were useful in understanding Freenet at a very small, localized level. I.e. understanding how it was programmed and some of the common things it did – but not necessarily the overall structure or purpose of the Freenet system. So, common knowledge will help you understand WHAT it is doing, but not WHY you’d want it to do it.”
(Peripheral member #10)

I conclude and summarize the different influences of project-specific as opposed to project-related knowledge in the following proposition:

**P4:** *Project-specific,* shared implicit and explicit knowledge leads to stronger coordination in open source software development than *project-related* implicit and explicit knowledge.

Why do these differences between the two contributor types and accordingly between their work integration exist in FREENET? The subsequent chapter will answer this question and it will provide insights into how the shared, project-specific knowledge in the FREENET project came to existence in the first place, how it was established, how it dispersed across the developers, and how it evolves over time.
4.3. **Evolution of coordinating knowledge**

The case of the young FREENET project makes tracing back the evolution of project-specific coordinating knowledge possible, which is, at the same time, an inherent reason for the differences between the two contributor types and for their (un-)coordinated behavior. The FREENET contributors reported that coordinating project-specific knowledge comes into existence through processes of *knowledge creation, capturing & replication, and transfer*. Taken together, these processes lead to a project-specific knowledge base that is capable of coordinating the contributors’ individual efforts, Core developers and peripheral project members additionally reported their differing involvements in these knowledge processes. As a result, they share different amounts and types of project-specific knowledge. In the following paragraphs I will provide a descriptive overview of these knowledge processes, and conclude a number of propositions that were derived from these insights.

4.3.1. **Individual creation of coordinating knowledge**

Within FREENET, members of the *core developer* group who are simultaneously also *early project contributors* can recall or reconstruct possible explanations of the rules prevailing in the project, and *why* these rules and standards exist, why they make sense in the context of the project and how they evolved. *Peripheral project members*, who are simultaneously *later project entrants*, in contrast, have no comparable explanations of the project’s specifics in terms of its overall direction, technical standards, or behavioral community rules. Members of this contributor group are (partly, if at all) aware of *what* the rules say, but they are not aware of *why* the prevailing rules make sense in the context of FREENET.

Since FREENET is an entirely innovative open source project, new project-specific knowledge becomes permanently created and added to the existing body of knowledge due to continuous adaptations. Well coordinated core developers and less coordinated peripheral project members differ, though, with respect to their involvement in the process of knowledge creation. On average, the members of these two contributor groups have entered FREENET at different points of time and therefore during diverse project
development stages; and as a result, core developers and peripheral project members have been involved in the creation of different types of project-specific knowledge, and they also own different amounts of it.

Figure 2.5: Freenet peripheral members’ entry to the project by means of submission of their first email to the developer mailing list.

Figure 2.6: Freenet core developers’ entry to the project by means of submission of their first email to the developer mailing list.
Points of time of entry to the project. The differences in knowledge about project-specifics between core developers and peripheral project members trace back to the early stages of the project's inception, and to the types of project-specific knowledge that has been created during different project stages. Members of the core developer group on average entered the project on day 109.83 (April 18th, 2000; sd. 87.64) past Freenet’s initiation by way of submission of their first email to the development mailing list. Members of the peripheral contributor group, in comparison, did so 85 days later, on average on day 194.3 (July 14th, 2000; sd. 88.11), which characterizes an entry that is about twice as ‘late’ compared to that of the core developer group members (figure 2.5 and 2.6). Members of the two contributor groups have, therefore, been able to participate in the creation of, firstly, different amounts, and secondly, different types of project-specific knowledge.

Amounts of created knowledge. Furthermore, the well-coordinated core developer group and the less coordinated peripheral member group differ with respect to how much project-specific knowledge they have created, as the following indicators (number of email contributions, thread initiation, CVS commits, general project documentation) will demonstrate. Core developers have generally created more knowledge in Freenet than members of the peripheral group.

In the year 2000, the 30 core developers contributed a total of 6,751 (60.22%) emails to the developer mailing list, which is on average 225 emails per core developer over the year. Together, the core developers initiated 61,08% (1,047) of the total number of 1,714 email threads, which is on average 34.9 threads initiated per core developer (von Krogh et al., 2003b).14 The 326 peripheral members, in contrast, contributed 4,459 emails (39.78%) to the developer list in total, with an average of 13.68 emails per person. The peripheral members initiated a total of 667 (38.92%) threads on the list, with on average 2.05 threads per peripheral member in 2000. Taken together, the core developers on average have more than 16 times as many emails posted on the list as the peripheral members, and they initiated more than 17 times as many threads compared to the peripheral contributors.
In 2000, an overall of 1,244 commits to the project’s code repository have been added (von Krogh et al., 2003b). Since only core developers have direct CVS access, the members of this group made all commits. Peripheral members, who have no direct CVS access, are still able to indirectly commit code over the core developers. In the open questions survey, 11 of the 20 peripheral members surveyed reported on no coding work at all, 4 peripheral members reported on little coding work, and of the remaining 5 peripheral members, the answers were unclear. These findings indicate that most of the coding work is completed by the core developers.

Similar things can be said about the core developers’ and peripheral members’ creation of general project documents. On the project website of the year 2000, I could exclusively find members of the core developer group who were originators or authors of publicly available documents (for instance, FAQ, project presentations, papers, Ian Clarke’s master thesis), describing the project’s aim, philosophy or work. Peripheral project members were not involved in the generation or publication of general project documents at all.

**Types of created knowledge.** The well-coordinated core developer group and the less coordinated peripheral project members reported, thirdly, of different types of project-specific knowledge about FREENET’s rules and standards which they have created. Members of the core developer group rather reported about the creation of path-creating, and architectural project-specific knowledge, while peripheral project members referred to the creation of path-dependent, content-, or component-related knowledge15. During the project’s very early days, FREENET’s basic ideas and how the ideas related to each other were created and laid down in the project architecture by the early project members and core developers as a path-creating foundation for the project’s overall direction, technical standards, and community rules. Their work pre-structured the project for all contributing project members as well as all other members entering at a later date, as the following examples will demonstrate.

During the early project stages, Ian Clarke laid down FREENET’s overall direction in his master thesis, although he faced difficulties in clearly structuring his thoughts (as described in the preceding paragraphs). From a perspective of technological project
standards, Ian Clarke decided, for instance, during this time, that software development in Freenet was to be based on the programming language Java, a decision that was never discussed or put into question by the project contributors. Some of the later submitting (core) members even decided to learn Java in order to be able to contribute to Freenet, and have as a result accepted Ian’s early decision. Freenet’s early days have similarly been decisive in establishing fundamental rules of behavior within the project community. Early project contributors found, for instance, that for Freenet a formal, purely democratic decision-making process in the form of a voting scheme did not make any sense. A precondition for such a decision-making process is that every project member is similarly aware of the topics to decide upon, and that he or she is familiar with all the discussion surrounding the topic in question. For Freenet, these preconditions have not been fulfilled, as one core developer recalls,

“An example of that would be people were saying we should have a way of storing permanent data on Freenet. What the core developers immediately realized is that it is impossible in terms of the current architecture, it would basically ruin it. Yet most people were, like, ‘this would really be a cool feature. We should do this.’ If we had a vote, the core developers would have been immediately outvoted which would have put us in a really silly position because it’s simply not possible to do this.” (Core developer A)

Since then, decisions in Freenet have been discussed on the mailing list, but the contributors do not formally decide by means of the mailing list. Instead, a handful of core developers have a strong voice on how things should be done in Freenet. Some difficult decisions are even being put into the future, since the early contributors have recognized that the time is not always ripe to decide on a certain topic at the time it occurs,

Peripheral members, in contrast, who have entered the project on average 85 days later compared to the core developers, reported of their minor path-dependent project contributions, such as for example encouraging the group to stick to Freenet’s very goals of anonymity and freedom of speech (peripheral member #15), discussion and documentation (peripheral member #10), or sending stories on situations where the software developed in Freenet could have been helpful (e.g. ladies’ freely exchanging dress design patterns over the internet, peripheral member #15). Their knowledge-
creating contributions therefore represent *path-dependent* extensions of the early project-specific ideas, rules and standards that properly fit into the already established project structure. Since the peripheral contributors lack a complete overview of the project’s rules and standards and of FreeNet’s full content, unlike the core developers, they are not able to create fundamentally new project paths that fit into the established project architecture. In order to contribute to the project, peripheral members need not, however, necessarily have a complete project overview; they only need to have some understanding of that part of the project to which they want to contribute.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Core developers &amp; Peripheral members</th>
</tr>
</thead>
</table>
| Knowledge evolution and first knowing advantage | “FreeNet has started with a concept that Ian Clarke has developed in his master thesis, Then, the project started and the first code was developed, FreeNet successively evolved with small patches being added, or some changes being made. So it is somehow a growing beast,” *(Core developer H)*  
“There are some conceptual hurdles to overcome to really understand FreeNet, Newer contributors often misunderstand some important aspects of the project,” *(Peripheral member #6)*  
“The older members also had a clearer idea of what the project wanted and had to keep telling the new members that ideas they had seen before wouldn’t work and why. Many times these topics would be pounded again and again on the discussion boards,” *(Peripheral member #3)*  
“I’d imagine the early contributors got more control over its [FreeNet] direction and design,” *(Peripheral member #8)*  
“I have no understanding of; Early stages of the project,” *(Peripheral member #11)*  
“No, I don't know either [two specific parts of the code], That was before me, I think that it’s very likely that Ian both came up with it and coded it,” *(Core developer C)*  
“I have no idea actually [who thought of it initially], I think it's a great idea, but I don't know who did it, It's been around ever since I've been on the project,” *(Core developer C)*  
“The source code is becoming increasingly complicated, Since FreeNet is not modularized, the individuals have difficulties to dig into the complex whole, if they hadn’t grown themselves with it and do not know why some things work, or how they work. It has become fairly difficult to learn the required knowledge. Secondly, in the beginning the FreeNet member group has been smaller and more familiar, There are still the same people discussing with each other, while the project itself has been growing enormously,” *(Core developer H)* |
\begin{table}[h]
\centering
\begin{tabular}{|l|p{12cm}|}
\hline
\textit{Individual knowledge creation} & \\
\hline
"Since I'm not really working with them, I don't really declare what I'm doing to people very much, I'll just do some stuff, then post it when I have something usable and make that available for people," (Core developer F) & \\
"The C client development is mostly solitary. Most of my interaction with other developers is - I'm picking through," (Core developer F) & \\
"In the case of core developer B, I would suspect that he probably just can't be bothered to explain to anyone else how it works, He'd rather do it himself," (Core developer A) & \\
"I had no role in the project other than to bother people with questions. As far as I could tell, all of the core developers were familiar with most of the code and could have worked on any area, The only exception was when a large new section of code was introduced by a single developer, who for a while would be the expert on that code," (Peripheral member #17) & \\
"When you're writing something all by yourself, you can organize everything the way you want, But, when other people come in and they move the files around a bit or they do some other things like that -- so, that was sort of -- kind of trivially annoying." (Core developer G) & \\
\hline
\end{tabular}
\caption{Evolution of coordinating knowledge, first knowing advantage and individual knowledge creation: Case evidence.}
\end{table}

Some peripheral project members additionally reported of their \textit{path-breaking} project contributions, and more concretely, of the creation of knowledge that could not be integrated into the project, since it was too different from what already existed. This type of knowledge does not create any fit with the previously established project-specific rules and cannot be integrated into the existing project structure, I summarize the prior discussion with the following propositions:

\begin{enumerate}
\item[\textbf{P5a:}] Knowledge about project-specific rules and standards that was created at an earlier (later) point in time during a project’s life cycle, that is of greater (smaller) amount, and \textit{path-creating} (\textit{path-dependent}) leads to more (less) coordination in open source software development.
\item[\textbf{P5b:}] The creation of \textit{path-creating} project-specific knowledge leads to stronger coordination in open source software development than the creation of \textit{path-dependent}, project-specific knowledge.
\item[\textbf{P5c:}] The creation of \textit{path-dependent}, project-specific knowledge is more likely to be integrated into the project’s knowledge base than the creation of \textit{path-breaking} project-specific knowledge.
\end{enumerate}
**Knowledge creation in individual isolation.** Although the core developers’ and peripheral project members’ involvement into FREENET’s knowledge creation process differ broadly, the members of both contributor types similarly report of their knowledge creation activities that were completed in individual isolation. New knowledge of project-specific rules and standards becomes consciously or unconsciously created and added whenever a specific situation requires the contributors to do so, since no valid prior solution exists. The new, innovative character of FREENET requires from its members, firstly, to think of emerging project-specific challenges of all kinds (overall direction, technical standards, community rules), until they find a viable solution by means of a trial-and-error process and until they are able to communicate it. Since FREENET is entirely innovative, the project contributors are dealing with very new subjects, which they themselves find difficult to externalize, to explain, or to communicate at all. They first need some time by themselves to clarify and structure their private thoughts. Knowledge creation in FREENET is more an individual rather than an organizational task.

Members of the core developer group, for instance, reported of their own individual code repositories that were stored on their personal computers, which they used to work and play by themselves. They reported, additionally, about newly created patches of code that they only presented to or discussed with their colleagues as soon as these patches appeared well enough to themselves. Code patches can be a more or less extensive piece of software. For additional case evidence, see table 2.7.

### 4.3.2. Capturing & replication of coordinating knowledge

On an individual level, the FREENET contributors reported of the re-use of their individually created, project-specific knowledge of all three types. In the case of positive experiences with their newly created knowledge, they use repeatedly as a template for their own recurring activities. Negative experiences with their created knowledge on an individual level, instead, tend to lead to give up the newly created knowledge. That which worked once on an individual level is repeated to such an extent in future comparable situations that this method is captured in the contributors’ personal and individual knowledge repositories. Through the multiple repetitions of previously
created solutions, the knowledge is internalized and taken for granted, ready to be recalled whenever it appears to be useful. Future activities can consequently build on a growing body of individual project-specific knowledge, which does not need to be questioned again and that reduces the complexity of situations in which the contributors are confronted with new decisions to a minimum.

The following example will illustrate the capturing and replication of individually created project-specific coordinating knowledge. The core developers reported, for instance, of situations in which they code extremely new and complicated pieces of software. Since very new code bears the danger of grave mistakes and of not compiling, they have learned that they cope best with the uncertain process by themselves instead of cooperating with their colleagues. They reported to submit code only as soon as it appears ripe enough to be presented to the other contributors. By means of trial-and-error learning, however, the core developers recognized that they were better able to cope with the development of complex new code patterns in isolation rather than through a premature discussion with their co-developers.

The contributors have thereby learned that a individually developed code is of higher quality compared to a jointly developed code. When creating new code, the developers are often not capable of expressing their very new ideas to their colleagues, since no prior template exists. During the early stages in the creative process, they are not able to clearly express what they are exactly doing, since coding has much to do with playing and feeling what could work. At the same time, their behavior is a safety mechanism, since a software code that does not work properly as soon as it is implemented into the entire network system can ruin both the software and a contributor's reputation. Finally, since Freenet is a networked system, every mistake can cause further bugs in the code, which makes properly developed software an essential aspect for the project.

The decision to employ such a coding style has never been explicitly discussed or decided upon within the Freenet community. The above-described coding style is, however, well accepted within the developer community, as the example of core developer B illustrates. Core developer B codes very extensive, new and important pieces of code by himself and he recalled,
“Often times if I am doing something that is a radical change, then I have to hold on to it myself for a while. Because I am changing and I don’t want to break it, it takes back to what is in the repository what is actually a working version.” (Core developer B)

Core developer B, at the same time, is famous for his outstanding coding work and several of the contributors during the interviews that were led or in the open questions survey have referred to him. One of the core contributors described him as follows,

“Core developer B is the programming god.” (Core developer H)

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Core Developers &amp; Peripheral Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual knowledge capturing &amp; replication</td>
<td>“As it stands now, White Rose is manic code, it does a lot of strange things that I’m just playing about with. It has its own use and threading, I think it’s quite nice, I’ll probably release that into a separate library, because that could be used with a lot of projects, I’ve never seen anyone do it for another research platform as well.” (Core developer D)</td>
</tr>
<tr>
<td></td>
<td>“Firstly, I have to work. The first thing is that it has to compile, we don’t want to put stuff in it that does not compile and that everyone gets that piece of code. So when we have been good in development, it often happens that we end up making changes that break the ability of the new versions to talk to the old versions, and then I have to hold back on these radical changes until we can say okay, there is an old version that people can use now during our experiments of the new version.” (Core developer B)</td>
</tr>
<tr>
<td></td>
<td>“I do have a code base, which I’m not going to release, because I really don’t want other people working on it. [...] I probably would have liked that [other people working with me on this code], but I don’t think the codes would have turned out so good. Somewhat paradoxically, I think the project would be worse off for it. [...] I think that if it had happened in the beginning, it wouldn’t have come out so well. Once other people are working on the code, it’s very difficult to say, ‘I want to scrap it all and start again.’ [...] As it has turned out, as the protocol dissolved, the designs I had initially - while good for the protocol as it was, now that the protocol’s changed, other designs are better. So, I would have probably stuck with the old bad design.” (Core developer D)</td>
</tr>
<tr>
<td></td>
<td>“Core developer B usually likes to do very few commit, he tries to just sort of sit on something for a long time and he’ll just make - kind of changes to everything and they he’ll commit an outline and he’ll put like a two-line comment on it, something like, ‘Prop-upper, it’ll be like the most major change you’ve ever seen on Freenet, but it doesn’t have the right added in it and it doesn’t [...] kind of a lot commit, [...] Now my style is that I like to commit more often, because I don’t want to have to deal with merging caution, because the risk is that the longer you hold on to a change without committing, the more likely it is that someone else may come and change, which conflicts with your change and you’ll have to resolve it.” (Core developer I)</td>
</tr>
</tbody>
</table>

Table 2.8: Individual knowledge capturing and replication: Case evidence.
The example illustrates how core developer B was continuously able to internalize his positive experience of coding by himself. He has learned and created a behavioral rule for himself, and since it worked, he has repeated his strategy in successive and comparable situations. The rule of coding in individual isolation has in a third step been transferred and it spread over the FREENET community to become organizational. I will illustrate this step in the next paragraphs, for further case evidence see table 2.8.

4.3.3. Transfer of coordinating knowledge

In the preceding paragraphs it was demonstrated that project-specific knowledge with respect to the rules and standards in FREENET was created, captured & replicated by individuals who work by themselves in isolation. To become organizational, the knowledge is in part additionally transferred to other project members. Since an open source project cannot rely on formal, institutionalized forms of knowledge transfer, the mechanisms in place tend to be rather subtle in nature and the well coordinated core developers and the less coordinated peripheral members differ again with respect to their knowledge transfer behaviors. In the following, I will give an overview on how coordinating knowledge is transferred in the FREENET project. More specifically, I will give an overview of the following, knowledge codification through various media, the contributors’ active and passive knowledge transfer strategies, of the differences between tacit and explicit knowledge, and between knowledge creators and knowledge absorbers.

Knowledge codification through various media. The FREENET project profits from a high degree of knowledge codification and externalization, since electronic devices are the primary means of communication, Electronic media store and accumulate the created software and surrounding discussion, and they are easily accessible to all interested contributors. The various communication media transfer primarily different (yet not exclusively, and to some degree overlapping) types of project-specific knowledge.

With respect to the overall project direction, the developers consistently agree on the political and philosophical goals and perspectives the project founder, Ian Clarke, reflected upon in his master thesis, which led to FREENET’s inception. Other core
developers contributed to the overall project direction by means of other publicly available general documents, e.g., the project FAQ (frequently asked questions), project papers or presentations. Since all of these documents are freely accessible via the internet, their knowledge content is easily transferable to those individuals interested in FREENET,

“I downloaded Ian’s thesis and decided that this was worth giving my effort to.”
(Core developer B)

“Well, it started out being just because Freenet was such a cool idea and it sounded like fun to work with it” (Core developer C)

Similarly, the contributors are expected to behave in accordance with the community’s rules of behavior that are (implicitly and explicitly) transferred by means of the discussions on the mailing list,

“There are some rules, points of views that developed throughout the course of the project, as they have been discussed on the mailing list frequently, […] those points of views and the regulations of the community.” (Core developer H)

With respect to the project’s technical standards, the major means of transferring the accumulated knowledge is FREENET’s source code, which contains all aspects of the previous coding work, its content, architecture and supporting comments made by the programmers. FREENET’s developers are asked to study the source code before they commit code, or before they submit their comments to the development mailing list (for additional case evidence see table 2.9),

“For new developers it’s a rather obscure project. One reason therefore is that no documentation of the source code exists. As a consequence, people who ask ‘how does this work?’ get the answer ‘read the fucking source code’. (Core developer H)
Table 29: Knowledge transfer media: Case evidence.

Active and passive knowledge transfer behavior. The Freenet project members reported on active and passive knowledge transfer behaviors. Firstly, the project contributors are implicitly expected to personally take over all the responsibility on their Freenet membership, which includes the responsibility for their own learning progress, for their individual knowledge or work. They are asked to actively search for, to learn about and to absorb the knowledge that is necessary to contribute to Freenet. Nobody would ever tell a project member what he or she ought to know in order to contribute to the project. The well coordinated core developers and the less coordinated peripheral project members differ with respect to their involvement in active searching strategies.

Members of the core developer group tend to make use of all three knowledge transfer media (mailing list, source code, general project documents); and they search for new project-specific knowledge of all three types (overall project direction, technical standards, and community rules). Although each dimension gives only incomplete information about the entire project, the core developers gain a complete understanding
of the project’s content and of causalities within or between the knowledge types and of the project architecture. At the same time, the contributors stressed that it is necessary to take the relevant knowledge from various sources and not to simply rely on a single source of knowledge,

“If someone wanted to participate in Freenet, I would advise him to read through the last three months’ email archives, and to inform himself on the project. Then, I would advise to read the initial concept of Freenet, Ian’s initial idea, his master thesis and the other few publications on Freenet. And finally, I would advise ‘Read the fucking source code’.” (Core developer H)

“A large part of the discussion on the Freenet lists was political, and to some extent the technical landscape was defined by political goals, so it was necessary not only to understand the technical side of the project but to keep its social goals in mind while discussing possible solutions.” (Peripheral member #17)

Peripheral members, in contrast, tend to limit their attention to less than all three types of project-specific knowledge, and they do not use all knowledge transfer media. Members of the peripheral group reported to primarily focus on reading publicly available documents, which indicate the overall project direction, and/ or they reported on frequently browsing the development mailing list that transfers behavioral community rules. Many peripheral contributors do not read the source code at all, and thus lacking the knowledge with respect to technical project standards. At the same time, the peripheral project members are not aware of their knowledge deficits and of their incomplete project overview. This discussion leads to conclude the following:

P6a: The use of more/ multiple knowledge transfer media (mailing list, general documents, source code) leads to stronger coordination in open source software development than the use of less/ only one knowledge transfer medium.

P6b: Project-specific knowledge about more/ multiple knowledge types (overall direction, technical standards, community rules) leads to stronger coordination in open source software development than knowledge about less/ one type of project-specific knowledge.
Secondly, the project’s core developers and knowledge creators consequently live and demonstrate the behavior that is desired in Freenet, the project’s aim and the relevant technological rules, and so they transfer the corresponding knowledge to their colleagues, who remain passive knowledge absorbers. To the core developers, the project’s aims of freedom of speech, anonymity and privacy, for instance, are most crucial. They make the aims’ importance clear through the frequent encryption of their emails, and they complain about their colleagues who do no do the same, even if the subject on the email is something entirely trivial. In parallel, some core developers permanently review the work of others, commenting on, correcting or even removing their contributions, and so they actively transfer knowledge about the overall project direction, the technology and about the community rules by means of their positive or negative feedback. Nevertheless, their behavior requires that new developers, in the first place, become active themselves, and that they provide some work or comments to which the core developers can give some feedback.

Active searching behavior and passive knowledge absorption both enable the transfer of knowledge and some overall project coordination in Freenet in an efficient way. Knowledge transfer in Freenet takes place only where and when it is necessary. It happens between those contributors who jointly work on an identical, specific project task. Consequently, the project contributors neither do own the same types nor the identical amounts of project-specific coordinating knowledge. Knowledge transfer and knowledge absorption are both time consuming and they require an invested effort from the individuals. These activities create costs for every individual contributor. The project members reported, for instance, on extensive email traffic with an average of 31.34 emails a day (sd, 21.7, in peak times up to 113 emails daily), which they had to browse to keep on track of the project’s progress. For further case evidence see table 2.10.
Tacit & explicit knowledge and knowledge creators & absorbers. Both types of project contributors reported of insufficient project documentation of all three project-specific knowledge types, and of their difficulties in capturing and learning the individually created project-specific knowledge. This finding indicates that knowledge creators do not make all of their individually and newly created project-specific knowledge explicit, since they cannot or wish not to do so. A more or less extensive part of the individually created knowledge remains implicit and stays with the creators. Their personal knowledge does not spread entirely over the internet, ready to be actively searched for by the remaining entirety of project contributors who wish to absorb the newly created knowledge.

To capture all of a project’s very intricacies, details and architecture, the project contributors must, therefore, be directly involved in the knowledge creation process themselves. Knowledge creators enjoy a knowledge advantage over knowledge absorbers,
which only increases with the rate at which a project member is engaged in knowledge creating activities (von Krogh et al., 2003b). Knowledge absorbers, in contrast, who have not been involved in the process of knowledge creation, must first absorb as much as possible of the existing, accumulated project-specific knowledge in order to catch up with the knowledge creators. One peripheral member notes,

“They [the early project contributors and knowledge creators] knew the system better from scratch instead of having a steep learning curve to begin with.” (Peripheral member #19)

At the same time, absorbing all of the previously created knowledge is a difficult if not impossible task, since the accumulated knowledge is to some degree tacit, stored in the creator’s head, and therefore not easily accessible. In addition to this, no proper project documentation exists, which provides further information about what can explicitly be found in the electronic devices. One core developer remarked,

“To some degree I could not understand the project in its entirety, even today.”
(Core developer H)

Since knowledge creators are primarily core developers, and knowledge absorbers are, to a large degree, peripheral project contributors, their differing behaviors again are able to explain the groups’ differences with respect to the project’s coordination. I do summarize the preceding discussion as follows:

P7: The creation of project-specific knowledge leads to stronger coordination in open source software development than the absorption of project-specific knowledge.

Knowledge that has been newly created, repeatedly tested or adjusted, and which has proved to be successful, becomes transferred to other project members. The created knowledge, thus, disperses across the project; it turns into shared knowledge and so becomes organizational. The continuous repetition and replication of knowledge leads to the establishment of an organizationally shared and unquestioned coordinating knowledge base. The subsequent second order analysis will further generalize these
findings in a general model of coordination in open source software development. For additional case evidence, see table 2.11.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Core Developers &amp; Peripheral Members</th>
</tr>
</thead>
</table>
| Knowledge transfer: Implicit and explicit knowledge, lack of documentation | “One of the biggest problems is that the Java code is all we have as far as documentation goes, A lot of that isn't the most readable code in the world. Most of my interaction with the other FreeNet developers lately has been picking through the Java code,” (Core developer F)  
“Core developer B went ahead and he wrote a whole bunch of code to do that, He just sort of committed it, And there wasn’t a whole lot of comment on it, at first, but it was a major change in the sense that lots and lots of lines of code got changed, It wasn’t an event in the sense of there being a lot of discussion about it,” (Core developer I)  
“At the moment, you can’t ask, ‘Give me a list of the files that are in FreeNet,’ You can only tell me in a sense, ‘This particular file is in FreeNet,’ So, it’s as though you were looking on your computer, and you weren’t able to open up any folders or anything, You can only say there is a file named FreeNet.pdf in this folder.” (Core developer I)  
“It’s really detail-oriented stuff, because there is no decent documentation,” (Core developer F)  
“The actual protocol was different enough that you couldn’t really grasp all of it from using it, As a user you imagine it as a blackboard; as a programmer you think of it as a peer-to-peer protocol,” (Peripheral member #8)  
“I suspect you can only gain a good understanding of FreeNet through the mailing list, it's easy to misunderstand how it works unless you can ask questions, There are various papers available but they sometimes contradict each other because the design is still changing, Summaries of how FreeNet works (in newspaper articles etc.) tend to go wrong as soon as they depart from directly quoting the developers,” (Peripheral member #17) |
| Knowledge transfer: Only where required              | “The mailing list is quite a high-traffic list, Sometimes it’s about a hundred mailings a day or so, So, you can’t really go through the archive if you are looking for some discussion or some decision.” (Core developer H)  
“On FreeNet it does seem that the list gets hundreds and hundreds of messages, So many messages all the time, I can’t really read it anymore.” (Core developer H)  
“People will commit when they think it’s good and usually it will stay in there except when it breaks stuff or when it makes stuff insecure, then people will start complaining,” (Core developer H)  
“After a long time on a mailing list you learn to recognize frequently asked questions (or frequently proposed solutions) and filter them out, so you can read the list more quickly,” (Peripheral member #17) |

Table 2.11: Evolution of coordinating knowledge: Case evidence,
5. Case Study Freenet: Second Order Findings

In the following, I will summarize the findings and propositions of the preceding chapters in a generalized framework that abstracts from the Freenet case sample. The following paragraphs will, firstly, refer to knowledge as coordination mechanism, and secondly, to its evolution over time.

5.1. Defining dominant knowledge as coordination mechanism

The empirical case of the Freenet open source software development project illustrates how multiple and distributed individuals are capable of collectively generating coherent output despite the absence of any traditional formal coordination mechanisms. In the setting under question, unity of effort is achieved by means of shared and organization-specific knowledge about rules and standards, which substitutes for traditional, formal coordination mechanisms. Therefore, it is knowledge that coordinates not the rules and standards per se. The rules prevail in the project, but only if members know them, can the individuals behave in accordance with the rules and standards⁵⁸.

Shared, organization-specific knowledge encompasses both implicit (stored in the contributors’ heads) and explicit (stored on the project website, mailing list, code repository) parts, and the organizational members take it for granted without fundamentally questioning the knowledge at any point of time. To serve as a coordinating mechanism, organization-specific knowledge needs to fulfill two necessary, but not sufficient conditions. It must, firstly, cover three distinctive dimensions and inform of an organization’s overall direction, about the work’s technical standards, and of the behavioral community rules. Secondly, organization-specific knowledge must be organizational instead of individual, and furthermore, it must remain repeatedly in use. Both conditions will be discussed in the following.
5.1.1. First condition: Three knowledge dimensions

As a first condition, knowledge employed to coordinate an organization must cover each of these distinctive and complementary dimensions. Firstly, knowledge of the organization’s overall direction, which provides for a vision or a strategy regarding where the collective work could lead in the longer run. Neither the concrete path of how to get there, nor the concrete final output must clearly be defined. Rather, it is enough to sketch a rough picture for the contributing individuals of what the organization aims at achieving.

Secondly, the shared organization-specific knowledge informs of the (technical) standards required for the product to be produced. It defines standards that distinguish between good and poor work, and it defines what should be worked on as well as how this should be done within a specific organization. The knowledge does not, however, cover the exact content or structure of the organizational members’ efforts, but it gives a frame of reference of what kind of work is adequate and therefore accepted in a certain organization. Finally, organization-specific knowledge covers informal rules of how the organizational members should behave in relation to one another within the organizational community, and how they are to work and to communicate with one another.

Although never made entirely explicit, knowledge with respect to these three dimensions spans a field of activity in which the overall organization is situated and in which it evolves over time. On the one hand, knowledge defines boundaries that are taken-for-granted, which give enough stability and orientation so that the organizational members can deliver work, and which are necessary to provide coherent output. By means of this mechanism, the total amount of complexity and uncertainty that is inherent in working on an innovative product can be reduced to a minimum and prevents the organizational members from becoming overwhelmed by a large number of working details. On the other hand, the organization’s boundaries need at no point of time be entirely defined. They can be subject to permanent and flexible adaptation whenever necessary, which is a prerequisite for creative, innovative work for which not all working details can be determined in advance. Knowledge that keeps the organization together
can consequently be modified in at least one of the above-mentioned dimensions at any point of time (see also figure 2.12).

![Diagram](image)

Figure 2.12: Three dimensions of dominant knowledge: Overall direction, technical standards, community rules and organization at two different points of time, t and t+1.

The three knowledge dimensions complement one another, and taken together, they are capable of sketching a rough picture of what and of how the organizational members should work. Every single one of the three knowledge dimensions is crucial for the organizational members to achieve a complete overview of the organization’s work and to understand how or why the organization’s parts fit into the given framework or if they fit at all. Consequently, knowledge of the three dimensions is enough to generate coordination of the organizational members’ individual efforts and of their pieces of work. Knowledge of less than all three dimensions, instead, cannot be sufficient to understand the organization in its entirety. An individual’s incomplete organization-specific knowledge, nevertheless, is sufficient to participate in the collective work. How much an individual knows (the number of dimensions and the accumulated amount of knowledge in each dimension) defines what kind of work he or she is capable of completing and integrating into the whole. Organizational members need, however, to
know the rules and standards of at least one knowledge dimension to be able to contribute to the collective work.

The necessity of all three knowledge dimensions is, in fact, comparable to the process of doing academic research. If every participating individual in a dispersed group of researchers is familiar with the collective research aim, which may be subject to changes (research aim and questions: overall direction), they all know the standards to which their output should aspire (e.g. inductive vs. deductive research, the rules of authentic statistical methods: technical standards). They also know how they should behave in relation to one another (e.g. contacts by email and once a week a telephone conference, being well prepared for the calls, etc.: community rules), and the researchers are thus able to work together cohesively on a new field of study. If one or several of the researchers are not familiar with one or several of their collective rules, extensive effort and time investment is necessary to achieve a minimum of mutual understanding in a greater effort to produce a coherent collective output.

5.1.2. Second condition: Organizational and repeatedly in use

As a second condition for knowledge to be considered as coordinating knowledge it has to be organizational and repeatedly in use. On the one hand, and in order to coordinate an organization, at least two people within the organization must share the piece of knowledge in question, which turns it into organizational instead of individual knowledge. Coordination then can take place, but between the individuals who share the same knowledge, but nowhere else. Shared, organization-specific knowledge is consequently not identical to the individual project members' personal knowledge, nor do all organizational members share the same amounts or contents of organization-specific knowledge.

The project members need only acquire the knowledge, which is necessary in order to fulfill their specific tasks in the organization. Since not all individuals in the organization fulfill identical functions, their personal knowledge bases might differ in terms of what and how much they know. Since knowledge creation and absorption are both time intensive and also require much effort, the fulfillment of these tasks produces
intangible costs for the corresponding organizational members. Every individual contributing voluntarily is free to decide for herself on how much she wants to invest in the collective work.

On the other hand, for knowledge to be considered a coordination mechanism, its use must take place repeatedly. For a single-time use of knowledge cannot provide for coordination in the longer run. Only its continuous repetition lets the organizational members take the knowledge for granted. I will clarify this aspect in more detail in the following paragraphs (5.2) on the evolution of dominant knowledge.

5.13. Dominant knowledge definition

I shall term a shared knowledge base that fulfills the above-mentioned conditions (three dimensions: Overall direction, technical standards, community rules, and it is organizational and repeatedly used), and which is able to coordinate an organization, ‘dominant knowledge,’ and define it as follows,

'Dominant knowledge' characterizes the accumulation of organization-specific tacit or explicit knowledge of prevailing rules and standards that is stored in an organization where it fulfills a coordinating role. It is as well organizational in nature (shared by several individuals) as used repeatedly, and it covers all of the following three dimensions: The overall direction of the organization (Where will the journey lead?), technical standards to be fulfilled (What is to worked on, how, and how well?) and behavioral rules of the community (How do the organizational members work with one another?). Dominant knowledge evolves during an organization’s life cycle through the processes of knowledge creation, capturing & replication and transfer.

Additionally, I summarize the preceding discussion in the following proposition:

P8: Dominant knowledge leads to organizational coordination in open source software development.
Given the appropriate information, the model on knowledge as a coordination mechanism (figure 2.12) allows for situating and comparing every individual member of a focal organization at a certain point in time and space, which is defined by the three knowledge dimensions. Additionally, it allows to trace back the focal organization’s and its members’ evolution over time. An organizational member’s accumulated knowledge is valuable only in relation to the accumulated knowledge of a focal organization. If an individual’s knowledge is situated outside of an organization’s three knowledge dimensions, no ‘fit’ and consequently no coordination and integration of his or her work into the greater whole can be achieved.

The great number of unspoken rules prevailing in the FREENET project enables the project members to focus on their efforts of the innovative and demanding development tasks, but it does not require them to concentrate on the project administration task, which represents a fairly unpopular task for software developers. Participation in the project requires the project contributors to know and understand the rules of the entire game that are taken for granted in all of the three knowledge dimensions (overall direction, technical standards, community rules), and to demonstrate their commitment by playing the game right. The shared organizational knowledge base plays an essential role as project-specific backbone, which ensures the participants’ coherent behavior and output without the need to be formalized. Compared to formal ‘law-like’ rules and standards, their informal counterparts are more flexible and can faster be adapted to an ever-changing situation in an innovative and only limitedly predictive setting, such as in the FREENET project.

How does the dominant knowledge come into existence in the first place, how does it evolve over time? I shall give an answer to these questions in the subsequent paragraphs.
5.2. Evolution of dominant knowledge

The FReeNet case study suggests that each dimension of coordinating knowledge (overall direction, technical standards, community rules) comes into existence through a multi-step evolutionary process of knowledge creation, capturing & replication, and transfer. Knowledge transcends two distinct boundaries, namely from an individual to an organizational level, and from a single-time use of knowledge to its repeated use. In addition, early organizational members profit from an advantage over members entering at a later point, as will be summarized in the following.

5.2.1. Multi-step evolutionary process

Coordinating dominant knowledge evolves through a multi-step process, In a first step, individual organizational members create new knowledge every time they are confronted with a new and challenging situation for which no established solution or template exists. Frequently occurring challenges force the organizational members to cope with the current situation, to reflect on it, and to search for and employ a viable solution. The contributors are involved in a multiple, often unconscious trial-and-error process to keep the workflow going and to advance the progress of their collective work. Taken together, all of the contributors’ activities and decisions determine the project’s location and boundaries with respect to the overall project direction, its technological standards and the behavioral community rules. By creating organization-specific knowledge, the contributors lay a foundation, which can serve as a template for future organizational activities, such as agenda setting or the project architecture. Moreover, they learn which kinds of created rules bring about justified behavior in the community.

Since many tasks in the innovation process are highly complex and uncertain, they cannot easily be put into words and be discussed with others. Working and playing in isolation thus enables individuals to search for and reflect on a viable solution to a specific problem. During the early stages of an organizational life cycle, previously generated knowledge of all three dimensions is only available to a limited extent; therefore, the organization’s members must create a lot of new and fundamental knowledge. During an organization’s initial stages of development, knowledge creation tends, therefore, to be path-creating, establishing the ‘roots’ of an organization. For this
reason, it can serve as a template and a foundation for this organization’s future activities. Knowledge that is created in later stages of the organizational evolution, in contrast, tends to be in line with the early decisions and it is, thus, path-dependent.

Individually created knowledge on its way to becoming organizational might follow two different paths, Firstly, through capturing and replication of the knowledge on an individual level and through the subsequent transfer to other organizational members (steps 2 and 3). Or secondly, through the immediate transfer from the individual to the organizational level (step 3), see also figure 2.13.

![Figure 2.13: Evolution of dominant knowledge.](image)

In a second step, newly created project-specific knowledge may become captured on an individual level in the individual members’ personal (often unconscious) knowledge repositories – in their heads and in their actions - for future replication. Knowledge that proves to be successful for single-time activities, becomes re-used in future recurring situations, and over time, the knowledge repeatedly used turns into internalized patterns of behavior, ready to be recalled without being questioned again. While positive experiences, which result from single-time activities, do lead to the reinforcement of newly created knowledge, negative experiences characterize dead ends, or trajectories that will be excluded from further consideration.
Over time, early organizational members possess a growing personal repository of project-specific knowledge, and the more they are engaged in project-related activities, the greater their personal repository of knowledge becomes. The more their stored pieces or patterns of knowledge become recalled, the stronger they are subject to individual internalization. The internalized knowledge does not need to be questioned every time it is used, and it enables the organizational members to focus on new and challenging tasks demanding their entire concentration.

In a third step, either newly created individual knowledge or individual routines, which prove to be successful and adequate within the context of a specific organization are transferred to an organizational level — and more specifically, to at least one other organizational member. Previously established templates are spread over the organization, which further strengthens the existing dominant knowledge. This knowledge is not fundamentally called into question, but it is seen as previously tested and consequently it is institutionalized. Knowledge absorbers are organizational members who either actively seek or who imitate the knowledge which is required to fulfill and understand a certain task.

In doing so, they demonstrate their willingness and motivation to contribute to the collective work. They can also act as passive knowledge absorbers of both positive and negative feedback from their colleagues. Both mechanisms lead to a wider distribution of the organization-specific knowledge across the organization, and at the same time, the mechanisms filter out those pieces of knowledge, which proved not to be valuable for the context in question. Taken together, the mechanism keeps the individuals’ investments in terms of time and effort low, thereby enabling a greater degree of organizational efficiency.20

In a final step, knowledge that was transferred to multiple organizational members turns into a dominant coordinating knowledge base, provided the organizational members frequently recall the established knowledge patterns or templates. The knowledge’s multiple recalling in various organizational locations further strengthens knowledge that was once created during the organization’s early days of existence. Knowledge that could survive an organization’s evolution while it was frequently being used and tested, finally results in it becoming dominant and thereby serving as a frame of
reference for future activities in the organization. In this way it also serves to coordinate the members’ joint efforts.

5.2.2. First knowing advantage

Since the process of building a coordinating dominant knowledge base is evolutionary in nature and cumulative over time, early contributors and knowledge creators benefit from an advantage over their later entering colleagues who are essentially knowledge absorbers. I will term their advantage ‘first-knowing advantage’. In character, the first knowing advantage resembles a first-mover advantage on an industry level of analysis as described by Lieberman & Montgomery (1988). The first knowing advantage is grounded upon three interrelated qualities, (1) the point in time during an organization’s life cycle; (2) the type of knowledge created; (3) the overall amounts of individual and organizational coordinating dominant knowledge.

Firstly, organizations go through different stages during their life cycles (e.g., Greiner, 1972; Mintzberg, 1984; Freeman, 1982; Gray & Ariss, 1985) with changing situations for their members. During an organization’s stage of inception, the organization must first be built up from scratch. This situation leaves most freedom for early entering organizational members to search for and to occupy distinct roles or niches, and to specialize in certain tasks. Early contributors must not fight to be known by the entire population of contributors, since they can hardly be ignored in a community that is still small in number. Later entrants, however, are confronted with growing difficulties in both finding an empty niche in the organization and in attracting their colleagues’ attention. The more an organization grows, the greater these hurdles become.

Secondly, early entrants have an advantage over later entering contributors due to the different types of knowledge created during different organizational stages. During an organization’s early stages of inception, fundamental organizational decisions must be met to nudge the organization’s growth and evolution. These decisions include an organization’s visionary ideas, behavioral rules, and technical concepts, as well as how these decisions all are interrelated. This reflects the creation of knowledge paths. Early entering organizational members are able to directly influence and control the
organization’s very direction and design and to adapt it to their personal ideas, interests, and capabilities.

Since they create the fundamental knowledge paths themselves, early project contributors have a deep understanding of both basic components and the components’ architecture (Henderson & Clark, 1990). As a consequence, they are easily able to fit new knowledge into the established architecture. Later entering organizational members, in contrast, must first learn, absorb and comprehend what has been created before, which is a challenging task, since not all of the created knowledge is explicitly available and easily accessible. Later entering organizational members have less influence on the organization’s very direction or on basic ideas. The knowledge they add is rather path-dependent than path-creating.

Finally, early organizational members benefit from the cumulative nature of their personal, individual amounts of (coordinating) knowledge and the growing dominant knowledge base. Given that early entrants continuously absorb the organization’s growing organizational dominant knowledge, at every point in time, they need only grasp small patches of what is going on in the organization’s evolution and invest only little time in order to achieve this. The more they know, the faster and easier they become capable of grasping newly added knowledge due to their growing absorptive capacity (Cohen & Levinthal, 1990).

Later entering organizational members, in contrast, must absorb all of the knowledge created before their entry in order to understand it in its entirety and to be coordinated, which is a time-consuming task. In comparison to the early entrants, they have only less absorptive capacity and they cannot capture new ideas as fast as early contributors can. I summarize these findings in the following proposition:

P9: Early organizational members and knowledge creators benefit from a first-knowing advantage over later entering organizational members and knowledge adopters in open source software development.

In the remaining chapters I will briefly discuss the dominant knowledge construct and compare it to existing theoretical concepts,
6. **What Dominant Knowledge is not**

Informal coordination mechanisms are not new to management and organization theory. Previously defined constructs that denote informal coordination mechanisms include shared values, symbols and rituals (*organizational culture*: e.g. Schein, 1992; Smircich, 1983; Rüegg-Stürm & Gritsch, 2001; Peters & Waterman, 1983), identical patterns of interpretation and shared mental models (*cognitive approaches*: e.g. Bettis & Prahalad, 1994; Weick & Roberts, 1993; von Krogh & Roos, 1996), or overlapping, interlinked activities and work processes (*organizational routines*: e.g. Nelson & Winter, 1982; March & Simon, 1958; Pentland & Rueter, 1994).

This study contributes two new findings to the ongoing discussion; *firstly*, the study suggests that some organizations can entirely rely on informal coordination without any formal mechanisms to be in place. *Secondly*, dominant knowledge, the mechanism that was found to informally coordinate the **Freenet** project, is partly overlapping, but unequal to any previously discussed concept, I will briefly discuss both aspects in the following.

*Firstly*, prior work has recognized the value of an informal coordination mechanism in organizations and authors have found that these instruments serve as a substitute to formal organization (Wilkens & Ouchi, 1983; Peters & Waterman, 1983), as a counter-productive mechanism (Wilkens & Ouchi, 1983), or as a complement to formal coordination instruments (Williamson, 1990; Barnard, 1938; Martinez & Jarillo, 1983; Rüegg-Stürm & Gritsch, 2001; Rüegg-Stürm, 2003; Poppo & Zenger, 2002). Prior work has, however, not been able to identify any informal coordination instrument that is effective as a sole mechanism for achieving coordination in organizations. Informal coordination instruments have, instead, been found to coordinate in combination with one or several formal mechanisms.

Therefore, the reasons might, *on the one hand*, be related to the fact that prior work has primarily focused on profit-oriented firms, but authors have only rarely considered voluntary organizations, such as open source development communities. Only recently have researchers recognized the innovativeness and market power of some prominent open source projects, such as **Linux**, **Perl** or **Apache**, which has led to a closer
investigation of these organizations. On the other hand, and in contrast to other forms of organizing, open source projects offer unique research settings, since they cannot employ any formal coordination instrument. The open source setting allows for us to focus our attention exclusively on informal coordination mechanisms, which might already have existed before but which have been overlooked in traditional research settings. It could, therefore, be the case that dominant knowledge is also in place in traditional organizations, yet with less intensity compared to the FREENET open source project.

Secondly, dominant knowledge, the informal coordination mechanisms that was found to be in place in the FREENET open source project, is overlapping with previously established constructs of informal coordination in organizations. All informal mechanisms thus have in common that they achieve unity of effort through the members’ informally shared, communal understandings of any types (Frost, 1997). At the same time, dominant knowledge is not identical to previously identified concepts of informal coordination, since it has some unique characteristics, In the following paragraphs, I will briefly describe both commonalities and differences to previously discovered concepts of informal coordination and refer to organizational culture, dominant logic, routines, architectural knowledge, and dominant design.

Organizational culture. Organizational culture has been introduced as “the accumulated shared learning of a given group, covering behavioral, emotional and cognitive elements of the group members’ total psychological functioning” (Schein, 1992: 10). The construct refers to an organization’s members’ shared values and rituals, their shared beliefs and to a shared meaning attributed to certain symbols and rituals (Schein, 1992; Smircich, 1983; Rüegg-Stürm & Gritsch, 2001). Similar to dominant knowledge, organizational culture can serve as an informal coordination instrument that substitutes formal structural mechanisms, and which is most efficient in organizations that are faced with high levels of uncertainty or where the members of the organization are supposed to closely cooperate for their work (Peters & Waterman, 1983; Wilkins & Ouchi, 1983). Organizational culture overlaps with the dominant knowledge’s ‘community rules’ dimension. The two constructs differ, though, with respect to the two remaining dominant knowledge dimensions (overall direction, technical standards). Similar to a dominant management logic, organizational culture tends to additionally be grounded in
people’s personal and subjective interpretations (Wollnik, 1993), while the dominant knowledge concept refers to a more objective reality that is actually being given.

In comparison to prior informal coordination mechanisms, dominant knowledge is, therefore, less deeply located into the individuals’ value and belief systems that is a crucial part of their personality. Dominant knowledge is, consequently, more superficial and can be easier adapted to changing conditions, since the deeper a person’s convictions are located in his or her value and belief system, the more resistant the individual will be to any change he or she faces. A person’s fundamental values, for instance, are more difficult to change than his or her beliefs (Kotter, 1996). Dominant knowledge should allow for more flexibility and adaptation than organizational culture or a dominant management logic, and it refers consequently to both stocks and flows (Decarolis, & Deeds, 1999; Dierckx & Cool, 1989) of organizational knowledge.

**Dominant logic.** A dominant general management logic is defined as “*a mind set or a world view or a conceptualization of the business and the administrative tools to accomplish goals and make decisions in that business. It is stored as a shared cognitive map (or set of schemas) among the dominant coalition.*” (Prahalad & Bettis, 1986: 491). A dominant management logic filters top managers’ attention towards what is already known, while other information is largely ignored (Bettis & Prahalad, 1994). Dominant logic, similar to other constructs on cognition,²¹ is a construct that denotes how managers interpret their world and what kind of information they tend to recognize in this process.

The newly introduced *dominant knowledge* construct does not stress how organizational members interpret their world, but it aims at focusing on rather objective, concrete knowledge about rules and standards. People with identical dominant knowledge share the same amounts and types of coordinating knowledge, while people with a comparable dominant logic interpret the available information identically, no matter how much (and sometimes what) they know. The newly introduced dominant knowledge is in comparison to the dominant logic concept not only limited to an organization’s top management team, but it is spread over the whole of all individuals who are a part of the focal organization.
Organizational routines. Routines are defined as follows, “We will regard a set of activities as routinized [...] to the degree that choice has been simplified by the development of a fixed response to defined stimuli. If search has been eliminated, but a choice remains in the form of a clearly defined and systematic computing routine, we will still say that the activities are routinized” (March & Simon, 1958). Similar to the concept of routines, dominant knowledge is a dynamic, path-dependent construct, which evolves through an evolutionary process.

Dominant knowledge differs with respect to some details, Routines stress fixed responses to stimuli, as well as the ‘automatic’ interplay of several individuals complementing one another in terms of their complex patterns of behavior (March & Simon, 1958; Nelson & Winter, 1982). The dominant knowledge construct, in contrast, focuses not on what is different and complementary between individuals, but rather on what is similar and what, subsequently, connects their work. In order to do so, complex patterns of behavior are no a prerequisite for dominant knowledge to be effective. It is, rather, key to know what and how to do at all within a focal organization than knowing an efficient sequence of tasks that are somehow related.

Architectural knowledge. Henderson & Clark define architectural knowledge as “knowledge about the ways in which the components [of a product] are integrated and linked into a coherent whole” (1990: 11). The authors refer to knowledge that is directly related to the produced good, which comes close to the ‘technical standards’ dimension presented by dominant knowledge in this study. Henderson & Clark find that communication channels, information filters, and problem solving strategies play a critical role in establishing an organization’s architectural knowledge, a finding that is, at the same time, similar to what this study refers to as the ‘community rules’ dimension of dominant knowledge. The major difference between these two concepts is consequently the ‘overall direction & vision’ dimension, which is prevailing in the new concept, but which is not inherent to what Henderson & Clark term ‘architectural knowledge.’
**Dominant design.** A dominant design is defined as a product class’ design “that wins the allegiance of the marketplace, the one that competitors and innovators must adhere to if they hope to command significant market following.” (Utterback, 1994: 24; see also: Abernathy, 1978). A dominant design refers to a product in an industry or in the marketplace, and the construct focuses on the product’s technical attributes, which producers, sellers and customers accept as given standards. The unit of analysis for a dominant design is the product’s characteristics in an industry (Anderson & Tushman, 1990), while dominant knowledge refers to pieces of knowledge in an organization. As dominant design is an entirely product-related and technical construct, it comes close to one of three dominant knowledge dimensions, namely to the ‘technical standards’ dimension, while the remaining two (overall direction, community rules) remain untouched by this construct.

A dominant (product) design and dominant knowledge share their fundamental ideas of dominance among equals (product attributes or pieces of knowledge), which comes into existence through an evolutionary process of variation, selection and retention (Campbell, 1969; and figure 2.13). In addition to that, prior work on dominant design and this study on dominant knowledge share some further lines of thinking. Firstly, prior literature on dominant design has suggested that dominance results in the products’ and the innovating firms’ temporary monopoly situation on their market (Anderson & Tushman, 1990). This study has similarly been able to illustrate that an individual who initiates and who owns critical portions of the focal organization’s dominant knowledge takes in a higher position in the project’s informal hierarchy (see also paper 3 of this dissertation).

Secondly, research on a dominant product design has found that the pace at which major innovation occurs slows down significantly after the emergence of the dominance (Anderson & Tushman, 1990). In a comparable fashion, this study finds that once dominant knowledge paths are created and established, the organization will be extended through the creation of path dependent knowledge, while the creation of path breaking knowledge is not accepted in the project (propositions 5a-c).
Finally, prior work out that the corresponding industry in which the design is effective has found turns to be more orderly and more stable through the emergence of a dominant product design (Anderson & Tushman, 1990). The reason for this change in the market place is that the product’s elements can more easily be combined and integrated, since they are more compatible, which, in turn, results in more certainty and in more coordinated behavior among the industry’s players. This study found a similar mechanism to be in place, since the individuals’ work could be more easily coordinated in the FREENET project through the establishment of some stabilizing knowledge about rules and standards in the organization.
7. Concluding Remarks

Summary, Coordination characterizes a key topic in management and organization theory. Only little is known to date about how coordination is achieved in communities or in other new forms of organizing beyond the market-firm continuum. These new forms of organizing seem to be most appropriate for coping with the requirements of an ever-changing environment since they enable continuous learning, change and innovation (e.g. Volberda, 1996), which makes new forms of organizing a promising field of study. The present study focused on the case of FREENET, an innovative open source software development project. FREENET is a dispersed, virtual community of voluntary contributing software developers who manage to jointly bring about a coherent and successful product. In contrast to firms, FREENET cannot rely on formal, structural or bureaucratic coordination mechanisms, as its final output is a public good, and property rights to it are strongly diluted. This leads to the set of questions, which guided this work,

first, how is coordination generated in the Freenet open source software development project in the absence of traditional formal coordination mechanisms? By means of an exploratory single case study in the tradition of grounded theorizing, I found a substitute to traditional formal coordination mechanisms to be in place in this setting. An informal, project-specific, implicit and explicit knowledge base, which is shared by the project members, plays a crucial role for achieving unity of effort among the FREENET project members, Project-related knowledge, instead, plays only a minor role in this process. In order to coordination to occur, knowledge needs to fulfill certain conditions: It must cover each of three distinctive dimensions that together span a field of activity in which the overall project is situated and in which it evolves over time. These dimensions include the knowledge of the project’s overall direction, knowledge of its technical standards and knowledge of informal rules within the community. For purposes of coordination, knowledge must additionally be organizational and repeatedly put into use. I introduced the construct ‘dominant knowledge’ to characterize knowledge that is able to coordinate the FREENET project.
The second question, which guided this work, asked, how does dominant knowledge come into existence in the first place, how does it evolve over time? My findings suggest, that dominant knowledge has its origins during the early stages of a projects' inception. Project-specific knowledge goes through a four-step evolutionary process to become dominant and to coordinate a project. It transcends two boundaries, namely from the individual to the organizational level of analysis, and from a single-time to the repeated use of knowledge. Key mechanisms in this process are knowledge creation, capturing & replication, and transfer, which may lead to an overall coordinating dominant knowledge base. Early project members are additionally found to profit from a first knowing advantage.

**Brief discussion.** Some findings of this study are confirmed through prior work. Firstly, dominant knowledge's value as a new theoretical construct has been confirmed through O'Mahony's (2002: 93) work. In her study, she aimed at identifying tactics that community-based open source projects use to manage their (internal) work processes and their relations with firms. She identified three areas of open source projects, which could preserve their communal life and keep it together, and which, therefore, build the core of a community. The three areas she has found are 'work practices,' 'technical direction' and 'goals,' which closely correspond to the three dominant knowledge dimensions (community rules, technical standards, overall direction), which this study found to stabilize and coordinate the Freenet project.

Rüegg-Stürm and Gomez (1997) have further concentrated on one of dominant knowledge's dimensions and they have discussed the effects of an organization's vision, which they defined as "creative images of a possible extraordinary future" (p. 72). When organizations face unstable situations for internal and/ or for external reasons, a vision is expected to encourage entrepreneurial behavior and to cause an increase in flexibility and integration. The authors' perspective supports, on the one hand, the coordinating role of knowledge about an organization's vision that defines all possible ways of behavior in the organization (Collins & Porras, 1991). On the other hand, the authors' perspective supports the adaptable character of this specific type of organizational knowledge, similar to this study.
Thirdly, and most interestingly, some findings of this study were additionally confirmed through software development in a closed source firm setting. Kraut & Streeter (1999), in their survey study on coordination in 65 software development projects in a R&D company, revealed that formal coordination instruments (e.g., formal meetings, written requirement documents or data dictionaries) are less valuable than informal mechanisms (e.g., unscheduled group meetings, or interpersonal networks) to achieve unity of effort in this setting. The FREENET contributors chose, consciously or unconsciously, a coordination mechanism that seems to suit best the specific needs of this kind of setting.

Curtis and colleagues (1988) in their qualitative study of 19 software development projects in 9 companies further found, that communication and coordination breakdowns characterize one of three major salient problems of software development. They have stressed the central role of knowledge that integrates and that gives the employees an overview of their firms. They found that, “Although individual staff members understood different components of the application, the deep integration of various knowledge domains required to integrate the design of a large, complex system was a scarce attribute.” (1988: 1271). One systems engineer who was interviewed for their study remarked, “Writing code isn’t the problem, understanding the problem is the problem.” (1988: 1271).

Limitations and future research. The study’s findings and some methodological constraints to this research, however, leave room for future research. From a methodological point of view, a single pilot case is, on the one hand, an adequate means for gaining initial and in-depth insights into a field of study where only little prior knowledge exists (Eisenhardt, 1989; Yin, 1994), as it was the case for this piece of work. On the other hand, the generalization of my findings remains limited, especially since the FREENET open source project is of outstanding innovative character. Future research is therefore required on different levels of analysis to test the study’s external validity.

Firstly, a future comparison of multiple open source software development projects should reveal in how far the answers found for the FREENET case prove to be similarly valid for other projects. In a first step, such analysis could be realized through the comparison of multiple case studies, and in a second step, through quantitative data
testing. For the latter, an operationalization of the dominant knowledge construct as an informal and project-specific coordination mechanism might be a challenging task. Future work should, additionally, control for variables, which are related to the projects. For instance, a project’s age or stage of development, it’s size, the project success and activity, or the produced type of software could be tested. It could be that knowledge, which serves as both project output and as a coordination instrument, is related to one or several of these variables, and their variation influences a project’s dynamics. While this work found that knowledge about rules and standards coordinates the Freenet project, it would also be interesting to learn if knowledge content shared by a number of organizational members can achieve a similar integration of the individuals’ efforts.

Secondly, it would be most interesting to learn if coordination in communities apart from software development communities, such as sports communities (Franke & Shah, 2001), or the scientific community (Bezroukov, 1999) takes place by means of the same mechanism as identified through this study. Does dominant knowledge as a coordination mechanism then hold exclusively in innovating communities, or is it also valid for communities focusing on non-innovating activities? Would it also hold in the fields of architecture, fashionable clothes, or car design? Since one dimension of the dominant knowledge construct is closely tied to technical activities (‘technical standards’), it could be that the construct in other than software development communities must first be adapted to their specific situations.

The question of how coordination is generated in communities in general is especially interesting against the background that the community-model for organizing innovative work could be a promising mode for bringing about innovative and knowledge-related output (e.g. Benkler, 2003; Meyer, 2003), a task with which many (especially large) firms are currently struggling. If so, under what internal and external circumstances is the community-mode of organizing adequate or successful at all? Studies that are going into this direction of research could finally lead to a contingency model or theory of organization in the context of communities. Such theory should be capable of determining for which activities a community is superior to other modes of organizing, and why or under what circumstances this is the case (e.g. Christensen, 2002; Sutton & Staw, 1995; Whetten, 1989).
Thirdly, an exciting puzzle is whether dominant knowledge can also be found in profit-oriented firms, which can (and do) rely on traditional structural and formal coordination mechanisms that are no valid options in the case of open source software development. If so, has dominant knowledge as coordination mechanism so far been overlooked, since no sensitivity for it existed (as the construct is new) and coordination was wrongly ascribed to alternative, more obvious coordination mechanisms? Open source projects are a unique research setting insofar as they offer an opportunity to study informal coordination, since traditional formal coordination mechanisms simply do not exist in the community context. Especially in young, innovating high-technology ventures, flat hierarchical structures and informal coordination mechanisms seem to play a crucial role (e.g., Hannan et al., 2000; Grand et al., 2002), and communities offer a promising field to expand the findings from the FREENET case,

This discussion leads to some additional questions including, does knowledge in young firms play a similar coordinating role as in the FREENET case that was described here? If so, what is similar, what is different? Is dominant knowledge in a company setting a substitute or just a complement to the traditional, formal coordination mechanisms? And most of all, is coordination through knowledge ‘manageable’? If so, how, and under which conditions? Is coordination through a dominant knowledge base in a firm more successful than traditional formal coordination mechanisms? How exactly should the architecture of innovating firms then be designed? If informal coordination mechanisms prove to be sufficient to coordinate firm activities, we will have to re-interpret Coase’s (1937) question for the existence of organizations and ask, ‘Why is there any formal organization?’

From a perspective that is related to the specific context and findings of the FREENET case, some additional questions remain open and could be subject to future work. On the one hand, this study found dominant knowledge and its evolution to coordinate the FREENET project, and it revealed that not all project members possess the same amounts, or types of (dominant) knowledge. The paper did not answer, however, to what consequences these differences in knowledge lead for the individual project members and why. Are all three dimensions of equal importance in the process of coordinating? Future research should take a closer look at the developers as units of analysis and investigate the consequences of their diversity. On the other hand, a second exciting
question arises out of the specific context and findings of the Freenet case: What would happen if one or several of the core developers and early project contributors who keep most of the coordinating dominant knowledge in their heads, and who have the most complete overviews of Freenet, left the project? Would it change dramatically? Would the project collapse or could it survive without any fundamental difficulties?
Endnotes Part 2

1 For an abridged version of this part of the dissertation see: Kugler (2004).

2 In their quantitative study of multiple projects that are registered on the SourceForge website, Lakhani & Wolf (2003) found the following four clusters of motivators through a web-based survey: First, improvement of programming skills and intellectual stimulation (29% of respondents), secondly, non-work need and extrinsic user need (27%), thirdly, work need and payment for the contributions, extrinsic motivations (25%), and finally, obligation/community-based intrinsic motivations (19%).

3 Effectiveness in the study of Stewart & Gosain (2003) is defined as completion of software tasks in which the project members are engaged. The ideology (culture) notion is defined along with Trice & Beyer (1993: 33) as “shared, relatively coherently interrelated sets of emotionally charged beliefs, values, and norms that bind some people together and help them make sense of their worlds”.

4 A culture that covers the entirety of all open source development contributors to any project might be compared to other communities, such as the worldwide community of architects, engineers or ski instructors. Within the latter, all members share the same passion for outdoor activities, for skiing, the winter and the mountains, they have earned their graduation through a similar education and have passed tough exams, where they learned about the same norms, such as for example not to wear white skiing clothes (which are badly visible in the snow). This knowledge enables, for instance, a Swiss instructor to work in Japan or in New Zealand and to talk to his or her colleagues all over the world (in case they speak the same languages). The knowledge is, however, not enough to coordinate a certain ski school’s everyday work with many students, changing weather conditions, and maybe an accident. In order to do so other mechanisms are additionally necessary.

5 Construct validity denotes if an argument is supported by the appropriate evidence, internal validity denotes if an explanation of cause and effect makes sense in a particular setting, external validity asks if a theory can be applied to other situations, or not (Christensen et al, 2002; Numagami, 1998).

6 While the majority of the interviews were led by telephone, there was one exception. One interview was conducted face-to-face. All interviews with the Freenet core developers (first and second round) emerged out of a research collaboration between the University of St. Gallen and MIT, and the following persons contributed to lead the interviews: Simon Grand, Eric von Hippel, Karim Lakhani, Georg von Krogh, Petra Kugler, Sebastian Späth (in alphabetical order).

7 One of the core developers, for instance, introduced himself as ‘Mr. Bad’, which at the same time is his nickname in the context of the Freenet project. Although Mr. Bad revealed his identity during an interview, this study respects his wish for anonymity.

8 The code repository, or concurrent versioning system (CVS), is a software tool that is widely used for open source projects, in which the current working version of the produced software code and changes to it along with a brief documentation of the code become stored. The CVS enables for work synchronization and to trace back the changes, which the developers have made to the code. In contrast to the software users’ working directory on their own personal computers, CVS is being stored on a central server, ‘the repository’. Software users can make use of the latest stable software version, while the project contributors at the same time further develop and modify the program without interrupting the software’s use.

9 Since the year 2000, the Freenet project has evolved and changed significantly. In 2003, one of the developers (Matthew Toseland) worked, for instance, as a full-time and paid employee for the project. This
fact bears (among others) the danger that Freenet’s character as ‘true’ or ‘pure’ open source project may have changed, for example by changing the contributors intrinsic as opposed to extrinsic motivation through financial remuneration (Deci, 1971). As a consequence, I saw it as crucial for this study and the study’s validity, to refer to peripheral project members who had contributed to the project during the same time period as their fellow core developers (the year 2000) and decided to go the way as described here.

10 A database containing data about Freenet’s mailing list and CVS was build up in the context of a research collaboration between the University of St Gallen and MIT, and the following persons contributed to collect and calculate the data; Eric von Hippel, Karim Lakhani, Georg von Krogh, Petra Kugler, Sebastian Späth (in alphabetical order). The database was subsequently expanded during the course of conducting this work and writing the dissertation.

11 Glaser and Strauss (1967) and Strauss & Corbin (1990) term the steps in the coding process as follows: Firstly, open coding, through which the researcher searches for patterns in the data out of which he builds core conceptualizations, categories, definitions, properties and dimensions. Secondly, axial coding, a procedure for connecting the categories that were found through open coding. This step aims at uncovering causal or intervening conditions for the phenomena observed in the data, consequences or hypotheses. Finally, selective coding, which finally integrates the categories into a coherent line of argumentation or model.

12 The overall goals of the Freenet project as defined by its founder Ian Clarke are as follows: (1) the network should have no centralized control or administration; (2) it should be virtually impossible to forcibly remove a piece of information from the network; (3) both authors and readers of information should remain anonymous if they wish to do so; (4) information will be distributed throughout the network in such a way that it is difficult to determine where the information is being stored; (5) availability of information should increase in proportion to the demand that information thus preventing the Slashdot effect; (6) information moves from parts of the internet where it is in low demand to areas where demand is greater. (www.freenetproject.org, or freenet.sourceforge.net, both 2000).

13 In this work, I will interpret the two types of contributors (core developers and peripheral members) as ‘ideal types’ in order to clarify and better contrast their characteristics. The empirical data reveals, more concretely, that the groups are not two discrete categories, project members rather seem to be located on a continuum between the two ideal types as extremes.

14 Within the core developer group, there is great variance with respect to the amount of created knowledge. Email contributions on the developer mailing list; The four most active core developers account together for 50.48% (sd. 184.48) of all email contributions (83.82% of all core developer emails) (also; von Krogh et al., 2003b); thread initiation; The four most active core developers initiated 34.19% (sd. 118.56) of all threads on the developer mailing list (55.97% of all threads initiated by core developers); CVS commits; The four most active core developers account for 52.73% (sd. 13.47) of all code commits (by core developers) in 2000 (also; von Krogh et al., 2003b). The four most active core developers additionally entered the project on average on 28 of the year 2000 (January 28), with three of them writing their first email in the first five days of the year, and the fourth contributor wrote it on April 12.

15 Henderson and Clark (1990: 11) define component knowledge as; “knowledge about each of the core design concepts and the way in which they are implemented in a particular component”. Architectural knowledge refers to “knowledge about the ways in which the components are integrated and linked together into a coherent whole”.

Authors use various definitions for path-dependence, path-creation, and path-breaking events, and they use the constructs primarily in the context of organizational or strategic change. Originally, these constructs trace back to the work of David (1985) and Arthur (1989). I will use these constructs in the context of knowledge, which is new in the literature, and I refer to the following definitions; Path dependent knowledge
is knowledge that “shapes the probability of moves by actors in the technological domain such that a self perpetuating cycle is established which leads to a […] lock-in” (Boland, Lytinen, Yoo, 2003: 10), and it builds on current knowledge instead of exploring new ones (Karim & Mitchell, 2000). Path-breaking knowledge deviates from the existing paths of knowledge, and it “focuses on how actors can […] deviate from what appears to be the common sense, established patterns,” (Boland, Lytinen, Yoo, 2003: 10. the authors refer with this definition to path-creation). Path-creating knowledge signifies knowledge that is newly established, where no prior path exists which the knowledge could follow, and from which it could deviate.

A metaphor may additionally clarify the relationship of path-creating and path-dependent knowledge creation, as the project evolution can be compared to the growth of a tree. In the early days of the tree’s life, its genes define the roots, its trunk and branches. In later stages of its existence, the tree will steadily grow, but only in minor steps. During later stages, leaves and small branches will add, but only on previously existing major branches.

The major means of communication in Freenet or in other open source software development projects are the mailing list and other forums, the code repository/CVS, publicly available documents on the project in general, such as FAQ (Frequently Asked Questions), papers, presentations, newspaper or journal articles, etc. The knowledge they contain is usually being stored on the project website and so it is easily accessible for all current or future project contributors. Communication takes additionally place on internet online chat forums, such as AIM (AOL Instant Messenger), or IRC (Internet Relay Chat), which do not automatically store the communication. As was already shown in the methodological part of this work, the project contributors ascribe only a minor role to the online chat forums, therefore I did not further concentrate on these sources of evidence for this work.

The knowledge capturing- and transfer media in Freenet do not exclusively focus on only one type of project-specific knowledge (overall direction, community rules, technological standards), in fact, there is a great overlap on what kinds of knowledge they actually do transfer. As a tendency, however, these media focus on one type of knowledge as explained in the body of the text. For clarification purposes I will concentrate on these tendencies of focus and not further explain the knowledge transfer overlaps.

An interesting question for future research is, if not only knowledge on rules and standards is capable of coordinating an organization, but if this is also the case for content knowledge. In fact, in this study I could not always clearly distinguish between these types of knowledge, since the rules make no sense without content knowledge.

A preliminary version of this model was described in Grand, Kugler & Urwyler (2002).

An additional interesting question on the evolution of dominant knowledge is to ask for: Under what circumstances is which of the two paths chosen (1) from individual knowledge creation to individual capturing and replication, or (2) the direct transfer from individual knowledge creation to the organizational level? The available data on the Freenet case could not give a clear answer to this question since the two paths are a result of the data analysis and were not explicitly asked for in the interviews or open questions survey which lead to coming to this findings.

Other authors refer to comparable cognitive implicit coordination mechanisms as organizational cognition (Smircich, 1983), collective mind (Weick & Roberts, 1983), or shared mental models (Espinosa et al., 2001 & 2002).
Part Three

Informally Hierarchical: Structuring Innovative Work through a Knowledge Hierarchy in Open Source Software Development.

Model Generation and (partial) Testing
Informally Hierarchical: Structuring Innovative Work through a Knowledge Hierarchy in Open Source Software Development. Model Generation and (partial) Testing

1. Introduction

Traditional organization theory has focused on formal organizations on the market–firm continuum, which are characterized by two features; namely by an explicit hierarchy and by bureaucracy & a formal structure. Both characteristics trace back to Weber’s idea of a rational bureaucracy (Weber, 1978; e.g., Williamson, 1975; Barnard, 1938; Pugh et al., 1968; Van de Ven et al., 1969), and they are supposed to compensate for human’s bounded rationality and cognitive limitations (Barnard, 1938; March & Simon, 1958; Thompson, 1967).

An explicitly given hierarchy describes a structure of subordination in organizations and legitimizes the individuals’ differentiated structure of power, decision rights, authority, or access to information. Individuals at the top of a formal hierarchy are supposed to have the best overview of their organizations, This should enable them to make the best decisions (Galbraith, 1973; Mintzberg, 1979; Nadler & Tushman, 1988). At the same time, top executives are firm owners or their representatives, and since they hold all property rights to the firm, they have every right to determine the organization’s direction and activities. Individuals at the top of the hierarchy, therefore, enjoy both most of the rights and the most important rights, while the opposite is true for their subordinates.

Traditional organization theory has, secondly, stressed formal organizational structures and a bureaucracy, which should control for the individuals’ behavior in an organization. Burns & Stalker (1961) have termed this type of organization ‘mechanistic’, and it is characterized by autocratic leadership, high degrees of specialization, standardization and formalization, and by a highly centralized decision-making processes. While mechanistic organizations have been found to be most appropriate for
firms confronted with stable market conditions (Burns & Stalker, 1961), for organizational departments with a high degree of routinization (Lawrence & Lorsch, 1967 a&b), or for large organizations (Child, 1972a; Blau & Schoenherr, 1971; Pugh et al., 1969; Van de Ven, 1976), this type of organization is not suitable for firms that are confronted with dynamic environments or with innovative tasks (Burns & Stalker, 1961).

Recent work in the field of management and organization theory is increasingly searching for new forms of organizing, which are more appropriate under dynamic, uncertain conditions. New forms of organizing often go beyond our traditional understanding of organizations on the market-firm continuum, such as in the case of networks (Miles & Snow, 1982; Powell, 1990; Burt, 2000), the learning organization (Fiol & Lyles, 1985; Huber, 1991; Levitt & March, 1988), or communities of different kinds, including online communities or communities of practice (Ahuja & Carley, 1999; Benkler, 2003; Brown & Duguid, 1991). In general, new forms of organizing employ informal rather than formal coordination mechanisms, creating open system structures as opposed to closed system structures. To date, the subject of a hierarchy (of any sort) in new forms of organizing could not yet attract any attention, and it is therefore a blank spot in this growing field of research. Hierarchies can take various forms and must not necessarily be based exclusively on formal authority; they can be similarly informal.

Open source software development communities are one promising new form of organizing, which differ in a wide array of aspects from the traditional profit-oriented firm model. In open source communities, a smaller or larger number of software developers voluntarily collaborate on all kinds of software and without being paid for their efforts. A typical project as, for instance, the Perl programming language (www.perl.org), the Sendmail mail transport agent (www.sendmail.org), or the PGP encryption software (www.pgp.org) exists exclusively on a virtual platform, and all communication between the contributors takes place by means of internet-based devices.

Open source software is often complex, strikingly innovative, stable and surprisingly successful against its commercial counterparts designed in firms. In contrast to software that was developed in a company setting, open source software is a public good, since its central component, the source code, is publicly accessible to anybody who wishes to download, improve or modify the program. No individual consequently holds
any centralized property rights to the software, which could enable him or her to determine where, when, or how the software should be developed. In this respect, no individual has the right to decide on any formal structures or on a hierarchy that should guide the community. Moreover, the contributors to an open source project would probably not accept any formal hierarchy in any case, since they voluntarily donate their time and efforts to the project without being paid for it.

Prior contributions to the fast-growing body of work on open source software development communities, however, are characterized by a striking contradiction. *On the one hand*, prior literature has stressed the democratic and distributed character of open source communities (e.g. Benkler, 2003; Bessen, 2001; von Hippel & von Krogh, 2003). For instance, open source communities have been found to generate low development costs, and to be able to bring about high-quality and highly innovative software, since they profit from widespread information networks.

*On the other hand*, researchers have agreed on the existence of a differentiated member structure with two main categories of contributors, *firstly*, (core) developers, and *secondly*, lurkers or peripheral contributors. *Core developers* contribute a greater amount of work to their respective communities, their contributions are more important for their projects, and they are rather intrinsically (as opposed to extrinsically) motivated (Benkler, 2003; von Krogh et al., 2003a). *Peripheral project members*, in contrast, make the majority of many projects’ contributors. They work less than do core developers and they complete work, which the core developers do not wish to take over, such as functionality testing or bug reporting (Mockus et al., 2000 & 2002; Nonnecke & Preece, 2000; Edwards, 2001). Prior studies have consistently reported that core developers and peripheral members have different rights within a project, what indicates that some sort of hierarchy may be in place in the seemingly democratic open source setting. A hierarchical structure of open source software contributors, however, has to date not explicitly been recognized or analyzed.

Against this background, this work aims at showing that hierarchies prevail in the seemingly democratic open source setting, yet in a different form compared to what we know from prior work about organizations on the market–firm continuum. More specifically, the paper aims at finding answers to the following research questions, firstly,
does the hierarchical principle exist in (seemingly democratic) open source software development communities? Secondly, what is the hierarchy in open source software projects grounded upon, and what determines the hierarchical position of individuals in this setting?

To answer to these questions, this work employs a two-step empirical and analytical procedure, which generates and tests a model of a (knowledge) hierarchy in open source software communities. In a first step, qualitative data on the single case of the Freenet open source project serve to inductively generate hypotheses and a model of a knowledge hierarchy, as it was found to be in place in this specific project. In a second step, quantitative data, which were generated through an online survey and complemented through publicly accessible project data, serve to test the findings generated in the first step. The second step is based upon data from four ongoing open source projects, CrystalSpace, TikiWiki, HSQLDB, and Stepmania.

My findings indicate that although open source software development communities appear to be entirely democratic at first sight, they are only limitedly so. In the communities studied, hierarchies play a role to structure the contributors’ collective work and to maintain a high level of quality in the project. In this setting, hierarchy differs widely from the bureaucratic type of hierarchy that has been studied in the traditional organization theory literature that deals with organizations on the market–firm continuum. In open source communities, the prevailing hierarchy is entirely informal, emergent and is grounded in the contributors’ project-specific knowledge. I refer to it as a ‘knowledge hierarchy’.

My research revealed that an individual’s position in the knowledge hierarchy is influenced by a variety of factors; firstly, by an individual’s accumulated amount of project-specific knowledge as a share of the total amount of accumulated project-specific knowledge prevailing in the entire community. The more an individual knows about a focal open source project, the higher will be his or her position in the informal hierarchy. Secondly, this factor is, in turn, influenced by several individual and organizational-level factors, which include the point of time the individual entered a project, his or her project-specific knowledge absorption behavior before he or she has entered a project, and the individuals’ ongoing activities during the time he or she is a member of the project.
In general, the earlier an individual enters a community, the more project-specific knowledge he or she is able to absorb or create prior to and after the entry, the higher the individuals’ position in the knowledge hierarchy. Furthermore, my findings indicate that accumulated project-specific knowledge creates a number of mobility barriers for present and prospective project contributors, vertical and horizontal mobility barriers within the community, as well as entry and exit barriers to a project.

The subsequent chapters are structured as follows: Next, I will briefly outline the first, theory-generating step of this work, which includes an overview of the methods employed, as well as my first and second order findings. In the third chapter, the methods and findings of the theory-testing part will be presented. Finally, I will conclude with a brief discussion of my conclusions and present promising avenues for future research.
2. Model Generation

2.1 Methods

My interest in studying a hierarchy in open source software development communities was originally more a by-product of another inductive study on coordination in the open source setting (paper 2 of this work) than a deliberate research aim in and of itself. During interviews with contributors to the Freenet open source community, the project members consistently referred implicitly or explicitly to a sort of hierarchy prevailing in their community. Additionally, a variety of indicators supported the interviewees’ statements, also indicating that a hierarchy was in place in the community in question. These findings gave rise to a motivation to inductively study the topic in question, I will outline my procedure in the following paragraphs.1

In this paper, I have structured the presentation of my analysis and findings in two complementary parts with each part consisting of a methods and a findings section. The first part is inductive in nature and it is grounded in a qualitative single case study design on the Freenet project. It aims at understanding whether a hierarchy of any sort can be found in the open source setting, and the factors are analyzed upon which the prevailing hierarchy is based. The output of this analysis is a model that identifies a number of factors which influence an individual’s position in the hierarchy (chapter 2).2 The second part of this work (chapter 3) is grounded in data that were gained by means of a quantitative survey and on archival data on four ongoing open source software development communities. The second part will (partly) test the inductively generated hypotheses.

Such a two-step, complementary procedure is especially useful when only little prior knowledge about a phenomenon exists, such as is the case for this work. The combination of various types of data and methods of analysis (qualitative & quantitative data; exploratory & theory testing) across multiple organizations leads to more objective and more generalizable findings than a more focused study would have been able to provide, since more accounts give more insights. If the findings are similar across methods and organizations, their plausibility is strengthened and the notion of validity
sustained, similar to what is achieved by means of data triangulation (Denzin, 1970; Bijlsma-Frankema & de Bunt, 2003). If the findings are dissimilar, they indicate some potential for future research by digging deeper into the subject in question, and to search for further, more differentiated explanations (Christensen et al., 2002). I will give a detailed overview of the two-step model-generating and testing process as it was used for this study in the following paragraphs.

2.1.1. Inductive single case study design and qualitative data

A hierarchical structure in open source software development communities has to date not explicitly been discussed in prior literature. Besides, it has been obvious that if a hierarchy is in place in this setting, it differs widely from the traditional type of a bureaucratic, authority-based hierarchy we know from most firms. The subject of a hierarchy in open source communities is thus a new concept and there is only little prior knowledge to which this study has been able to refer. A deductive approach to the topic that uses well-known variables, constructs or causalities, is not suitable to grasp the topic’s novelty. I subsequently decided to employ an exploratory, inductive research design to develop a basic, but deep understanding of the phenomenon and the causalities in question.

The grounded theorizing method in the tradition of Glaser and Strauss (1967) enables an approach to a complex phenomenon in a broadly unprejudiced and open way in order to gain new insights on the subject in question through exploratory, inductive research. The grounded theorizing method is suitable for any empirical setting that requires to dig deeply into a set of contemporary empirical data and to uncover new findings and answers to the initial research questions. In search for new insights qualitative data are in essence richer and deeper compared to quantitative data, and they are capable of grasping a complex, sensitive and new subject in its holistic context (Denzin & Lincoln, 2000). Qualitative data are more likely to uncover what purely quantitative data tend to reveal. Therefore, they are more suitable for a study that deals with a subtle topic, such as a hierarchy in a seemingly democratic setting. The study’s theory-generating part is primarily grounded in qualitative data; quantitative figures will only complement and support the findings.
The analysis of any sort of hierarchical structure in open source communities is a highly sensitive topic due to two reasons. *On the one hand*, and according to their original idea, open source communities are designed as an entirely democratic setting and without any structures of subordination that are grounded in an authority similar to hierarchies in most firms (e.g. Stallman, 1999). Explicitly asking the contributors if a hierarchy existed in their project might consequently not have been capable of revealing what was in fact going on in a certain community.

*On the other hand*, even if a hierarchical structure is in place in open source communities, it will be rather informal than formal or rather be implicit than explicit, since the existence of any hierarchy is officially denied in these community settings. It is thus likely that the object of study lies beyond what is most obvious and visible at first sight. Studying a hierarchy in open source projects is therefore a challenging task. For the present work, however, I have been able to circumvent these challenges, since the interviewees themselves indicated that a hierarchy existed in their respective projects. During the interviews I took up the subject and directly asked for their experiences and impressions with the hierarchy prevailing in their community and thereby was able to dig deeper into the sensitive topic.

*The case study method* is applicable to all situations in social sciences where only little known about a phenomenon, and which is so complex that it can only be captured in its entirety, such as is the case for the topic in question for this work. A hierarchy of any kind effects or is effected by the entire organization, including all of its members, how they are related to one another, and their differences or similarities. At the same time, the case study method has the potential to reveal findings that are both relevant and practically useful, since they are directly grounded in a real-life situation that may be one of a kind (Yin, 1994; Eisenhardt, 1989). A hierarchical structure in open source software communities is not only a new but also a complex, multifaceted topic with the project contributors as multiple units of analysis. To cope with the resulting complexity, for the theory-generating part of this work, I focused on a single case study on the FREENET project and will compare its findings to other open source communities in the second, theory-testing step of this work.
2.1.2. Case sampling: The Freenet project

I sampled the Freenet open source software development community (www.freenetproject.org) for a number of reasons, firstly, the relevance of this work’s subject has been discovered as the by-product of a second study, which analyzed the Freenet community. Since the contributors to this project have indicated the existence and hierarchies in their community, they have offered their own project as an appropriate setting for such a research topic. On the one hand, it was very likely to find valuable insights on hierarchies in this specific setting, since several project contributors have also reported findings consistent with what they discovered in Freenet. On the other hand, the contributors also proved willing to discuss this sensitive topic, which was uncertain for alternative communities in the light of my previous experience with prior contacts.

Secondly, since a large part of the open source software development communities’ work is publicly available on the internet, these projects offer unique opportunities with respect to data access. This unique data access is, however, limited to explicit types of data which inform on the contributors’ working style, working intensity, and on the individuals’ outputs. These data consequently illustrate only to a limited extent what lies beyond the most obvious, such as the contributors’ thoughts, impressions or intentions. This information is, nevertheless, crucial for the aim of this study. The access to more sensitive types of data (interviews or open questions) is more difficult, since it requires personal contacts to key individuals in the setting in question, I have already established some initial contacts to members of the Freenet community, and the initial sample of informants has been complemented during the course of the study. This has also been in line with theoretical sampling reflections (Glaser & Strauss, 1967; Charmaz, 2000).

Thirdly, the universe of ongoing open source software development communities is vast, and the projects differ broadly with respect to their sizes, success stories, aims, etc. (Krishnamurthy, 2002; Ghosh, 2002). Freenet is a mid-sized project that is large enough to uncover patterns of behavior in the community. Freenet is, at the same time, small enough to be analyzed as one entity, which is a necessary quality for this work’s complex research focus. Project data have been gathered to trace back Freenet’s first year of existence in 2000. At that point in time, the project was still young enough so that the
project members were able to recall critical events in the community’s evolution. At the same time, the project was also mature enough to identify the emergence of some project structures (including a hierarchy).

2.13. Data types, data sources and collection methods

I focused on a variety of sources of evidence, which helps to triangulate the project data and so enhances the data’s validity and generalizability. For the theory-generating part of the study I analyzed data from the following sources,

firstly, fourteen qualitative telephone interviews were conducted in two rounds with ten of the Freenet project’s 30 core developers. All interviewees were identified from the project’s official developer mailing list, which is available on the project’s website (www.freenetproject.org or freenet.sourceforge.net). The list separates core developers from non-core developers (or, peripheral project members), with the former having direct access to the project’s code repository (CVS), while peripheral members have no direct access to it. All interviews took between one and two hours, and they were recorded and transcribed to facilitate data analysis. All interviews were semi-structured with guidelines defining the overall structure and broad categories of interest. Such a procedure made it possible to remain as open and flexible as possible. Furthermore this method also made it possible to inductively find out central as opposed to irrelevant categories that could identify a hierarchy in the community in question. At the same time, the interviewees’ answers to identical questions guaranteed for a comparison of the contributors’ arguments.

A first interview round was conducted between October 2000 and January 2001. It aimed at generating a broad overview of the Freenet project aim and evolution, of contributors to it, and of the subject of hierarchies in general. I asked for the contributors’ personal backgrounds, and for indicators of a hierarchical project structure. In a second interview round, the preliminary insights were deepened, concretized and verified between March and November 2001. During this second interview round, my questions focused more strongly on the subject of a hierarchy in the community. By means of the second interview round, I aimed at finding out in what the hierarchy in place is
grounded, and which factors discriminate the people at the top of the hierarchy from those who are not. I was able to maintain contact with one core developer that has lasted for the entire process of this study. Through this contact, frequent informal discussions on the study’s findings were made possible during all stages of the work, which further strengthens the study’s construct and internal validity.

One major result of the first and second interview rounds was that great differences between core developers and peripheral project members are in place in the Freenet project. It became clear that it was not enough to exclusively focus on the project’s core developers as key informants. As a consequence, an open questions survey that was sent to peripheral project members served as a second data source. According to the idea of theoretical sampling, this procedure helped to fill gaps in the previously collected interview data and to refine emerging theoretical ideas (Glaser & Strauss, 1967; Charmaz, 2000). The target group for the open questions survey encompassed all 326 peripheral members to Freenet in the year 2000, although they did not necessarily contribute simultaneously (the same day, month, etc.).

I searched the email addresses of 154 (47%) randomly selected peripheral contributors out of the body of emails in the 2000 email development list, and contacted each of them by means of a personalized email by briefly explaining the project and asking for their support. Anonymity of email addresses (e.g. freehaven@..., or Nimbus156@...) was respected at all times and all contributors were also guaranteed complete anonymity of their responses, if they so wished. Additionally, they were able to choose between being contacted by telephone, email, or online chat. Of the 154 addresses that were contacted, 69 (45%) proved to be invalid, leaving 85 valid contacts.

In a second step, all peripheral members who demonstrated interest in the study were sent a second email with an open questions’ file attached into which they have been able to directly type in their answers and comments before they returned it. By and large, the file covered the same questions that were asked during the interviews with the Freenet core developers. The questions were, however, more focused on topics that proved to be critical for the study’s research aim. If the contacted peripheral project members had not responded to the second contact after 14 - 20 days, they were sent a carefully and politely formulated reminder email, to which the survey file was attached.
again. 20 project members returned the survey and their answers, and one additional member gave his statements by means of online chat, each reply ranging from 3 to 9 pages of comments. Most of the respondents showed great interest in the study and they offered additional support either by phone or by email if required.

As soon as a presentable working paper was available, interested peripheral members were sent some preliminary findings for further validation. All contacts and responses took place between September and November 2003. In the remainder of this paper, I will refer to both core developers and peripheral project members as ‘contributors’. The first contact email to peripheral project members, a list of all interview partners and survey respondents, interview guidelines and survey questions are listed in the appendix.

2.1.4. Data analysis in the tradition of grounded theorizing

Data analysis followed the principles of grounded theorizing going back to Glaser and Strauss (1967). Their approach is purely inductive and starts from scratch, which is applicable to a setting for which only little prior knowledge exists, such as for the subject in question for this study. The process of data analysis aimed at gaining new insights on the subject of a hierarchy in the Freenet open source community, and at coherently integrating these insights to new theory. The study’s output is a model on a hierarchy in the open source setting, and it explains why some contributors take in a higher position in the hierarchy, while others do not, and who has the greatest chances to occupy a higher position.

The grounded theorizing method provides a systematic approach for inductive data analysis, since it consists of a set of clear guidelines and steps from which to build explanatory frameworks (Charmaz, 2000). These guidelines include a description of the setting to be analyzed, the process of data coding through which data are broken down and put back together in order to reconstruct regularities in the data, as well as the principle of constant comparison of data (Glaser & Strauss, 1967; Strauss & Corbin, 1990; Eisenhardt, 1989). According to these principles, my data analysis went through several
steps from unstructured data to structured patterns in the data and towards hypotheses on how the emerging concepts are related to each other.

In a first step, my data analysis started with a description of the Freenet community and with the search for indicators of a hierarchy that is prevailing in the project. In a second step, my analysis focused on finding out which individuals were at the top of Freenet’s hierarchy as opposed to who was not. Through a constant comparison of different types of contributors to the project (top versus lower level), and of data from different sources (interviews versus open questions survey), my analysis aimed in the third step at finding out what discriminates the contributors who are on different hierarchical levels. The findings of this step indicate variables that influence the contributors’ hierarchical position, and I summarized the proposed relationships in a number of hypotheses.

Finally, I put the variables and their supposed relationships back together in a coherent model on a hierarchy in the Freenet open source community. During all steps of the data analysis process, the emerging theoretical concepts and relationships were constantly compared to the available empirical data. Such a comparison was possible, since the processes of data collection and analysis were overlapping and longitudinal in nature. Inherently, the process reflects a continuous iteration between inductive and deductive analysis, and it enhances the study’s internal and construct validity.

The following paragraphs will, firstly, give a descriptive, yet focused overview of my first-order findings. Secondly, the findings will be summarized on a more generalizing level in a second-order model on hierarchy in open source communities.
2.2. First Order Analysis

2.2.1. Indicators of a hierarchy in the Freenet project

On the Freenet project website no formal project structure or explicit hierarchy that is comparable to what we know from firms can be found, besides the differentiation between core developers and peripheral project members. Core developers are granted direct access to the project’s code repository, while peripheral members are not. The two contributor types are, however, not explicitly referred to as two different hierarchical layers. The project presents itself instead as a democratic entity, similar to most other open source software development communities (see for instance www.sourceforge.net). Democracy, freedom of speech, anti-censorship, anonymity and an outstanding high regard of knowledge are the most fundamental values and goals of the Freenet community, as declared in the philosophy part of their website (www.freenetproject.org). Hierarchy, though, seems officially not to exist in the project. For many contributors indeed, the inexistence of traditional formal structures is one motivator for freely devoting their time and efforts to an open source project,

“Part of it is that working on open source I don’t have a boss, It’s not my motivation to release the stuff I work on in open source, but it is the motivation to work without getting paid. I look at my code as something that I want to make as good as I can.”

(Core developer F)

The official project presentation, however, differs from how the project members illustrate Freenet. They describe some project characteristics that indicate that a hierarchy exists in the community in question, and they refer to the contributors’ inequalities of ranks, decision rights, of their degrees of freedom to act, or inequality of attention from their colleagues and to the contributors’ unequal reputation. Their characterization comes close to how authors of prior work have described a (formal) hierarchy in organizations, “The pyramid of hierarchical organizations represents a fusion of status, prestige, rewards, and power. […] As we ascend the pyramid all these increase and reach their maxima at the primade of the hierarchy,” (Katz & Kahn, 1966: 221; also: Tannenbaum et al., 1976). In the following I will briefly illustrate each of the above-mentioned inequalities between the Freenet contributors.
**Inequality of ranks.** The project contributors have consistently reported various informal ranks in Freenet, which indicate the project members' diversity and a relationship of subordination within the project. Ian Clarke, initiator of the project, occupies the *highest rank in the community*. His position seems to be unattainable for any other member, at least as long as Ian decides to remain a member of the project. Following Linus Torvalds, the Linux founder, other contributors who have stressed Ian’s outstanding position in the project call Ian a ‘benevolent dictator,’ or a ‘leader’.

The project contributors thereby recognize the originality and visionary character of Ian’s idea to found the project and they listen to what he says. The *second highest ranks* occupy a handful of core developers who are very active, very committed to the project, who have a good overview and some deep knowledge about the intricacies of Freenet. *Two more ranks* take in, firstly, the remaining core developers, and secondly, the peripheral members, who are subscribed to the project development mailing list, but who have no direct access to the Freenet code repository.

Contributors of all ranks range as project members, but the core developers refer to *actual project inclusion* only, if a person belongs to the upper ranks that comprise the project’s core developers who have direct CVS access. To the peripheral members it is often not clear why this difference is being made, but they recognize that not everybody is included into the core developer group and term it ‘elitist’. Implicitly, the differentiation between core developers and peripheral members therefore reflects at least two different ranks. In essence, this structure has comparably been found for other online communities, although it was not yet associated with any kind of a hierarchy (e.g. Zhang & Storck, 2003; Mockus et al., 2000 & 2002). For case evidence see table 3.1.

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project founder; Highest rank</td>
<td>“Ian Clarke is a sort of organizational head. He [...] plays an important role, because he’s at the top and people tend to listen to him.” (Core developer B)</td>
</tr>
<tr>
<td></td>
<td>“All Freenet developers and all persons that are regularly discussing on Freenet issues consider Ian as the founder of Freenet. He created this idea and this vision in the first place. It’s not really like a cult, but I think everybody respects this vision, because everybody who discusses Freenet on a regularly basis is amazed by this vision. So I think it’s just big respect which people have of Ian. People might not always agree on what he says and thinks, but he still is the founder of Freenet. So people listen a lot to him.” (Core developer H)</td>
</tr>
</tbody>
</table>
“Ian Clarke is the top.” * Peripheral member #3 *)

“I’d say that Ian is at the top, it was his idea, he did the initial work, I would characterize him as visionary and persuasive. You can have great ideas but not get anyone to work with you and Ian shows that this is not the case with him by having so many supporters and volunteers, The people at the top have to understand that it may be their idea but it is not their project. They only exist because users want them and eventually they will have to rely on someone else to be useful.” * Peripheral member #10 *)

“The most important thing from an open-source point of view is to be a benevolent dictator.” * Ian Clarke *)

**Other ranks**

“Ian is the head of the whole thing – like Linus Torvalds is to the Linux kernel, He set a general direction to move and was the benevolent dictator, Next, are 2-3 core developers that laid down the law most of the time and what could or couldn’t get in the main project, Criticism was encouraged on the code – which is different from criticizing the hierarchy, Those were taken as attacks on the project or the people and were not tolerated, I simply followed common courtesy and was careful with what I said on the project and kept it technical.” * Peripheral member #3 *)

“There’s generally an obvious leader, who is often not-coincidently making the most changes to the source. They are in charge of saying what goes in and what doesn’t. Sometimes this can be a small (~3 person) group, but a lot of the time, it’s one person, Sometimes it’s the founder, sometimes it’s one of the core developers who picked up the work after the founder left to do something else, Then there are a group of 5-30 (depending on the project) core developers, Well, it tends to be just natural recognition that they understand the project best, They’re making widescale changes, and there are clear benefits to having a unified vision for a project, [They understand the project best out of] a confluence of coding the most, joining early, and the past experience of having made architectural changes to the product,” * Peripheral member #6 *)

**Actual project inclusion**

“I had sometimes the feeling that some were kept out from the core, But this is just a feeling.” * Peripheral member #5 *)

“So in that way anyone can join the list and no one is stopped from posting, But actual inclusion that would be if you had access to the repository to add to the code and well, only core developer K and I have access to add or remove people form the repository. So I guess there has to be some sort of cooperation, and if there were suddenly 20 people, I’d be a little – perplexed. (Core developer B)

“A side effect of a small number of participants is one of the disadvantages alluded to here. And that is the elitist attitude that ‘We were here first, it’s OUR project’ and I saw some people struggling to get recognized as developers (something I was not) – but generally once you proved yourself it was fine, I’d imagine that it’s actually harder now because there are so many people and you’d be ignored, just for a different reason.” * Peripheral member #10 *)

“Freenet is an example of one end of the spectrum [of open source projects] where ‘coders rule absolutely’.” * Peripheral member #21 *)

“There is no need for a democracy, because there is not that many people really involved, I mean, there is a lot of people, but many of them are working on the periphery, So basically, it’s core developer A or it’s me, And maybe a couple of other people, so to say ‘of us’, And we can usually just talk it out,” * Core developer B *)

<table>
<thead>
<tr>
<th>Table 3.1:</th>
<th>Inequality of ranks: Case evidence.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

*) From the provided document.
**Inequality of decision rights.** Inequality between a project’s contributors is, secondly, reflected through a diversity of their rights to make decisions in Freenet. The contributors have reported that no official, explicit decision-making process is in place in the project. The project members have described how they deal with different types of decisions, and they have referred to critical and less critical decisions. In **critical, strategic questions** some contributors have more and farther-reaching decision rights than others,

“The [decision-making] process is not democratic, there are some individuals whose voice counts more than others’.” (Core developer H)

Critical decisions are, firstly, questions that touch the project’s fundamental overall direction and vision. Such decisions are, in essence, met by a small group of very active core developers who have, at the same time, a good project overview. In any case, the project initiator Ian has a final say in these types of decisions. One project contributor has illustrated an example,

“Ian Clarke as project leader has an extremely strong voice, It was once concluded to develop a browser plug-in for Freenet in order to browse by using Netscape or Internet Explorer. And the community favored realizing it. Ian Clarke fought it tooth and nail and finally nothing happened, it was never released. So Ian has a strong voice in meeting decisions.” (Core developer H)

A second type of critical decisions are ‘personnel decisions’, which separate between who becomes a core developer, and who doesn’t. These decisions determine who moves from the peripheral member group to the (informally) higher ranked core developer group. Only a handful of members of the core developer group have this ‘admin privilege,’ which is necessary in order to grant others’ direct access to the code repository.
<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No formal decision-making process</strong></td>
<td>“There is no formal decision-making process,” (Core developer H)</td>
</tr>
<tr>
<td></td>
<td>“We will probably, without saying so, integrate that [a certain feature] into Freenet, No one will really know this, It will just be that, ‘Oh, We’ve made this decision, We can’t tell you why.’” (Core developer D)</td>
</tr>
<tr>
<td><strong>Critical, strategic decisions</strong></td>
<td>“Ian Clarke and the 3-4 main core developers are the main decision makers, For the side projects there are various leaders, Some people (the client install developers) put themselves at the top of their project with people sending bug reports straight to them,” (Peripheral member #19)</td>
</tr>
<tr>
<td></td>
<td>“They have Ian Clarke as initiator and I think many people are listening to him and if he decides something, then it’s more or less fact, But never the less, there is no bug cracking, there is no release decision-making in place, That’s something I have criticized some time ago, because discussions takes place on the mailing list and actually there is no means to capture the results of a discussion,” (Core developer H)</td>
</tr>
<tr>
<td></td>
<td>“I think we all recognize that an important part of the success of open source is deciding who gets to be a developer and who stays in and who – […] – is not a developer, Because at some point you have to draw a line and say, ‘We don’t want this, your stuff is bad, go away,’ […] So I’ve left that up to the people who have the admin privilege on SourceForge, They are the people who have the capability of adding and dropping developers, […] That’s core developer B, core developer A, and core developer K, […] I try to stay out of that stuff, and sort of let them make the decisions about that kind of thing,” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“If something is going to go into Freenet, it really has to be bettered by core developer B and core developer C. That’s one branch or one group,” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“Let’s say yes [I decide on inclusion], Because anyone can send a piece of code and if I like it, I’ll just put it in,” (Core developer B)</td>
</tr>
<tr>
<td><strong>Less critical decisions</strong></td>
<td>“There’s a rough consensus that if there’s a problem with the Windows installer and the Windows code, ‘Talk to core developer H about it, He’s the one who’s been working on that for a while, He’ll be able to give you a hint in making the change, He’ll make the change himself,’ That’s one level, He’s the person that’s tending to do those changes, Then, there’s a second level where it’s, ‘Don’t make a change without talking to core developer H about it,’ He’s not just the person who is best suited to help you with it, He is now the person who ‘owns’ that code, and it would be a big mistake to try and make a change without talking to him about it,” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“Competency is essential, competency in specific areas of specialization, I am not entirely familiar with all of Freenet’s parts, which is the case for most developers. They are searching for their ‘niche’, in which they have most competencies, and for which they are responsible as a spokesperson. And with decisions, our voice counts more,” (Core developer H)</td>
</tr>
</tbody>
</table>

Table 3.2: Inequality of decision rights: Case evidence,
Finally, since direct code modifications or postings of new code to the repository can only be accomplished by the core developers, peripheral project members, who wish to contribute code, firstly have to get in touch with a core developer to commit their suggested changes. The core developers serve as gatekeepers who watch and control the code repository. If the core developers liked the suggested code they would include it, if not, they wouldn’t. Core developers thus have the power to decide on actual project directions, and they have more decision-making rights than the peripheral project members. Since Ian Clarke occupies an outstanding, predominant position in the core developer group and has a final saying in all decisions, he enjoys even more rights than the other core developers.

Less critical project features, which arise out of Freenet’s day-to-day work and which follow the established project directions, become discussed on the mailing list, and if a number of people agreed on the subject in question, the contributors would start coding it. In Freenet, the project contributors each tend to be specialized on certain project features or niches (von Krogh et al., 2003b) as, for example, the development of code on Freenet’s user interface (GUI). Those contributors who have initiated a certain feature are, at the same time, responsible for their area of specialization, since they are supposed to have most knowledge about their niche. But also when it comes to decisions about ‘their’ project features, the individuals responsible for a piece of work have more to say compared to more occasional members (see table 3.2).

Inequality of degrees of freedoms to act. Inequality between contributors is, thirdly, visible through different degrees of their freedoms when acting. Some contributors have more immunity when they act than others; they are allowed to do things or to behave in ways that would not be accepted from others. Higher ranked contributors, in essence, have more freedoms than lower ranked individuals. The most obvious freedom is to make direct as opposed to indirect changes to the project’s code repository, which distinguishes core developers from peripheral project members. Moreover, what and how much an individual is allowed to change in the project’s code repository differs widely between the contributors.
One very active and capable core developer, for instance, is allowed to change the most fundamental project features without the need to inform his colleagues and without getting punished for doing so. The same behavior would not be accepted from occasionally contributing peripheral project members. The FREENET contributors report, additionally, that the freedom to be harsh or unfriendly vis-à-vis their volunteering colleagues is accepted from a handful of higher ranked individuals, while the same behavior would be a faux pas from newcomers to the project. One contributor’s statement illustrates the difference (for additional case evidence see table 3.3),

“There’s an expectation that you will make changes to the system in accordance with your relative ranking within the system. Obviously, there’s no formal ranking. Core developer C can go in and change the networking from top to bottom, and that’s fine. No one is going to question that, I can go in and make some changes to some fundamental stuff. I’ll definitely have to pacify core developer B, K, and C. Other people at my level, peripheral member #23 and peripheral member #24, would be able to do that. Then there’s a lower-down thing, where we’re glad to get your input. We’re glad to have you fill in these tasks that haven’t been done yet, but don’t have all your own ideas and make big changes on your own.”
(Core developer J)

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degrees of freedom to act</td>
<td>“[And he said] ‘I’ll give you these commit rights [to the code repository]’. Actually we don’t have several levels of rights. So he gave me the right to change everything.’” (Core developer H)</td>
</tr>
<tr>
<td></td>
<td>“And core developer B, he is the programming god. He can do whatever he wants to, anyway, Because I think he is the only one who has a good overview of the code now.” (Core developer H)</td>
</tr>
<tr>
<td></td>
<td>“Ian Clarke was always in charge, without question. Oh, core developer B might get a little snippy sometimes, but final decisions always seemed to be up to Ian Clarke and, if he wanted to make changes in the direction of the project, he could do so without asking anyone.” (Peripheral member #15)</td>
</tr>
<tr>
<td></td>
<td>“[The ranking is] Ian Clarke, core developer K and core developer B, in that order, and there was no doubt, just from they way they answered questions, Ian was firm, but open to discussion, core developer K somewhat less so, and core developer B was a grouch.” (Peripheral member #15)</td>
</tr>
</tbody>
</table>

Table 3.3: Inequality of degrees of freedoms to act; Case evidence.
Inequality of attention & reputation. The contributors have reported about how and to whom they devote their time and attention, their most crucial and valuable resources for their voluntary contributions to FREENET. The contributors make broad differences between certain project topics, or between the creators and authors of a piece of work, and for each there are some more important than others. The contributors have reported, for instance, that they are most interested in the work of a handful of core developers, among them Ian Clarke, the project initiator. One contributor (core developer H) reported, in contrast, that he devotes only little attention and little time to newcomers’ work, if at all. He has reported on the use of a filter mechanism, which transfers the messages of one specific person directly into his computer’s trash can.

With respect to certain project topics, the contributors have reported that they take a closer look at work closely related to their own field of specialization and to their own contributions to FREENET. In addition to that, the contributors tend to read constructive ideas that progress FREENET along the ideas of already established project directions. The contributors’ (path-breaking) criticism on the project, instead, can barely attract the members’ attention, since criticism tends to be time-consuming, and it does not immediately lead to the project’s progress. As a consequence, the contributors’ behavior enables them to focus on some parts of the project, while for them personally others remain unnoticed, since they are irrelevant for their individual work. This kind of behavior helps the contributors to efficiently manage their voluntary and limited time investments for FREENET. All the issues that can be found on the developer mailing list, in general documents on the project (e.g. papers, foils, etc.) or in the code repository compete therefore directly with each other for the project members’ time and their attention. Those issues that are able to win the competition by attracting the contributors’ attention, are most likely to become realized in FREENET.  

The project contributors capable of attracting most attention in the project are essentially those individuals who are also able to attain the best reputation from their colleagues. On the one hand, a good reputation is a motivator for the voluntary contributors who receive no financial rewards for their efforts.
<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
</table>
| **Attention and time** | “As far as people that I especially listen to, I'll usually read stuff that core developer K or core developer A post, Core developers B and C are relatively rare enough now, which is strange, considering that they’re nominal leaders of the projects, I'll read what they have to say and anything that says, ‘I have a good idea,’ I'll tend to skip over, ‘Freenet is good, but...’” (Core developer I)  
“If you don’t code, your opinion is of a lot less significance, Ian Clarke has the most audible voice of anyone who’s not actually coding.” (Core developer F)  
“I'll usually browse all emails and I'll pre-select them according to their subject title and according to their author, I will read in any case mails that are written by Ian Clarke, I'd read the emails of other core developers, such as people who are also working on the Windows Installer [my field of specialization in the project], Newcomers' messages, I will rather skip over, [...] I am not sure if I'd read their first email at all, If their name starts to appear more often on the list, I'd probably take a closer look at the messages, besides the author is an idiot, In that case, I'd ignore his messages, or I’d even filter the email addresses,” (Core developer H) |
| **Reputation** | “Reputation is crucial, and at the same time, it is an indicator of the social pecking order in Freenet, But reputation is also what we get as a reward for our work, since we are not paid for what we deliver,” (Core developer H)  
“If you’ve never heard from them before, you don't really know if they can do it or not. If they’ve been around for a while, and have demonstrated that they can do that kind of thing, or if they’re someone who has released software that does that kind of thing, then you feel more secure about what they’re doing, in open source projects in general there’s some kind of pretty complicated social reputation thing going on where people will strive to maintain their own reputation and also to get a good idea of the capabilities of others, Anyone who claims to be able to do something and then it turns out that they can't do it, gets a bad reputation very quickly,” (Core developer F)  
“The way they were operating there, you can't just say, ‘Oh, I want to help your project, and I’m going to do this,’ You have to get their respect first, and that's a way to get someone’s respect, to do something,” (Core developer G)  
“In order to receive a good reputation one has to contribute to the discussion on the mailing list in a competent way, It is not necessary to submit source code, Core developer K is such an example; he talks competently and gives advice on what others could do, “(Core developer H)  
“Reputation in the open source community happens through activity, moreover by means of discussion, activity, contributing one’s own competencies and submitting changes, Taken together, it is if someone is capable of advancing the project, somehow,” (Core developer H) |

Table 3.4: Inequality of attention & reputation: Case evidence.
On the other hand, a good (or bad) reputation is an (retrospective) indicator of the contributors’ knowledge about the project (von Krogh et al, 2003a) and of their informal ranks in the community. Since no official information on the contributors’ personal working or educational background is available, their reputation indicates what type of work an individual is capable of or willing to contribute. This information makes the contributors’ efforts more certain and it enables for a minimum of (informal) planning and stability in the volatile open source setting. Those individuals, who were able to establish a good reputation over a longer period of time through consistently providing work and knowledge to the project, enjoy more decision rights, since their voice counts more. Newcomers, in contrast, must first fight to become established in the community by building up their reputation (table 3.4).

2.2.2. Bottom and top of the hierarchy.

The hierarchical structure pertaining to FREENET is, in contrast to traditional firm structures, entirely informal, but nevertheless it is prevailing, and the project contributors are expected to respect it at all times. They refer to the hierarchy as ‘social pecking order’, ‘flat hierarchy’ or ‘informal hierarchy’,

“When I was paying attention [to Freenet] it was still a fairly flat hierarchy.”
(Core developer I)

“It is an entirely informal hierarchy. There is never anything explicitly said or laid down.”
(Core developer H)

“There is some sort of social pecking order.” (Core developer H)

“By the time I got to Freenet, there was already an established pecking order, and the good parts had already been picked out. I tend not to do stuff that’s as interesting.”
(Core developer J)

Who are the FREENET contributors who occupy a position at the top of the informal project hierarchy, and who does not? A comparison of the indicators of a hierarchy illustrated above demonstrates that the positions in FREENET are not equally distributed between all project contributors in the sense that some individuals enjoy most decision rights, while others have the best reputation or most degrees of freedom to act. It is rather a handful of
core developers who are at the top of the hierarchy, since all indicators seem to match coherently and to positively correlate. Those who have the best reputation, at the same time, also attract most attention, have a final say in decisions, are ranked highest by their colleagues, and enjoy most freedoms when acting. Members of the core developer group consistently take in a higher position in FREENET than the members of the peripheral group.

However, the simple differentiation between core developers and peripheral project members is not sufficient to describe the hierarchical structure prevailing in FREENET. Neither all core developers nor all peripheral members are ranked or treated the same, Differences in the core developer group become clear, for instance, through Ian Clarke’s position; Ian is coherently recognized as an outstanding person who occupies the highest position in FREENET. The unambiguous hierarchical differentiation between the members of both contributor groups allows, however, to uncover those factors that influence an individual’s position in the informal hierarchy and it allows us to ask, 4

what lies beyond the relationship of hierarchical subordination as it was discovered in the FREENET project, if not the traditional formal mechanisms of bureaucracy, authority, and bundled property rights? What is the informal hierarchy prevailing in the FREENET project grounded upon?

2.2.3. Knowledge hierarchy

The hierarchy that was discovered in the FREENET project is unequal to the bureaucratic type of an authority-based hierarchy we know from most firms. The contributors’ hierarchical positions are rather influenced by their specific knowledge about the FREENET project, to which they donate their efforts, and I define knowledge along with Nonaka as “justified true belief” (Nonaka, 1994: 15). The project contributors themselves refer to the informal structure that is based on their project-specific knowledge as ‘meritocracy’ or as ‘well-informed democracy’ and one contributor has reported,
“It tends to be just natural recognition that they [people at the top of the informal hierarchy] understand the project best. They’re making the wide-scale changes, and there are clear benefits to having a unified vision for a project. [They understand the project best out of] a confluence of coding the most, joining early, and the past experience of having made architectural changes to the product.” (Peripheral member #16)

Those who have the profoundest knowledge of their collective project take in the highest ranks, they have a final say in fundamental decisions, they enjoy the most degrees of freedom to act, they attract most attention and get the best reputation. They can, therefore, be considered as the top of the informal hierarchy, and vice versa, and the more project-specific knowledge a contributor has, the higher will be his or her position in the informal hierarchy. The knowledge that influences the individuals’ hierarchical position in Freenet is rather project-specific than project-related, and it covers the project’s content, Freenet’s architecture and rules on the project’s overall direction, technical standards or on accepted behavior in the community (see part 2 of the dissertation).

In a virtual open source project, the only means through which the contributors’ newly added or modified knowledge can be transferred to others or be spread within the community, are electronic devices. The project members’ email or code contributions are, at the same time, the explicit part of their efforts, and their work becomes stored on the community website, where it accumulates over time. As one consequence, the project members’ explicit contributions can easily be recalled at any point of time, and as a second consequence, the project contributions that are stored allow for a wide-scale process of peer-review by their colleagues. All contributors are asked to make their arguments clear to their audience and to justify their work, since their peers will carefully review their contributions (for additional case evidence see table 3.5). One member has recalled,

“There’s also this intensive continuous peer review process that means that if somebody doesn’t have the appropriate skills or understanding, they will very quickly be admonished for it. It’s this intensive instantaneous peer review that makes it much easier for people to self-select.” (Core developer A)
Table 3.5: Knowledge influences the hierarchy; Case evidence.

2.2.4. Project dynamics and knowledge accumulation

Project-specific knowledge, which influences an individuals’ position in the informal project hierarchy, is stored and critical on two different levels of analysis, firstly, on the project level, and secondly, on the level of every individual contributor to Freenet. The explicit part of Freenet’s project-specific knowledge is stored on the project-level, more specifically in the project’s mailing list, in the code repository and in other publicly accessible documents on Freenet. On this level of analysis, Freenet’s knowledge base covers every explicit piece of knowledge that was created or modified, and that was made public since the project’s initiation. Project-specific knowledge refers additionally to the individual level of every single contributor to Freenet and characterizes how much he or she knows about the project. Unlike knowledge on the project-level, the individuals’ knowledge refers to both tacit & personal as well as explicit & publicly
accessible parts. On both levels, the resulting knowledge bases are no static stocks, but continuously in motion, evolving over time.

On the project level, the FREENET contributors report on the dynamic, time-dependent and consequently evolutionary character of the specific knowledge that resides in FREENET. The project contributors continuously submit messages to the email list and they permanently add source code to the repository or change the code. Therefore, project-specific knowledge grows & accumulates, it changes, and becomes more complex over time, which has also been confirmed by the FREENET project members,

“It is very hard to keep up [if someone misses to read the CVS or discussion list for some time]. There is so much going on in the project. GUIs, encryption, network protocols, searching.” (Peripheral member #15)

“Another developer and I were about starting on a GUI interface for Freenet in C or C++. Development on the core protocol was moving so fast that just the two of us couldn’t rewrite it quickly enough to catch up,” (Peripheral member #3)

“Over time, the source code becomes increasingly complex. […] The contributors who couldn’t grow along with the evolving project have difficulties to grasp the project’s complexity, since they could not learn how things work, or why things work. My impression is that it has become extraordinarily difficult to acquire the knowledge that is necessary to capture what is going on in the project.” (Core developer H)

The methods by which knowledge that resides in FREENET has accumulated over time become clear through the growing number of emails posted on the project’s developer mailing list in 2000 and through the increasing number of code logs added to the code repository during the same year (see figures 3.6 and 3.7). The figures illustrate both the accumulated and absolute numbers of email or code contributions to FREENET. These curves show that the accumulated numbers have a similar, almost linear shape, which illustrates that this project’s email list and code repository have grown in parallel. The (accumulated and absolute) numbers of CVS logs, however, are at every point in time considerably smaller than the numbers of emails posted on the project’s mailing list.
Figure 3.6: Monthly increase of emails posted to Freenet’s developer mailing list, and accumulated number of emails (based von Krogh et al., 2003b).

Figure 3.7: Monthly increase of code logs posted to Freenet’s code repository, and accumulated number of code logs (based von Krogh et al., 2003b).
Moreover, the ratio of emails and code commits has increased significantly over time and with a growing number of contributors to the project, in March 2000, the average number of emails per CVS log was 4.91, and in December 2000, it was 30.15. This finding indicates that the ‘noise’ or discussion level on the list has increased with every new contributor to the project. Since Freenet is a network in which many elements are inter-linked, the project’s complexity has expectedly grown rather exponentially than linearly. With every piece of code or email that added to Freenet, the project should have become increasingly complex over time. I conclude that the older and the larger an average open source software development community will become, the larger it’s accumulated project-specific knowledge base.

Similar to project-specific knowledge that resides on the project level, knowledge may also grow and accumulate, change and become more complex on the individual level of every project contributor. How much an individual knows about a specific project must consequently at every point in time (during a project’s evolution) be put in relation to the accumulated amount of knowledge residing in the entire project. A contributor who knows everything about a project at an early point in time (when only little project-specific knowledge exists) may know almost nothing about the identical community, if the project advanced rapidly and his or her individual knowledge does not grow at the same rate. One contributor characterized the illustrated relationship as follows (for additional case evidence see table 3,8),

“As a result of this ad-hoc structure and direction [in Freenu], if you don’t follow the discussions then you may find that your ideas or goals have been forgotten and new ones taking in their places. Imagine if someone got into a cave in 1980 and then came out in 2000, they’d be amazed by advances in technology and the goals around it. Skipping out on discussions lists is like this on an OSS project because they move so fast. (Certain ones move fast I should say… Xfree86 is notoriously slow paced).”

(Peripheral contributor #10)
<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation between individual and project knowledge, project dynamics</td>
<td>“Well, you'd have to ‘catch up’ [if a contributor missed to read the CVS or the discussion list for some time] reading archives/code,” (Peripheral member #6)</td>
</tr>
<tr>
<td></td>
<td>“To become a core developer in Freenet is <em>incredibly</em> difficult now. Also, projects become more complex and solidified,” (Peripheral member #16)</td>
</tr>
<tr>
<td></td>
<td>“Quantity/ quality of code and other contributions, which shifts over time.” (Peripheral member #16)</td>
</tr>
</tbody>
</table>

Table 3.8: Relation between individual knowledge & project knowledge, project dynamics: Case evidence.

2.2.5. Factors that influence the individuals’ project-specific knowledge

The FREENET contributors have reported a number of factors that influence the individuals’ amount of project-specific knowledge. Indirectly, these factors also influence the contributors’ position in the informal project hierarchy, since the more project-specific knowledge an individual has, the higher his or her position in the informal hierarchy. Firstly, the FREENET contributors referred to an individuals’ knowledge absorption behavior before he or she officially entered the project, Secondly they referred to the contributors’ ongoing involvement through knowledge absorption and creation activities while an individual is a member of the community. Finally, they referred to the point in time at which an individual entered the project and the longevity he or she is actively contributing to it. For each of these factors, the higher-ranked core developers’ behavior differs from that of the lower ranked peripheral contributors. I will illustrate each of the above-mentioned aspects in a more detailed analysis in the following paragraphs.

Individuals’ knowledge absorption behavior before project entry. Individuals who occupy upper-level positions in the informal FREENET hierarchy, who are essentially the project’s core developers, reported their passive ‘lurking’ and watching behavior before they officially entered the project by submitting their first email to the list (also; von Krogh et al, 2003b). By ‘lurking,’ the prospective contributors were able to obtain a broad overview of the project’s content, it’s architecture, or on the rules and standards that guide the community. The core developers characterized the benefits they got from their ‘lurking’ activities as ‘project overview’ or as ‘psychology of the project’. They reported
that ‘lurking’ helped them find a niche in Freenet (also: von Krogh et al., 2003b), or to define a new one. Such a specialization should be both related to the others’ contributions and not yet occupied by any other project member. The core developers reported that they had spent more time ‘lurking’ and that they did so through more types of knowledge transfer media than did the members of the peripheral contributor group.

The core developers reported that they were able to absorb project-specific knowledge from various knowledge transfer media, and they referred to the project’s developer mailing list, to the code repository, and to other publicly available project documents. Essentially, they absorb knowledge about the Freenet project from more than one, and in most cases from all types of knowledge transfer media, each of which stores and transfers a different type of (complementary) knowledge. As a consequence, the core contributors gain a broader overview of the project than any single knowledge resource has been able to provide. Besides, all core developers have reported that they had spent several months observing and absorbing deep project-specific knowledge until they felt ready to officially enter the project, and by submitting their first message to the email list. During this time of ‘lurking,’ some core developers already started to play with the code, yet without submitting their contributions to make them public in Freenet. One core developer recalled,

“It was two or three months [the time I lurked] and I was working without anybody telling what I was doing. Then, when I needed protocol information, I mentioned why I needed it.”
(Core developer D)

Members of the hierarchically lower positioned peripheral project contributor group, in contrast, reported on their differing behavior with respect to the absorption of project-specific knowledge prior to their entry to the project. On the one hand, peripheral project members ‘lurked’ only for a short period of time, e.g., a few days, if at all. Some peripheral members recalled that they jumped into the project right from the start and without any knowledge of what was going on in the project, On the other hand, the individuals who had spent some time absorbing Freenet-specific knowledge, focused primarily on only one or two knowledge transfer media, but they did not read through all the knowledge that was available,
When ‘lurking,’ the peripheral members focused on the project’s email list or on some publicly available documents on FREENET, but not on the project’s source code. Of the 20 peripheral FREENET members who reported on their own behavior, 12 remembered the time they had ‘lurked’ before they entered the project, further reported that 11 individuals browsed general project documents, such as Ian’s thesis or journal articles, 6 peripheral contributors browsed the email list and only 4 read through the code.

In comparison to the peripheral project contributors, the core developers spent more time absorbing project-specific knowledge of FREENET. They used more channels to absorb different types of knowledge before entering the project. Members of the core developer group, who are in upper-level positions in the hierarchy, have consequently been able to absorb more project-specific knowledge before they entered the project. On average, a member of the core developer group was granted access to the CVS after 16.50 days (sd. 60.44) he or she submitted the first email to the development mailing list. For the core developers, the time span between their official entry to the project and their promotion into a (informally) higher position in the hierarchy was on average half a month.

Immediately after their official project entry, the members of this group were able to use and demonstrate their knowledge and their willingness to contribute to FREENET, while the members of the peripheral contributor group lacked the respective knowledge at the time of their project entry. The core developers were immediately able to gain a good reputation in the project, and become active on the mailing list, I conclude, therefore, that the more time a prospective contributor spends ‘lurking’ and absorbing project-specific knowledge before he or she enters an open source project, and the more types of knowledge transfer media he or she uses therefore, the higher will be her (immediate) position in the informal hierarchy. For additional case evidence see table 3.9.
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Peripheral members (lower positions in the hierarchy)</th>
<th>Core developers (higher positions in the hierarchy)</th>
</tr>
</thead>
</table>
| Knowledge absorption before project entry | “I just jumped right in, I have contributed to email lists and usenet discussions for years and have no problem with getting into something and getting involved with a discussion.” (Peripheral member #15)  
“Learning about the project took maybe a few days, Perhaps a week but I ‘entered’ the group by subscribing to the developer mailing list after a few days, I didn’t learn some aspects of the group till later – especially I didn’t learn how barely functional the system was till I started actually using it,” (Peripheral member #4)  
“I read about the project in the Freenet area of Sourceforge and links from there to get some background, I installed the program and subscribed to the various Freenet lists all about the same time.” (Peripheral member #14)  
“I think I’ve only looked at the email discussions and perhaps some documents on the web about it, I’m not sure I learnt anything.” (Peripheral member #1)  
“I think I read Ian’s thesis in the summer of 2000, before joining the mailing list, I never got round to reading the code, Most of my knowledge came from the mailing list, since the design was always evolving and the documents on the website tended to be out of date (although still useful as a reference).” (Peripheral member #17) | “I was sort of reluctant to start messing with stuff until I figured out what the psychology is, What people, who are already working on it, what their psychology was and what direction things were already going.” (Core developer I)  
“It might have been two or three months [the time I lurked]. Yes, I think this might be the time it needed, It’s really not that easy to estimate because I started just receiving, the mailing list, just without ever responding, And then step-by-step you start replying to some mails or start to send some new ideas, So that was really step-by-step, I didn’t start actively participating in the discussion at once.” (Core developer H)  
“Well, on the Freenet list, if one was a lurker, one would have an idea of what was going on with Freenet and how it worked and stuff; whereas, if you don’t lurk on the list, it’s much harder to get an idea of what’s going on, I guess you’d have to be reading through code rather than reading through the mailing list or something. In terms of just knowing about how it works, and how it’s going to work and stuff, if you’re interested in that, hanging around is advantageous.” (Core developer F)  
“Lurking enables to getting an overview of the topics and problems in the project, and of which have already been solved or which are still open, The latter enables us to become active ourselves and contribute something to the project, finding one’s niche,” (Core developer H) |
**Individuals’ continuous involvement while on the project: Knowledge absorption and creation.** The project contributors have reported on a second factor that influences how much project-specific knowledge they have. After they had submitted their first email to the list and thereby officially signaled their project entry, the contributors have, to different degrees, been actively involved in the progress of the project. In order to capture the project’s very intricacies and to keep on track with the permanently growing and changing project-specific knowledge base, absorbing project-specific knowledge cannot be enough.

The contributors have reported that while from the outside it often appears as if nothing was going on in Freenet, this was in reality not the case. Some knowledge such as, for instance, personality conflicts, detailed technical problems or technical decisions, can only be recognized if the contributors constantly followed the project’s progress. The contributors’ continuous involvement refers to both the (passive) absorption of project-specific knowledge created by others, as well as their own (active) creation and modification of new knowledge. However, the behaviors of core developers who are in higher positions in the informal hierarchy and of peripheral members, who occupy the lower positions, differ with regard to both aspects.

**Firstly,** the two contributor types differ with respect to the average amount of time they had invested by passively absorbing the knowledge created by others in the project after the contributors had entered the project. Since the contributors permanently give up ideas or goals that prove not to be relevant or suitable for their community, and they add new knowledge instead, the Freenet project is continuously evolving. For keeping on track on how the project is being modified by others, the contributors are asked to constantly follow Freenet’s evolution and to browse the mailing list, the code repository, or other publicly available documents about the project. The contributors have referred to their passive knowledge absorption as their personal ‘learning’ or as ‘experience benefits’.

Members of the peripheral member group have reported that on average they had spent an hour or less a day browsing the email list or reading through other project-related documents. Only few peripheral members read the source code continuously. Most members of this group take no look at the code repository at all and they do not
know how, or if, the source code progresses. Core developers, in contrast, on an ordinary day spend on average about an hour or significantly more time reading the mailing list, browsing code modifications or other general documents.

In addition to that, and unlike the peripheral members, the core developers reported on work-intense peak times such as, for instance, before a new code version is released or during semester breaks, which places more time at the disposal of those core developers who are students. They reported that in peak times, which may last for weeks, they spent several (up to twelve) hours a day working on Freenet (absorbing, thinking and creating knowledge). Core developers should therefore continuously absorb a greater amount of project-specific knowledge than do peripheral project members, and so they gain a higher position in the hierarchy. In sum, the more project-specific knowledge a contributor to an open source software development community continuously absorbs after her entry to the project, the more project-specific knowledge he will have, and the higher her position in the hierarchy.

Secondly, the contributors are continuously asked to (actively) create and contribute new knowledge themselves. An individual contributor’s passive absorption of project-specific knowledge is invisible to his or her fellow project members. To become recognized by the other project contributors, an individual must also actively and continuously submit and externalize his or her ideas, discussion, or code contributions to the virtual platform. Active contributions to the project are not limited to submitting code; it is similarly acceptable to ask intelligent questions on the mailing list (which is, for instance, reported on core developer L), or to structure the project’s evolving vision in papers or in other documents about Freenet, One contributor has remarked,

“The person who gets their way is the person who’s willing to put the most time into it.”
(Core developer F)

On the one hand, such activity signals to all project contributors that a project member is available and willing to contribute to the collective work. The project’s external boundaries of who can be considered as an active project member and who cannot, thereby become more predictable and less uncertain. On the other hand, by actively submitting knowledge to the project, a member demonstrates his or her capabilities,
skills and personal understanding of the project, facts about which the other contributors have no other information, since no formal recruiting process exists for new project members.

Project members who actively contribute work of high quality to the project and who demonstrate their knowledge to the others, do build up a good reputation, they are ranked higher by their colleagues and they enjoy more decision rights, while contributors who deliver bad work receive sanctions from their peers. Individuals who actively reveal their knowledge are able to reduce the uncertainty on what or how much he or she is capable of submitting, and so their behavior therefore enhances the project’s stability. The higher positioned core developer group and the lower ranked peripheral member group differ again with respect to how much project-specific knowledge they create in Freenet, as the following indicators (email contribution, thread initiation, CVS commits, general project documentation) will demonstrate. Core developers permanently create more knowledge for Freenet than do members of the peripheral contributor group.

In the year 2000, the 30 core developers contributed a total of 6,751 (60.22%) emails to the developer mailing list, which is on average 225 emails per core developer over the year. Together, the core developers initiated 61.08% (1,047) of the total 1,714 (von Krogh et al., 2003b) email threads, which is on average 34.9 threads initiated per core developer. The 326 peripheral members, in contrast, contributed a total of 4,459 emails (39.78%) to the developer list, with on average 13.68 emails per person. The peripheral members initiated a total of 667 (38.92%) threads on the list, with on average 2.05 threads per peripheral member in 2000. On average, the core developers posted more than 16 times the number of emails the peripheral members had submitted, and they initiated more than 17 times as many threads compared to an average peripheral contributor.

In addition to that, most of the 9,496 threads that received an answer were initiated by one of the 30 core developers, 6,418, or 67.59% of the threads that were answered to have been initiated by a member of the core developer group, while 3,078 (32.41%) were initiated by one of the 326 peripheral members. The core developers primarily ‘talk’ to members of their own group and 5,204 (54.80%) of the thread replies were written by another core developer. The peripheral members similarly prefer to answer the threads that were initiated by core developers (12.78% of their replies) and they tend to reply
with a lower intensity to those messages that were initiated by members of their own group (10.67%), see also the figure 3.10.

Figure 3.10: Thread initiation and thread reply in Freenet, year 2000.

In 2000, a total of 1,244 commits were added to the project’s code repository (von Krogh et al., 2003b). Since only core developers have direct CVS access, the members of this group have made all commits. Peripheral members without direct CVS access are still able to indirectly commit code by contacting the core developers. In the open questions survey, 11 of the 20 responding peripheral members report on no coding work at all, 4 peripheral members report on few coding work, and of the remaining 5 peripheral members, the answers were unclear. These findings indicate that most of the coding work is being completed by the core developers.

Similar things can be said of the core developers’ and peripheral members’ creation of general project documentation. On the year 2000 project website, I could exclusively find members of the core developer group as originators or authors of publicly available documents (such as, for instance, FAQ, project presentation, papers, Ian Clarke’s master thesis) describing the project’s aim, philosophy or work. Peripheral project members were
not at all involved in the publication of general project documents. Therefore, the more project-specific knowledge a contributor to an open source software development community continuously creates after his or her entry to the project, the more project-specific knowledge he will have, and the higher her position in the hierarchy will be. For additional case evidence see table 3.11.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Peripheral members (lower positions in the hierarchy)</th>
<th>Core developers (higher positions in the hierarchy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge absorption and creation post project entry</td>
<td>“Reading mostly, not a lot of time, probably a month or two at an hour or so a day,” (Peripheral member #11)</td>
<td>“Probably I average about an hour, between an hour and two hours a day, […] This summer, at the end of the summer, when me and core developer B were working on trying to implement the key types and everything else, I was probably working between six and eight hours a day on it, […] That really was my summer job and I was probably spending six or eight hours a day, seven days a week,” (Core developer C)</td>
</tr>
<tr>
<td></td>
<td>“All in all maybe a few tenths of hours, including all reading,” (Peripheral member #13)</td>
<td>“If you look back and take a look at the summer, I was contributing frequently, The summer I spent almost my entire day basically working on Freenet, So then I was doing maybe 10 hours a day, contributing all the time, So, it’s not a full time job, When I am going to school I have to look what I can do,” (Core developer B)</td>
</tr>
<tr>
<td></td>
<td>“Difficult to say, but probably less than an hour a day averaged over time, including reading/responding to email,” (Peripheral member #6)</td>
<td>“I would say anywhere from five to eight hours a day, […] — On average, it would be something like six hours a day, six or seven hours a day, […] Seven days a week, […] That has been maybe the last month and one-half or two months, […] Before that, I was spending maybe two or three hours a day on it, […] Right now, I’m devoting most of my time to Freenet,” (Core developer F)</td>
</tr>
<tr>
<td></td>
<td>“I spent about an hour a day on Freenet, I was tinkering with it, I was inserting and retrieving files, seeing how long they would last, etc, I also read most of the lists,” (Peripheral member #14)</td>
<td>“I’m thinking that July was design and some implementation, August was some heavy implementation, I know we spent probably 12 hours a day on it, Probably the first couple weeks in September as well,” (Core developer C)</td>
</tr>
<tr>
<td></td>
<td>“Probably about the equivalent of 2 months of effort, I spent about 10% reading code, 70% reading email, and the rest writing code,” (Peripheral member #8)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.11: Knowledge absorption & creation post project entry: Case evidence.
Point of time of entry & time span on the project. The FREENET project contributors have named their point of time of entry to the community and the longevity they had spent on the project as a third factor that influences the individuals’ accumulated amount of project-specific knowledge and which consequently also influences their position in the informal project hierarchy. The core developers, who consistently take in higher positions in the FREENET hierarchy, behave differently from the peripheral project contributors.

Firstly, the core developers entered the project on average at an earlier point of time during the project’s life cycle. Members of the core developer group entered the project by submitting their first email to the development mailing list on average on day 109.83 (April 18th, 2000; sd. 87.64) past FREENET’s initiation, Members of the peripheral contributor group, in comparison, did so 85 days later, on average on day 194.3 (July 14th, 2000; sd. 88.11), which characterizes an entry that is about twice as ‘late’ compared to that of the members of the core developer group.

Contributors to FREENET, who joined the project at an earlier point in time, spent more time on the project, thus being able to gain more familiarity with it. Early project contributors have been able to grow step-by-step along with the project’s progress, and thereby they have been able to grasp FREENET’s complexity in its entirety, in all of its details. Since many project-specific themes are repeatedly discussed on the mailing list, early project members are able to quickly realize new and relevant topics. As a consequence of being on FREENET longer, the project members have reported that core contributors are trusted more, and at the same time, they enjoy more decision rights such as, for instance, on the project’s direction. Later entrants must first grasp why and how things work in the project, and they must demonstrate their capabilities and willingness to contribute to the collaborative work. Therefore, the earlier (during a community’s life cycle) a contributor enters a community, the more project-specific knowledge he or she will have, and the higher his or her position will be in the informal hierarchy.

Secondly, the core developers contribute to the FREENET project as a rather stable working force, since they stay on the project for a longer time span compared to the members of the peripheral member group. In 2000, the core developers’ average project longevity was 185.97 days (sd. 131.56), while members of the peripheral contributor
group stayed on average 24.84 days (sd. 53.93) on the project, which is about one eighth of the core developers’ project longevity. As a consequence, the core developers have (again) more time available to dig deeply into the project specifics and to absorb the knowledge that resides in Freenet.

Members of the peripheral contributor group, who are for the most part located in lower ranked positions in the informal hierarchy, have not been able to spend a comparable longevity on the project, and consequently, have less time for absorbing knowledge of how Freenet functions. This creates a temporal disadvantage. Therefore, the longer a contributor is a member of an open source project, the more project-specific knowledge he or she will have, and the higher will be the position in the emerging hierarchy. For additional case evidence see table 3.12.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Peripheral members (lower positions in the hierarchy)</th>
<th>Core developers (higher positions in the hierarchy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point of time of entry to a community &amp; longevity on the project</td>
<td>“The core developer base was pretty solid at the time I joined. There seemed to be something of a control issue; people who had been around more had more to say in terms of the project direction, But again this is standard in most organizations.” (Peripheral member #8) “People who were more active or were elders in the project seemed to have more control over its direction. They were more trusted.” (Peripheral member #8)</td>
<td>“I have been on the project longer and most of the code […] is written by me now. I’m the one who has the most in my head of about how that part of Freenet is working.” (Core developer B) “I think too late in the sense they may have done a lot of the work I have been working on and the learning curve would have been steeper because the system has gotten much more complicated in the last eight months and I’m not sure I would have been able to get my shoe in the door with that small patch I submitted, because it wouldn’t have been necessary at that point.” (Core developer C) “It happened to core developer K to a degree. Basically, core developers B and C came along and they were capable of spending more time than he was and it did kind of become confusing and he realized that he really wasn’t able to keep up with the stuff that core developers B and C were doing and so he kind of dropped out,” (Core developer A)</td>
</tr>
</tbody>
</table>

Table 3.12: Point of time of entry to the project and longevity on the project: Case evidence.
2.2.6. Entry, exit and mobility barriers

Project-specific knowledge in the Freenet open source project continuously accumulates, changes and becomes more complex. By virtue of the contributors’ ongoing work, it takes considerably more time and effort for every interested person to recapitulate and absorb previously created knowledge of the project. The quality and quantity of project-specific knowledge that resides in the Freenet project thus creates barriers to the members’ promotion within and across the project’s permeable and fluid boundaries. Knowledge, therefore, creates entry, exit as well as horizontal or vertical mobility barriers, as illustrated below (figure 3.13).

![Diagram of entry, exit and mobility barriers]

Figure 3.13: Entry, exit and mobility barriers.

As defined for this study, potential contributors officially enter the Freenet project by submitting their first email to the developer mailing list, which signals their interest and availability to a larger member audience. Submitting emails is open to any volunteer who enrolls to the list, and no other formal entry barrier or recruiting process exists for contributing to the project. Since interested people seem to ‘hang around,’ ready to contribute to Freenet, the project exhibits no lack of talented developers who possess the relevant project-related skills or capabilities. Many potential contributors are software developers who could otherwise not be hired legally, since they are not looking for a full or part time job, or since they are already employed by a firm. Not all individuals who are interested in contributing to Freenet, however, successfully enter the project, since
they must first overcome an entry barrier to the project. At the same time, actual project members cannot freely wander between the project’s hierarchical levels or between their individual specializations in FREENET. The contributors reported four different types of barriers in their community.

These barriers separate, firstly, outsiders - people who observe the project (lurkers), and occasional contributors from those who are capable of contributing on a regular and longer-term basis. I will consider this as an entry barrier. Secondly, people who once devoted their efforts to FREENET hesitate to permanently leave the project and stop contributing, which I will interpret as an exit barrier. Thirdly, contributors to FREENET cannot freely decide and wander between the project’s hierarchical levels, which I will consider as a vertical mobility barrier. The most obvious vertical mobility barrier exists between members of the peripheral contributor group and members of the core developer group, but vertical mobility barriers are not limited to that. Finally, on an identical hierarchical level, the FREENET contributors are separated by their individual fields of specialization, which I will interpret as horizontal mobility barriers.

**Entry barrier & contributor selection.** Before interested people are capable of frequently contributing to FREENET, they must first absorb the project’s very fundamentals, for example by lurking. The contributors have reported that they start to take those newcomers seriously, who make intelligent contributions of any kind to the project. Individuals who announce their interests, but who give no testimonial of their knowledge, have difficulties of becoming recognized as a full project member. The contributors have reported about FREENET’s entry barriers,

“Entry barriers would be knowledge, not structural. Key requirements would be knowledge of Java, networking, crypto, Freenet.” (Peripheral member #14)

“There are some conceptual hurdles to overcome to really understand Freenet. Newer contributors often misunderstand some important aspects of the project.”
(Peripheral member #6)

“The barrier of entry into the project is a little high. A little higher than a lot of people would like,” (Core developer B)
<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry barriers</td>
<td>“It’s the higher barriers of entry. You have to be able to contribute. To contribute to the project you really need to have an understanding of the technology and you really need to make an effort in understanding it, compared with when you have something easy visible to go to, to shoot for,” (Core developer B)</td>
</tr>
<tr>
<td></td>
<td>“The code base isn’t that small... there is a decent barrier to entry for new developers. The design patterns are different in all software projects. It takes some time to get used to them. Luckily I didn’t have to do any core modifications.” (Peripheral member #3)</td>
</tr>
<tr>
<td></td>
<td>“I think it is actually quite a high barrier to entry to try to understand it all. There are thousands of messages there. There is quite a lot of development talk on there, so it is pretty technical. I think you could argue that there’s a lot of talk and not much being done, but a lot of it is understanding, which there’s a lot of that needed on something like Freenet.” (Core developer E)</td>
</tr>
<tr>
<td></td>
<td>“The biggest barrier to new developers is dedication to the project. Learning about the project and ultimately giving patches and new features takes time. This wears out folks that only have a fleeting interest in the project, so in the end are those that are really dedicated to the project.” (Peripheral member #3)</td>
</tr>
<tr>
<td></td>
<td>“One really negative point that makes the entry barriers really, really high is the lack of documentation. Because you have the code. And people asking for help are often referred to ‘Okay, the answer is in the code.’” (Core developer H)</td>
</tr>
<tr>
<td>New contributor selection</td>
<td>“I don’t think we really go outside. We don’t open the door and look for people except for when the problem is big enough that we don’t think we can handle it ourselves,” (Core developer C)</td>
</tr>
<tr>
<td></td>
<td>“We’ve got a couple of people who wouldn’t be legal to hire,” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“It is really interesting in that we don’t generally have to look for people with the relevant experience. Usually they are interested and so they’re already hanging around, or someone who has relevant experience is interesting or hanging around, or interested in a related project that we can refer to,” (Core developer F)</td>
</tr>
<tr>
<td>Learning curve</td>
<td>“The learning curve is the hardest part of getting into a project. I noticed on the main Freenet core team, a new member would only really start to pick up steam after about two months. The older members also had a clearer idea of what the project wanted and had to keep telling the new members that ideas they had seen before wouldn’t work and why.” (Peripheral member #3)</td>
</tr>
<tr>
<td></td>
<td>“Too late in the sense they may have done a lot of the work I have been working on and the learning curve would have been steeper because the system has gotten much more complicated in the last eight months and I’m not sure I would have been able to get my shoe in the door with that small patch I submitted, because it wouldn’t have been necessary at that point.” (Core developer C)</td>
</tr>
</tbody>
</table>

Table 3.14: Entry barriers and new contributor selection: Case evidence.
Prospective project members go through a selection process in which they are informally tested, *on the one hand*, their contributions demonstrate their *willingness, interest and commitment* to the project, and *on the other hand*, their contributions reflect their *ability* to contribute to *FREENET*. Since no formal project documentation on the project exists, absorbing previously created knowledge of *FREENET* demands considerable investments of time and effort by the interested developers. Those who are willing to invest the necessary time, must also demonstrate their learning efforts to the other project members and actively contribute to the project. The project’s entry barriers give people with superficial, fleeting interest, or with limited dedication to the collective work only a small chance to be recognized as a full project member.

*FREENET’s* knowledge-based entry barrier serves therefore as an informal and implicit selection mechanism, which separates individuals who are dedicated to the project from those who are not. Those individuals who are willing to invest considerable time and effort in absorbing the project’s accumulated knowledge base, are likely to contribute to the project in the future. At the same time, the project’s entry barriers continuously rise over time, since the accumulated project knowledge base a prospective member must capture to be able to contribute, rises as well (For additional case evidence see figure 3.14).

*Exit barrier*, The philosophy of open source projects is based on volunteers who come and go, and no formal process exists through which people announce if they want to leave a project to which they contribute, It was probably for this reason that I was not able to find a single announcement of a contributor who left the project on the *FREENET 2000* development mailing list, To contribute to a certain project, the developers first must absorb knowledge of it which is project-specific and of only limited use to other software development work. Their investments thus represent both sunk costs and a barrier for leaving this community. The *FREENET* contributors reported on their project-specific investments for acquiring the necessary knowledge in terms of *time* and *intellectual effort*, as well as on their *emotional investments*, The latter come into existence through the contributors’ strong identification with the project of their choice, and through a sense of responsibility for their specialized work. With the members’
increasing personal investments in the project, their individual barriers for leaving the community rise as well.

Since peripheral project contributors and core developers hold on average different investments in the project, their personal barriers for leaving differ as well. Peripheral members invest on average less time and less effort in the project, and they remain there only for a shorter period of time. Their willingness to permanently leave the project is consequently greater, compared to the core developers. The peripheral members’ reasons for leaving FREENET range from ‘time constraints’ to ‘feeling useless in the project’ and ‘not being able to contribute to its progress,’ or, the peripheral members contribute work that receives no feedback, that becomes ignored and that is not integrated into the project’s ongoing progress.

The core developers have on average greater investments in FREENET, which hinder them from easily giving up their prior work. They claimed that they left the project only if there existed some serious reasons to do so, for example, if the project had no more challenging work to offer to them. Or, if they found the project’s evolving vision stupid and something to which it was not worth contributing. If a core developer left the project it would be of greater visibility and significance to the other contributors, since their names tend to appear more often on the websites. Also, since their submitted contributions to the project contain only the explicit part of their work but not the contributors’ implicit knowledge, much of the project-specific knowledge vanished from the project, no longer being accessible to others.

It is for this reason that the core developers hypothesized that if one of the upper-level core members left the project, the other members of that group would ask him or her for the underlying reasons thereof or, they would complain about the decision to leave (see also table 3.15). I conclude that the more project-specific knowledge a contributor to an open source software development project has absorbed and created, the higher will be his or her position in the hierarchy, and the higher will be his personal exit barrier to the project.
<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit barriers</td>
<td>“We’ve had some people who contributed before and just stopped contributing, yes, I don’t think we’re really interested in people dropping out, no, But I suppose if I’d just stopped doing any work they’d look it up, […] People come and go, that’s the nature of that, the whole free flowing thing of open source,” (Core developer B)</td>
</tr>
<tr>
<td></td>
<td>“I would have to have something pretty significant to pull me away, The main[exit] barrier is that personally, I’ve taken on some responsibility for the project, so I’d have to transfer that responsibility, Whenever you transfer responsibility for a particular task, the task isn’t going to get done the same way that you did it, I would have to let go of some of my ideas about how things are supposed to work and say, ‘That’s going to be OK, if this other person does this task, They do it differently from how I would do it, and I can live with that,’ It would take some kind of emotional and personal dis-involvement from the work at hand to do that, I think that almost all of the barriers would be personal, for me, […] I think there might be some resistance there, Another thing would be people saying, ‘None of us understands what you’ve been doing, You can’t go now, You have to write the documentation, and tell us what’s going on,” (Core developer J)</td>
</tr>
<tr>
<td></td>
<td>“I have no plans to drop my commitment, What is important is to stay relevant and that we don’t do something extremely stupid, That’s the only thing, If there had to be something extremely stupid, we’d just drop support,” (Core developer B)</td>
</tr>
<tr>
<td></td>
<td>“I stopped contributing because I stopped being interested and feeling useful, After the project exploded there were more than enough people with their hands in the soup, and I didn’t feel like putting in much more effort to stay afloat as a visible element in discussion, I’m not intending to reentering the project because I have much less free time, more interests of my own, and the project doesn’t seem to really need more help,” (Peripheral member #10)</td>
</tr>
<tr>
<td></td>
<td>“I stopped contributing to Freenet due to my belief that the project wasn’t living up to expectations, After reviewing the design of the system, I didn’t think it was going to go as far as I wanted in terms of usability and scalability so I was less motivated to contribute anything, Also I started running out of free time,” (Peripheral member #8)</td>
</tr>
</tbody>
</table>

Table 3.15: Exit barriers: Case evidence.

**Horizontal mobility barriers.** Project-specific knowledge creates, **thirdly,** horizontal mobility barriers which hinder the project contributors from frequently wandering between their personal specializations on the identical hierarchical levels. The most obvious horizontal mobility barriers exist between the members of the core developer group or, between the members of the peripheral member group, but they are not limited to these. Specialization in the project requires that these contributors have in-depth knowledge about a certain task, and only few project members are reported to have it. Those who had created or found their own niche and specialization in the project were able to accumulate all the necessary knowledge related to a certain task, which is
only in part explicitly available and accessible to others. Those who are interested in a certain project specialization must first absorb the necessary knowledge that is related to the field of specialization, and the respective knowledge might also accumulate over time. Von Krogh et al. (2003b) in their study of the Freenet project reported of highly specialized behavior, and about 80% of all CVS files were created and/or modified by a maximum of 2 developers.

At the same time, members of both the core developer and the peripheral member group simultaneously report that in a certain field of specialization there is only limited capacity for a certain number of contributors who would like to work on the task in question. As soon as the field’s limit in capacity is reached, interested contributors are forced to wait for the tasks’ future availability, which can be guaranteed only if the responsible project member decided to leave his post. Although the procedure occurs in an entirely informal manner, it is similar to the succession in a company setting.

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal mobility barriers</td>
<td>“Now — this [task] has actually been taken over by core developer F,” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“What’s interesting is that if one, for example, core developer K, the original guy who volunteered to help me out, started to drop off in his interest and then somebody else rose up to take his place.” (Core developer A)</td>
</tr>
<tr>
<td></td>
<td>“There’s rough consensus that if there’s a problem with the Windows installer and the Windows code, ‘Talk to core developer H about it, He’s the one who’s been working on that for a while, He’ll be able to give you a hint in making the change, He’ll make the change himself,’ […] He’s the person that’s tending to do those changes.” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“What I have found is that if any of them drop out somebody else will rise to the surface to take his place,” (Core developer A)</td>
</tr>
</tbody>
</table>

Table 3.16: Horizontal mobility barriers: Case evidence.

Project members who wish to work on a new project specialization have the best chances to do so if they created their own niche that no one else yet occupied, or, to which no other project member was able to or wished to contribute. Core developer E, for instance, reported that he started to work on a GUI (general user interface) for FREENET, a task in which no other contributor was interested. Working on a certain field of specialization or creating it has a great advantage over being a ‘generalist,’ somebody
who has an overview of the entire project, and who is capable of contributing in a broad variety of fields. Since working on a certain field of specialization requires rather deep, but not necessarily broad project knowledge, specialization shifts the horizontal mobility barrier (or, alternatively the entry barrier) downwards (see table 3.16).

**Vertical mobility barriers.** Project-specific knowledge finally creates vertical mobility barriers in FREENET that separate the project’s contributors of various informal hierarchical levels (also; von Krogh et al., 2003b). Again, the most obvious vertical mobility barrier exists between the members of the core developer group and the members of the peripheral contributor group. No formal process of vertical promotion exists for the project, and the core developers have described the process of becoming a core developer as ‘organic’. To rise from the peripheral member group to the core developer group, the contributors must be granted direct access to the FREENET code repository, which only a handful of core developers are able to provide. To be granted access to the CVS is, at the same time, a crucial step for the project member, and in 2000, none of the core developers was taken away the access to the code repository. To become a core developer characterizes the permanent promotion to a higher hierarchical level in the project, and only few contributors actually become core developers in FREENET.

Besides the individuals’ project-specific knowledge (their point of entry to the project, their knowledge creation and absorption), the availability of posts in the project is also crucial for the contributors’ vertical promotion, similar to the horizontal mobility in FREENET. One member of the upper-level core developer group has claimed, for example, that there is essentially only capacity for a handful of ‘true’ or, upper-level core developers,

"There is only room for three or four core developers and so if one of them dropped out it would kind of leave room for someone else to take their place and it wouldn’t be any official ceremony or anything. It would just happen organically." (Core developer A)

The limited number of available posts becomes clear by the small number of core developers who are active in the project. In 2000, although the accumulated number of (core and peripheral) project contributors constantly rose, the number of core developers who
posted to the list or who submitted code to the repository, remained stable. The absolute number of people who posted at least one email to the list increased rapidly during the project’s initial months in 2000 before it evened out on a monthly average of between 60 and 80 posters (average number of posters per month in 2000: 60.67, sd. 24.12, see also figure 3.17). In 2000, over the year the average monthly number of active core developers who submitted at least one message to the list was only 13.83 (sd. 4.04), and the average monthly number of core developers who made an entry to the CVS was 8.33 (sd. 2.27).

The core developers illustrated how the vertical mobility barrier rose over time due to the permanently accumulating project-specific knowledge. Early project entrants reported that they were granted access to the code repository through an easygoing process. They reported on how they had posted a message to the mailing list, or how they wrote a private email to one of the key people and so asked for their access to the CVS, which they were granted right away. Besides that, any other demonstration of what they knew on the project was not necessary, such as in the case of core developer I,

“...I wrote this message to the mailing list saying ‘Hi I’m I, I’ve been programming Java and C a long time. I know a bit about Java.’ And so forth, things along these lines, I didn’t really say anything in particular that I wanted to work on because at this point I didn’t really know exactly what was being developed and was sort of in need of work. So I just put up this general message and I actually wasn’t really expecting to become a core developer right away. So I think core developer A just put me in because there weren’t that many developers at that time.” (Core developer I)

Over time, the policy, through which peripheral members became core developers changed. The reason for this was that the contributors had to cope with a growing number of people who claimed to be interested in the project. Moreover, not all contributors who were granted access to the code repository in fact started to code for FREENET, and their behaviors characterized a decrease in transparency for the members of the core developer group. With their behavior of not contributing to the code base, these developers violated the unwritten community rules, and they did not demonstrate their project-specific knowledge or coding skills as it was at least informally expected from them.
Figure 3.17: Absolute number of Freenet contributors (core developers and peripheral members) who contributed at least once to the mailing list in month x of 2000.\textsuperscript{10}

Since that time, the policy by which peripheral members are granted access to the code repository changed. Interested people first have to demonstrate their willingness and ability to contribute by way of submitting code. They first have to deliver code patches to the actual core developers, who serve as gatekeepers and who put in whatever they feel fits into the existing code base. \textit{On the one hand}, the members’ willingness and ability to deliver code that fits into FREENET separates members of the hierarchically lower positioned peripheral group from those of the higher positioned core developer group. \textit{On the other hand}, submitting code is not enough for becoming a core developer. Interested contributors must additionally demonstrate that they understand and respect the rules that hold the project together. One contributor has recalled,

"\textit{To become a core developer you must submit patches to improve the software and get to the point where there is little disagreement whether your fixes or features are worth adding. Once you commit access, you need to continue sending patches when you work on other parts you don’t normally work on, to show that you understand delegation and respect the internal structure of the group. Having now demonstrated skill and respect you need to demonstrate leadership by coming up with an idea to pursue and implement or taking}"

206
**action and fixing a problem that’s already been identified. When you’ve done this you’ve gotten to the point where your contributions are essential to moving the project forward and I qualify this as a ‘core developer.’” (Peripheral member #10)**

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Case evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vertical mobility barriers</strong></td>
<td>“Absolutely [there is vertical promotion possible, To do so, one had] to consistently be making good, solid changes. Basically you can rise as high as your commitment is. Yes, that [horizontal promotion] happens, too. Generally when there’s a vacancy or someone drops out, etc.” (Peripheral member #16)</td>
</tr>
<tr>
<td></td>
<td>“The question that people ask themselves is, ‘Can I make a difference?’” (Core developer A)</td>
</tr>
<tr>
<td></td>
<td>“For Freenet, what gets you into the project is that you’ve got good work or good ideas that you’ve already contributed. Almost unanimously, the people that I’ve seen join after I joined have been people who say, ‘I’ve done this work already, I’ve made it work, I’ve submitted it to someone who’s already in the project. It works, It’s getting to be a hassle for me to continue submitting patches or documentation, so give me CVS access, and let me start doing it.’ Usually, it’s said, ‘It’s easier for us, and it’s easier for you, so let’s get you in.’ Conversely, there’s another kind of person.” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“The primary mobility barriers were the initial elitism that could be won over, but the biggest barrier is a barrier that I think is essential for the operation of projects; You have to actually do something good to be recognized, And as more work gets done that is harder to do, and rightly so because even though it’s a hobby, OSS is not some club to just play around in. You can make your own playhouse but some people are trying to accomplish something.” (Peripheral member #10)</td>
</tr>
<tr>
<td></td>
<td>“We get these messages from people posting messages fairly similar to mine. But then, nothing would ever come of them, like they would be added as developers and they would never contribute any code. I think, eventually what core developer B started doing was telling people just to post patches, to post their code to the mailing list and then someone else would incorporate it into the code.” (Core developer I)</td>
</tr>
<tr>
<td></td>
<td>“A lot of people come and say that they want to contribute, But then we try to be cordial in the beginning, saying that there is so much to do, we are trying to be helpful, But so little came out of it, that I have become a little cynical with people approaching us, I usually say, ‘come back when you have some code,’ The code is publicly available, People don’t have to ask, they just check it out and see if there is something they can do.” (Core developer B)</td>
</tr>
<tr>
<td></td>
<td>“The way it happens [getting CVS access] is that someone is making contributions. It gets to a point where they’re sending contributions to mailing lists or another individual developer so much that either that developer or one of the managers says, ‘We need to get this person their own access,’ In a lot of ways, it’s more about willing to do work than it is about the quality of the work that you do, If the work is substandard or not done well, it’s not going to get in, If you send garbage to people, they’re not even going to bother adding stuff. If you send stuff that works, they’re going to like it.” (Core developer I)</td>
</tr>
</tbody>
</table>

Table 3.18: Vertical mobility barriers; Case evidence.
As a consequence of the limited number of available posts in the core developer group, potential developers are asked to act strategically. They are expected to signal that they have something valuable to contribute to the project and that they are consequently a gain for the entire community. This finding was also reported by von Krogh and colleagues (2003b), who termed such a contribution a ‘feature gift’. People who show that they can contribute some unique knowledge to FREENET thereby signal that they have something valuable to contribute to the project. Valuable project contributions must not necessarily be outstanding coding skills.

One core developer, for instance, a PhD student who is familiar with conceptual and analytical thinking, discovered that FREENET was evolving quickly, and that Ian Clarke’s master thesis which had initiated the project, was by no means sufficient to describe the project’s vague direction. Core developer I started to summarize the project’s vision in a number of published or unpublished papers (Hong, 2001), since he discovered that this task characterized work that he was best able to contribute to the community. As a result of the other contributors not wanting to work on any type of project documentation, core developer I’s contribution was unique and therefore served as a welcomed addition to the project.

The following paragraphs will summarize the findings of the first order analysis in a more general way and I will propose a model of a knowledge hierarchy.
23. Second Order Analysis:

Knowledge Hierarchy Definition and Model

In the following paragraphs I will summarize the first-order descriptions in a more generalizing way. I will introduce the multidimensional construct ‘knowledge hierarchy’. Furthermore, I will present both a model of factors that influence an individual’s position in the hierarchy and hypotheses on the identified causal relationships. Although no formal authority-based hierarchy exists in open source software development communities, the contributors structure their organizations informally.

Knowledge determines hierarchy. The informal hierarchical structure that is prevailing in FRENET was found to be grounded in how much an individual knows about his organization, and the more knowledge an individual has, the higher will be his or her position in the informal hierarchy. Since the individuals’ knowledge is organization-specific, it refers primarily to this specific organization, but not necessarily to others. The informal hierarchy that is based on organization-specific knowledge brings individuals possessing certain qualities together at the top of the hierarchy. These individuals are usually the ones who have the best overview of their own organization, who are supposed to be capable of meeting crucial decisions best, and who should therefore complete the most important work.

In essence, the informal hierarchy manages to realize what a formal authority-based hierarchy originally was supposed to achieve (Galbraith, 1973; Mintzberg, 1979; Nadler & Tushman, 1988). Contrary to a firm setting, community members contribute their efforts anonymously and they are reduced to their working style and the work’s content, while other factors, such as clothing, looks, personal contacts or sympathy, are widely irrelevant.

Knowledge accumulation over time. Both the individuals’ organization-specific knowledge and the total amount of knowledge that was created and/or is stored in an organization are subject to modification, since the knowledge accumulates, changes, or it becomes increasingly complex over time. The value of an individuals’ knowledge at a certain point in time is, therefore, only valuable in relation to the organizations’ entire accumulated knowledge base. The greater an individual’s share of the organization’s
accumulated knowledge base, the higher will be his or her position in the informal knowledge hierarchy (*hypothesis 1*).

**Definition of a knowledge hierarchy.** I will term an informal hierarchy that is grounded in organization-specific knowledge, such as knowledge content, knowledge about the organization’s architecture or on its rules & standards a ‘knowledge hierarchy’, and define it as follows,

A ‘knowledge hierarchy’ is a structuring mechanism in which organizational members can achieve a temporary position of informal subordination that grants the individuals’ inequalities in the organization such as diversity of rank, decision rights, reputation, attention, or diversity of their degrees of freedom to act. An individual’s position in the knowledge hierarchy is influenced by how much organization-specific knowledge he or she has at a certain point in time.

**Influencing factors.** The volume of individual to organizational project-specific knowledge at a certain point in time depends on both individual and organizational level factors. On an **individual level**, three factors influence an organizational member’s position in the knowledge hierarchy. An individual in the organization is thereby no passive subject; rather, he or she is capable of playing with and influencing these factors. Organizational members are consequently capable of influencing their portion of accumulated organization-specific knowledge and thereby indirectly influencing their own position in the knowledge hierarchy.

Individual-level factors are, **firstly**, the amount of knowledge an individual absorbs before his or her entry into the organization. The factor determines how much an individual knows at the time of his project entry, and it should influence an individual’s position that is reached directly after his entry to the project (*hypotheses 2a&b and 5a&b*). **Secondly**, an individual’s organization-specific knowledge is influenced by his or her **continuous involvement** with the organization while simultaneously being a member of it. His or her involvement refers to both the individual’s passive knowledge absorption and active knowledge creation behavior while being a part of the organization (*hypotheses 3a&b and 6a&b*). **Finally**, an individual’s amount of organization-specific knowledge is influenced by the **time span** he or she spends on the project. The time span refers, on the
one hand, to the point in time at which an individual enters a community, and on the other hand, to the longevity of the individuals tenure with the organization (hypotheses 4a&b and 7a&b).

On an organizational level, the volume of individual to organizational project-specific knowledge at a certain point of time depends on two additional factors. The organizational-level factors influence how much knowledge is accumulated in the entire community, and an individual member only has control over these external factors to a limited degree. The organization-level factors are, firstly, the focal organization’s age, which reflects the community’s stage in its life cycle. At the same time, the organizational age mirrors how much organization-specific knowledge could be accumulated since the organization’s initiation, which should be more for an older community, and less for a younger organization (hypothesis 8). The second organization-level factor is an organization’s size, since a larger organization is supposed to have accumulated more organization-specific knowledge than a smaller one (hypothesis 9).

A number of other factors not mentioned so far, but which are plausible, may additionally influence an individual’s position in an open source community’s informal hierarchy. From private talks with developers of a number of projects and with experts in the field, it has become clear that, firstly, a person’s reputation in the entire open source community across all existing projects strongly influences an individuals’ position in a project. If Linus Torvalds, the Linux founder who is, at the same time, a cult figure in the community, wanted to enter a certain open source project, he would immediately be granted core developer status without the need for demonstrating his proper organization-specific knowledge.

Secondly, the contributions that individuals post to a certain open source project are not always directly related to the communal (project) work. Some contributors, for instance, post poems on the mailing lists, or they complain about firms that produce software in a profit-oriented environment. While the open source contributors tolerate the latter, poems that are not work-related are not tolerated. As a consequence, an individual’s involvement with a community refers not to the creation and absorption of any kind of project-specific knowledge, but to what kind of knowledge he or she contributes.
Although both factors are most plausible for the context of this work, I did not further integrate them to the knowledge hierarchy model or the quantitative analysis. The reasons for this are, on the one hand, that an inclusion of both factors would dramatically increase the complexity of this pilot study on an informal hierarchy in open source communities. On the other hand, both factors did not directly emerge out of the FREENET case, on which the first order analysis of this work is based. Both factors, however, could be subject to future studies in this field of research.

**Knowledge hierarchy model and hypotheses.** I will summarize my findings and the implicitly or explicitly discussed hypotheses of the preceding paragraphs in a model of a knowledge hierarchy in (open source) communities as illustrated below (figure 3.19). The listed hypotheses refer, firstly, to a direct, un-mediated relationship between independent and dependent variables (hypotheses 2 to 4). Secondly, the hypotheses refer to relationships that are mediated by the individuals’ share of the community’s accumulated knowledge base (hypotheses 1 & 5 to 9). All proposed relationships are positive, and in all hypotheses ‘knowledge’ refers to ‘organization-specific knowledge’.

![Knowledge hierarchy model](image)

**Figure 3.19:** Knowledge hierarchy model.
Community-specific knowledge, dynamics and knowledge accumulation

H1: The greater the share of a contributor’s accumulated amount of knowledge of the total amount of accumulated knowledge that resides in an OSS project at a certain point of time, the higher will be his or her position in the knowledge hierarchy.

Individuals’ knowledge absorption before entry to the community

H2a: The more time a prospective contributor to an OSS community spends on absorbing knowledge before his or her entry to the organization, the higher will be his (immediate) position in the knowledge hierarchy. *(direct relationship)*

H2b: The more types of knowledge transfer media a prospective contributor to an OSS project uses in order to absorb knowledge before his or her entry to the organization, the higher will be his (immediate) position in the knowledge hierarchy. *(direct relationship)*

H5a: The more time a prospective contributor to an OSS community spends on absorbing knowledge before his or her entry to the organization, the greater (deeper) will be his share of individual knowledge of the total amount of knowledge that is accumulated in the organization, and the higher will be her (immediate) position in the knowledge hierarchy. *(mediated relationship)*

H5b: The more types of knowledge transfer media a prospective contributor to an OSS project uses to absorb knowledge before his or her entry to the organization, the greater (broader) will be his share of individual knowledge of the total amount of knowledge that is accumulated in the organization, and the higher will be her (immediate) position in the knowledge hierarchy. *(mediated relationship)*

Individuals’ continuous involvement to the community

H3a: The more knowledge a contributor to an OSS community continuously absorbs after his entry to the project, the higher will be her position in the knowledge hierarchy. *(direct relationship)*
H3b: The more knowledge a contributor to an OSS community *continuously creates after his entry* to the project, the higher will be her position in the knowledge hierarchy, *(direct relationship)*

H6a: The more knowledge a contributor to an OSS community *continuously absorbs after his entry* to the project, the greater will be her share of individual knowledge of the total amount of knowledge that is accumulated in the organization, and the higher will be his position in the knowledge hierarchy, *(mediated relationship)*

H6b: The more knowledge a contributor to an OSS community *continuously creates after his entry* to the project, the greater will be his share of individual knowledge of the total amount of knowledge that is accumulated in the organization, and the higher will be her position in the knowledge hierarchy, *(mediated relationship)*

**Point of time of entry and longevity on the community**

H4a: The *earlier* in an OSS community’s life cycle, a contributor *enters a project*, the higher will be her position in the knowledge hierarchy. *(direct relationship)*

H4b: The *longer* a contributor to an OSS community *is active on the project*, the higher will be her position in the knowledge hierarchy, *(direct relationship)*

H7a: The *earlier* a contributor *enters a project* in an OSS community’s life cycle, the greater will be his share of individual knowledge of the total amount of knowledge that is accumulated in the organization, and the higher will be her position in the knowledge hierarchy, *(mediated relationship)*

H7b: The *longer* a contributor to an OSS community *is active on the project*, the greater will be his or her share of individual knowledge of the total amount of knowledge that is accumulated in the organization, and the higher will be her position in the knowledge hierarchy, *(mediated relationship)*
**Organizational age and size**

H8: The older an OSS community becomes, the greater its accumulated project-specific knowledge base will be.

H9: The larger an OSS community becomes, the greater its accumulated project-specific knowledge base will be.

*Why does a knowledge hierarchy make sense in (open source) communities?* Organization-specific knowledge as origin of an informal hierarchy in (open source) communities serves as a simplifying mechanism that structures the otherwise open, boundary-less and ambiguous setting. An informal hierarchy implicitly leads to a (value free?) categorization of the community’s members and thereby enables for a minimum of informal or even implicit planning of the collective work. The informal mechanism separates between different types of contributors, who are treated differently and who enjoy more or less rights with respect to the category to which they belong. The emerging structure helps the contributors distinguish between participants who are likely to deliver the work that is desired, and those contributors for whom the likelihood is low or unknown.

The contributors, *firstly*, signal through their activities that they are *available and willing* to continuously deliver contributions to the collective work; *secondly*, they demonstrate their *capabilities* for advancing the collective work in a reliable way. *Thirdly*, the contributors communicate that they will *produce coherent and coordinated work* that fits into the organization’s context without questioning its fundamentals. A hierarchy that is based on the contributors’ organization-specific knowledge is no entirely certain indicator for the contributors’ work, since it refers to the past and bears the danger of overlooking participants who could deliver valuable ideas in the future. In an otherwise unstructured setting, the hierarchy makes it possible for an organization to achieve a minimum of certainty, which is necessary to advance the collective work.

In the following paragraphs, I will give an overview of the methods used to (partly) test the knowledge hierarchy model, and of the findings that resulted from the analysis.
3. Model Testing

3.1 Methods

The second part of this study aims at deductively testing both the previously generated hypotheses and the model of a knowledge hierarchy in open source software communities. Quantitative data were gathered, firstly, by means of a web-based survey that was filled in by members of four open source software development communities. The analysis is, secondly, based on publicly accessible archival data of two open source projects, with the members of both projects also having filled in the survey. The archival data include the projects’ developer mailing lists, and their code repositories. I will give a brief overview of the community sampling, variable operationalization, on data sources and analysis of the data in the following.

3.1.1. Community sampling

I sampled four communities that are listed on the SOURCEFORGE website (sourceforge.net, a host for open source communities), CRYSTALSPACE, HSQLDB, STEPMANIA, TikiWiki. Each of the four communities suffices the following sampling criteria, firstly, all communities are mid-sized, and they are smaller than the very large projects such as LINUX or APACHE with thousands of contributors, but all communities are large enough so that the researcher can discover patterns within the contributor community. All four communities had at least 60 core developers at the time of study, which was necessary to guarantee for some variance between the contributing individuals. An average open source project usually does not lack peripheral contributors, but often they have only a small number of core developers.

Secondly, all communities can be considered as active projects, and at the time of study in November 2003 each of the four projects ranged among SOURCEFORGE’s 100 most active communities, Thirdly, all sampled communities at the time of study existed for at least one year, which was necessary to make sure that informal structural or behavioral patterns had enough time to emerge and mature.
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>CrystalSpace</th>
<th>HSQLDB</th>
<th>Stepmania</th>
<th>TikiWiki</th>
</tr>
</thead>
<tbody>
<tr>
<td>Website</td>
<td>Crystal, source forge.net</td>
<td>Hsqldb, source forge.net</td>
<td><a href="http://www.stepmania.com">www.stepmania.com</a></td>
<td>Tikiwiki.org</td>
</tr>
<tr>
<td>Purpose</td>
<td>3D game</td>
<td>Relational database engine</td>
<td>Music/rhythm game</td>
<td>Content management system (CMS) and groupware</td>
</tr>
<tr>
<td>License</td>
<td>GNU library or LGPL</td>
<td>BSD license</td>
<td>GNU GPL</td>
<td>GNU library or LGPL</td>
</tr>
<tr>
<td>Sourceforge activity percentile</td>
<td>97.267%</td>
<td>99.369%</td>
<td>99.175%</td>
<td>99.87%</td>
</tr>
<tr>
<td>Programming language</td>
<td>C++</td>
<td>Java</td>
<td>C++</td>
<td>PHP</td>
</tr>
<tr>
<td>No, of core developers</td>
<td>60</td>
<td>83</td>
<td>107</td>
<td>158</td>
</tr>
<tr>
<td>No, of peripheral members</td>
<td>-</td>
<td>-</td>
<td>521</td>
<td>148</td>
</tr>
<tr>
<td>Total no, of contributors</td>
<td>-</td>
<td>-</td>
<td>627</td>
<td>306</td>
</tr>
<tr>
<td>Other</td>
<td>-</td>
<td>Origin was Hypersonic SQL, a closed OSS project.</td>
<td>Extensive core developer presentation.</td>
<td>Sourceforge project of the month on July 2003, extensive project documentation available,</td>
</tr>
<tr>
<td>Data gathered</td>
<td>survey</td>
<td>survey</td>
<td>survey, archival data</td>
<td>survey, archival data</td>
</tr>
<tr>
<td>No, of survey responses</td>
<td>14</td>
<td>4</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>Core developers among respondents</td>
<td>12 (85.71%)</td>
<td>2 (50%)</td>
<td>13 (86.67%)</td>
<td>15 (88.24%)</td>
</tr>
<tr>
<td>Peripheral members among respondents</td>
<td>2 (14.92%)</td>
<td>2 (50%)</td>
<td>2 (13.33%)</td>
<td>2 (11.76%)</td>
</tr>
<tr>
<td>No, of nationalities among respondents (Q3)</td>
<td>8</td>
<td>4</td>
<td>&gt;4</td>
<td>10</td>
</tr>
<tr>
<td>Average age of respondents (Q4)</td>
<td>27.64 [18; 48]</td>
<td>36.75 [26; 47]</td>
<td>20.07 [16; 26]</td>
<td>32.82 [18; 46]</td>
</tr>
</tbody>
</table>

Table 3.20: Characteristics of the communities studied; CrystalSpace, HSQLDB, Stepmania, TikiWiki.
Fourth, all selected communities distinguish unambiguously between at least two types of contributors, namely between core developers and peripheral members. The differentiation between these two contributor types was necessary to separate two hierarchical levels as a dependent variable (the individuals’ position in the knowledge hierarchy). As a final criterion, it was crucial to get in touch with and have the support of one or several key people in the sampled communities, who have been able to invite their colleagues to support the study, and who posted a link to the web-based survey on their community developer mailing lists. All four projects are briefly characterized in table 3.20, which refers to the time of study (November 2003). Additional information on each of the sampled projects is available on the communities’ websites. The web-based survey was online in November and December 2003.

3.1.2. Variable operationalization and data sources

The variables that are included in the knowledge hierarchy model were already more concretized through the above listed hypotheses (chapter 2.3). For quantitative model testing, I operationalized the variables and used a number of data sources for data gathering as outlined below.

Individuals’ position in the knowledge hierarchy. The individuals’ position in the informal knowledge hierarchy (at a certain point of time) is the dependent variable in the model. It is a multidimensional theoretical construct, and it is not directly observable. Any variable operationalizations must consequently make use of an indicator of the individuals’ position in the informal hierarchy. Feasible options in search of a feasible indicator were thereby, firstly, measures of the construct’s various dimensions, which include the individuals’ reputation, others’ attention they can attract, or their degrees of freedom to act, etc. A second option was to directly ask all project members for their perceptions of the hierarchical positions of their colleagues, to aggregate their responses and accordingly rank the resulting positions of every contributor to a certain project.
Both of these options for variable operationalization are, however, based on subjective data that are provided by the project members, such data bear the danger of a strong perception bias, Especially the second option proved to be exceptionally complex and time-consuming for the respondents, since it assumes that they have a complete overview of every individual who contributes to their project, and that the contributors are capable of judging every single individual of their community. For the StepMania project, for instance, every contributor had to judge or estimate the qualities of 107 core developers plus 521 peripheral project members, which is an exceptionally complex task. In order to gain the data in question, both of the above-mentioned options did not seem to be manageable for mid-sized communities as they were studied for this project.

I consequently decided to make use of a third option in order to operationalize the dependent variable. Throughout the entire model-generating part of this study, the Freenet contributors made a clear distinction between core developers and peripheral project contributors. The former were consistently attributed a higher position in the informal project hierarchy than were members of the peripheral member group. This finding was confirmed by independent open source contributors of various projects, and by experts of the software development industry. It can implicitly be found on many open source project websites. I will take up this ranking and ground my analysis on two distinct hierarchical groups: Core developers, who are in a higher hierarchical position and, peripheral members, who are in a subordinate position. The dependent variable is therefore nominally scaled, although the projects’ hierarchical ranking reflects a continuum rather than two discrete categories. I collected data only for one point of time, which refers to the whole time frame under study.

As peripheral project members range all individuals who submitted at least one email to their community’s developer mailing list, but who have no direct access to their projects’ code repository. Core developers, in contrast, do have direct access to their project’s code repository, and their status is additionally listed on the communities’ websites, and the lists are continuously updated. Besides that, core developers, who actively contribute code to their community, appear as contributors in the communities’ code repositories.
**Individuals’ knowledge absorption before project entry.** The first independent variable is a theoretical construct and it is not directly observable. The variable was concretized, *firstly*, by the individuals’ *time investments* in their knowledge absorption activities before their entry to the project, and *secondly*, by the *number of knowledge transfer media* they had used for this purpose. As data source served the web-based survey asking questions directed at the individuals’ total time investments in knowledge absorption activities before they had entered their project (in hours), and how they had spend the time (browsing the CVS, the development email list, or reading general project documents).\(^{14}\)

The gathered survey data mirror the individuals’ subjective and retrospective perceptions, and they could be subject to a memory bias, or an answering bias, which represent answers given with respect to social desirability. Since no alternative indicators were available for measuring the construct, I decided to design the survey’s answer categories rather broadly and was so able to avoid the respondents giving unclear answers to the survey’s questions. In addition to this, the survey included questions that give no direct information on the variables that are a part of the knowledge hierarchy model, but which aim at further validating the respondents’ comments; for instance, “*What was your first official contribution to the project?*”

**Individuals’ involvement to the community post entry.** The second independent variable was concretized, *firstly*, through the project contributors’ accumulated *passive knowledge absorption* behavior that was achieved through browsing the CVS, the mailing list, or other general documents on the project after the individuals’ official entry to their project was submitted. The variable was further operationalized through survey questions asking for the contributors’ average weekly *time investments* (in hours) in these activities, and how they had spent this time (i.e, which *knowledge transfer media* they had used). *Secondly*, the variable was concretized through the individuals’ *active knowledge creation* activities after they had officially entered the project. An operationalization was achieved by means of the *number of emails* the individuals submitted to the list.
Although the variable ‘knowledge creation’ to a certain community encloses various knowledge transfer media (email, code, general documents), I focused exclusively on emails. **Code** contributions, in comparison, served as a discriminating variable to separate core developers from peripheral members (only core developers have direct access to the code repository, while peripheral members don’t), it thus makes no sense to test for such an operationalization. Since not all created **general documents** are objectively accessible through the communities’ websites, I also did not include this third type of created knowledge into the quantitative analysis. Moreover, the survey asked for the contributors’ average **time investments** (per week) for each of the following activities, posting to the mailing list, modifying or adding source code, writing general project documentation.

**Time spent on the project.** The third independent variable refers, **on the one hand**, to the **point in time** during the projects’ life cycle when the individuals officially entered their **community**. I will interpret the day (date) of a contributor’s first email submission as his or her entry to the respective community. For members of the core developer group, I additionally considered the day of their first project contribution, code commit or email submission, whatever occurred first. Every date was converted into a numerical figure, starting with day 1, the first day a contribution was quoted in the data archives, and so it was standardized.

For this part of the study, the developer email archives of two communities were analyzed, **STEPMANIA** & **TikiWiki**. For **STEPMANIA**, a first contribution was quoted in the project’s archives on January 1, 2002, which was interpreted as day 1. For **TikiWiki**, a first contribution was quoted on July 9, 2002, which was similarly interpreted as day 1 of this community’s existence. For **STEPMANIA** this date is unequal to the community’s registration on the **SOURCEFORGE** website (October 16, 2001).

The variable refers, **on the other hand**, to the contributors’ **longevity on the project**. It was operationalized through the number of days between every single contributor’s first and last email submission in the time frame of the study, starting on day 1 (see above) and ending on December 31, 2003. For the core developers, I further calculated the time frame between their first and their last contribution to the community, code or email, whatever occurred first or last.
**Organizational age and size.** The two organization-level independent variables were operationalized, *firstly*, through the number of months since the community’s registration on the SOURCEFORGE website (*organizational age*), and *secondly*, through the accumulated number of community members, who submitted at least one email to the list until December 2003 (*organizational size*).

**Individuals’ personal accumulated knowledge as a share of the project’s total accumulated knowledge.** The variable is a multidimensional, theoretical construct and it is therefore only indirectly observable. The variable is, at the same time, a mediator in the knowledge hierarchy model. As was found empirically, and as was hypothesized, the variable covers, *firstly*, both the contributors’ past knowledge creation and absorption activities. *Secondly*, the variable covers all types of knowledge transfer media that are relevant in the specific community (email, code, general documents) and, *finally*, if refers to all contributors of a certain project and to various points of time. To approach the variable, I chose to compose an indicator which aggregates the above-mentioned factors.  

3.1.3. Data analysis

To test all of the previously listed hypotheses and the full knowledge hierarchy model, data from both sources, the web-based survey (knowledge absorption) and the community data archives, (knowledge creation) are necessary. With respect to the survey, only a small number of responses (Σ50) was filled in and returned by the communities’ members. 42 (84%) of the responding contributors are core developers, and only 8 (16%) respondents proved to be members of the communities’ peripheral groups. As a consequence, the dependent variable (individuals’ position in the knowledge hierarchy), which was operationalized through the individuals’ membership in either the core developer or in the peripheral contributor group, has too little variance in order to make any significant statements with respect to their group memberships.
In essence, though, the participation rate of members of both contributor groups implicitly confirms the model’s correctness for the setting in question. Those individuals (core contributors) who demonstrate great engagement in their communities are also interested in contributing to this study and in learning more about how their community functions, while peripheral members are not. Moreover, a link to the survey was posted on the developer mailing lists of each of the four projects. The analysis of TikiWiki and StepManias mailing lists indicate that many (of the accumulated number of) peripheral members contributed only for a short period of time, and they were no longer contributing at the time of asking. The survey has subsequently not been able to reach the entire, accumulated number of peripheral members that are included in the accumulated number of group members. This has reduced the number of potential respondents.

As a consequence of the lacking variance in the respondents’ group membership, I could not test the hypotheses, which include one or several of the constructs that were exclusively operationalized by means of the survey, since no alternative operationalization was available. As a solution to this situation, I will present descriptive statistics on the survey responses, but use the survey data not for testing the affected hypotheses (H 2a&2b, 3a). The survey responses primarily mirror members of the core developer group and the findings are of great relevance for this study in any case. Additionally, the survey data can give an impression of similarities and differences between the sampled communities. Since to date only a small number of cross-sectional or comparative studies on open source communities exist, these findings are also of great relevance for learning more on how these communities function.

All mediated relationships can neither be described nor tested at all, since the mediator variable is grounded in an aggregation of data on all/ a large number of project contributors and on data, which would only the survey could have gathered (H1, 5a&b, 3a, 6a&b, 7a&b). The following paragraphs will thus exclusively refer to the knowledge hierarchy’s direct relationships. I will additionally briefly describe the findings on hypotheses 8 and 9.
<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Relationship</th>
<th>Testing &amp; description</th>
<th>Community evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>mediator - dependent</td>
<td>not possible</td>
<td>-</td>
</tr>
<tr>
<td>H2&amp;b</td>
<td>direct</td>
<td>survey; Description of findings</td>
<td>CRYSTALSPACE, HSQLDB, Stephania, TikiWiki</td>
</tr>
<tr>
<td>H3a</td>
<td>direct</td>
<td>survey; Description of findings</td>
<td>CRYSTALSPACE, HSQLDB, Stephania, TikiWiki</td>
</tr>
<tr>
<td>H3b</td>
<td>direct</td>
<td>discriminant analysis</td>
<td>Stephania, TikiWiki</td>
</tr>
<tr>
<td>H4a&amp;b</td>
<td>direct</td>
<td>discriminant analysis</td>
<td>Stephania, TikiWiki</td>
</tr>
<tr>
<td>H5a&amp;b</td>
<td>mediated</td>
<td>not possible</td>
<td>-</td>
</tr>
<tr>
<td>H6a&amp;b</td>
<td>mediated</td>
<td>not possible</td>
<td>-</td>
</tr>
<tr>
<td>H7a&amp;b</td>
<td>mediated</td>
<td>not possible</td>
<td>-</td>
</tr>
<tr>
<td>H8</td>
<td>independent - mediator</td>
<td>description of findings</td>
<td>Stephania, TikiWiki</td>
</tr>
<tr>
<td>H9</td>
<td>independent - mediator</td>
<td>description of findings</td>
<td>Stephania, TikiWiki</td>
</tr>
</tbody>
</table>

Table 3.21: Overview of hypotheses – testing and description.

I tested all hypotheses, for which the required data were available (H3b, 4a&b, see also table 3.21). The results of these tests can still indicate whether or not the knowledge hierarchy model makes sense for open source communities other than the Freenet project. The analysis could not, however, be tested for the full knowledge model’s validity and the variance it explains. I restricted the analysis to two communities, Stephania and TikiWiki, since mid-sized open source projects are complex and their analysis requires extensive databases. Stephania and TikiWiki proved to be very different communities across all variables of interest, and the comparison of these two projects shed more light on the reasons for the difference. All survey and archival data were stored in a separate database to facilitate data analysis, and all calculations were based on SPSS 11.0 and Excel.

To test the hypotheses (H3b, 4a&b), all data were analyzed by means of a discriminant analysis, which tests the variables’ independent and aggregated, covariate influence on the individuals’ position in the knowledge hierarchy (dependent variable). The discriminant analysis tests for statistical significance in differences between samples that trace back to a number of independent factors, and the dependent variable is nominally scaled, as it is the case for this study. The analysis aims at testing whether samples are significantly different, and if so, to which independent variables the sample
differences can be traced back. The discriminant analysis therefore has the potential to predict to which group a new, unclassified element should be classified if no other information besides the independent variables is available\textsuperscript{17} (Brosius, 1998; Backhaus et al., 1994). I compared two independent samples, the core developer group and the peripheral contributor group of both the TIKIWIKI and the STEPMANIA projects.

### 3.2. Findings and discussion of quantitative findings

**Hypothesis 2a.** The hypothesis suggests a positive and direct relationship between the time a prospective contributor to an open source software development project spends absorbing organization-specific knowledge before his or her project entry, and the (immediate) position he has in the knowledge hierarchy after her entry to the community. The hypothesis could not be tested, but the web-based survey responses do allow for a brief characterization of the (higher ranked core developer) group members\textsuperscript{18}.

The responding individuals reported on their mediocre to large time investments on learning about their respective community of interest before they officially entered it. On average, they had invested between 11 and 50, and some of them up to 100 hours browsing the developer mailing list, the source code and other general sources of information (e.g., FAQ, papers, foils, articles). The respondents additionally reported that they had spent on average 50-100 hours each on private communication with current members of the respective communities and on other (not named) sources of information.

Although these findings indicate the core developers’ considerable time investments before they entered their community of interest, the respondents still reported difficulties in understanding what was going on in the projects. The survey responses suggest that the project contributors had most difficulties in understanding the projects’ specific technical project standards, followed by their lack in understanding the prevailing community rules, the project’s hierarchy, roles and specializations (see table 3.22).

For (core) contributors, excessive time investment in various knowledge transfer media seems consequently to play a crucial role for understanding the projects’ very characteristics, which is in line with the hypothesized relationship. For a complete
understanding of their community of interest, however, watching and absorbing knowledge seems not to be enough. The prospective contributors are most interested in directly communicating with current project members, who can give them answers to concrete questions, the prospective contributors could not find or which they did not want to search for in the community data archives.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>CrystalSpace</th>
<th>HSQLDB</th>
<th>Stepmania</th>
<th>TikiWiki</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Knowledge absorption before entry; Time spent on (Q13)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>… developer mailing list</td>
<td>&gt;11-50 hrs (2.6)</td>
<td>&lt;51-100 hrs (2.8)</td>
<td>&lt;11-50 hrs (1.8)</td>
<td>&gt;11-50 hrs (2.2)</td>
</tr>
<tr>
<td>… source code CVS</td>
<td>&gt;11-50 hrs (2.6)</td>
<td>&gt;11-50 hrs (2.5)</td>
<td>&gt;11-50 hrs (2.3)</td>
<td>&lt;51-100 hrs (2.7)</td>
</tr>
<tr>
<td>… general project information</td>
<td>&lt;11-50 hrs (1.9)</td>
<td>&gt;51-100 hrs (3.3)</td>
<td>11-50 hrs (2.0)</td>
<td>&lt;11-50 hrs (1.9)</td>
</tr>
<tr>
<td>… private communication with other project contributors</td>
<td>&lt;100-150 hrs (3.9)</td>
<td>&gt;100-150 hrs (4.3)</td>
<td>&gt;51-100 hrs (3.3)</td>
<td>&gt;11-50 hrs (2.4)</td>
</tr>
<tr>
<td>… other sources of information</td>
<td>&gt;51-100 hrs (3.3)</td>
<td>&lt;51-100 hrs (2.7)</td>
<td>&lt;51-100 hrs (2.8)</td>
<td>&lt;51-100 hrs (2.7)</td>
</tr>
<tr>
<td><strong>Knowledge absorption before entry; Source (Q13)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>… developer mailing list</td>
<td>47.06% (8)</td>
<td>75% (3)</td>
<td>28.57% (4)</td>
<td>58.82% (10)</td>
</tr>
<tr>
<td>… source code CVS</td>
<td>71.43% (10)</td>
<td>50% (2)</td>
<td>50% (7)</td>
<td>64.7% (11)</td>
</tr>
<tr>
<td>… general project information</td>
<td>71.43% (10)</td>
<td>100% (4)</td>
<td>47.06% (7)</td>
<td>47.06% (8)</td>
</tr>
<tr>
<td>… private communication with other project contributors</td>
<td>47.06% (8)</td>
<td>50% (2)</td>
<td>71.43% (10)</td>
<td>52.94% (9)</td>
</tr>
<tr>
<td>… other sources of information</td>
<td>78.57% (11)</td>
<td>25% (1)</td>
<td>50% (7)</td>
<td>47.06% (8)</td>
</tr>
<tr>
<td><strong>These activities helped to understand (Q14)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1: excellent … 5: Not at all)</td>
<td>On average: 2.0</td>
<td>On average: 2.5</td>
<td>On average: 2.8</td>
<td>On average: 2.6</td>
</tr>
<tr>
<td>… project vision and overall direction</td>
<td>2.8</td>
<td>3.5</td>
<td>3.1</td>
<td>2.5</td>
</tr>
<tr>
<td>… community rules</td>
<td>2.4</td>
<td>3.0</td>
<td>3.3</td>
<td>2.7</td>
</tr>
<tr>
<td>… technical standards</td>
<td>2.3</td>
<td>2.8</td>
<td>2.7</td>
<td>2.1</td>
</tr>
<tr>
<td>… project architecture &amp; content</td>
<td>2.5</td>
<td>3.5</td>
<td>2.7</td>
<td>2.1</td>
</tr>
<tr>
<td>… hierarchy, roles &amp; specialization</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.22: Knowledge absorption before project entry: Survey responses.
**Hypothesis 2b.** The hypothesis suggests a *positive, direct relationship between the number of (different) types of knowledge transfer media used for absorbing organization-specific knowledge before a prospective contributor’s entry to the community and an individual’s (immediate) position in the knowledge hierarchy* after his entry to the project. The hypothesis could not be tested; yet, the survey responses suggest the following characterization of the core developer group:

Contributors to all four communities reported on their simultaneous use of various knowledge transfer media, the mailing list, code repository, general project documents, private communication media (IRC, private email) or other, unspecified alternatives. In all four communities, about 50% or more of the respondents indicated that they used every single one of these knowledge transfer media types, with the Stepmania mailing list as the only exception (28.57%) (see table 3.22). This finding is similarly reflected in the entirely positive correlations between the contributors’ use of the various knowledge transfer media. The strongest correlations could be found between the categories ‘mailing list’ and ‘other,’ and between ‘general documents’ and ‘private discussion,’ the least correlations could be found between ‘CVS’ and ‘other,’ and ‘CVS’ and ‘general documents’. These findings are in line with the hypothesized relationship.

These findings indicate, on the one hand, that some contributors make use of several or all knowledge transfer media types, while others make no use of them at all. On the other hand, the results indicate again, that the communities’ official and permanently stored data archives (mailing list and general documents) are not enough in order to understand a project in its entirety, since these knowledge transfer media are used in combination with other, more personal means of communication. For the code archives, this seems not to be the case, since the CVS is found to be a knowledge source that is sufficient for understanding a project before a prospective contributor’s entry to the community.

**Hypothesis 3a.** The hypothesis suggests a *positive and direct relationship between the amount of organization-specific knowledge (time and number of different transfer media) a contributor continuously absorbs after his project entry and her position in the knowledge hierarchy.* The hypothesis could not be tested, and I will describe the (core) contributors’
behavior as indicated by the survey responses. The survey respondents of all four communities reported that they on average spend 1-3 hours per week on absorbing knowledge from each the mailing list and their communities’ code repositories. In all communities (except Stepmania), the respondents reported additionally that they spend slightly more time (2-3 hours) on browsing general project documents (FAQ, papers, foils, etc.). For the Stepmania contributors, general project information is of even greater importance than for the members of the other projects. They spend on average between 4-10 hours a week reading these knowledge sources.

All correlations between the numbers of hours that the contributors invested in browsing the various media are positive\(^2\), and the strongest correlations can be found between ‘mailing list’ and ‘CVS,’ and ‘mailing list’ and ‘general documents’. The findings indicate that while the contributors are actively contributing to a community, a single source of knowledge is not enough to entirely capture what is going on in the projects’ daily progress. The knowledge transfer media seem to transfer rather different, complementary types of knowledge. In sum, these findings are in line with the hypothesized relationships.

Notwithstanding, the number of contributors having reported to use a certain knowledge transfer medium while being members of the project differs broadly between the communities. For the Stepmania (85.71\%) and TikiWiki (58.82\%) contributors, the code repository is the most central knowledge source, while the CristalSpace contributors prefer to make use of general project documents as their most valuable knowledge source (42.86\%). Interestingly, the contributors’ continuous understanding of their project differs between the communities. Thy CristalSpace contributors, on average, have an almost full understanding of their project, while the HSQMDB and Stepmania contributors’ understanding is rather mediocre. One reason therefore could be that the HSQMDB and Stepmania members make less use of older project documentation than the contributors of the other two projects, which contains fundamental ideas that were created in the communities at an earlier point of time (table 3.23).
Table 3.23: Knowledge absorption post project entry: Survey responses.

Other survey findings. Additional survey responses give some information on the communities’ project-related background knowledge. The findings indicate that the communities attract different types of interested contributors, for evidence see table 3.24. StepMania and TikiWiki, both game programs, attract rather young contributors (average age 20.07 and 27.64 years); TikiWiki (content management system, 32.82 years) and HSQLDB (relational database engine, 36.75 years) are more attractive for older contributors, who tend additionally to use the evolving programs themselves. This finding is additionally supported by the contributors’ occupational or student status.
More than 60% of Stepmania’s contributors are high school, college or university students, while TikiWiki’s contributors do primarily work (about 90%).

The contributors to all four communities have reported a large share of related knowledge of software engineering (for all communities more than 73%), which is a prerequisite to understanding a project’s aim and content. They reported their knowledge of various programming languages, and contributors to the Stepmania project do on average know a smaller number of different languages (66.7% of the respondents: 1-5 languages) than do the contributors to the other communities (between 50% and 71.43%; 6 and more languages). The Stepmania contributors differ also with respect to their experiences with other open source communities from the members of the other three projects. Less than half of the responding Stepmania contributors (46.7%) reported that they had been involved in any other open source community, while for the remaining three communities the share is significantly higher (64.71-100%).

Surprisingly, according to the contributors’ responses, the communities’ aim is not necessarily responsible for attracting a certain type of project member, Stepmania and CrystalSpace are both game programs, but only Stepmania differs broadly from the other three communities. Stepmania seems to be a community for open source ‘newbies’ with low entry barriers, while Cristalspace or TikiWiki are more advanced communities.
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>CrystalSpace</th>
<th>HSQLDB</th>
<th>Stepmania</th>
<th>TikiWiki</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average age (Q4)</strong></td>
<td>27.64 [18;48]</td>
<td>36.75 [26;47]</td>
<td>20.07 [16;26]</td>
<td>32.82 [18;46]</td>
</tr>
<tr>
<td><strong>Occupation/ student status (Q5)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...student</td>
<td>42.86% (6)</td>
<td>0% (0)</td>
<td>46.51% (6)</td>
<td>5.88% (1)</td>
</tr>
<tr>
<td>...work</td>
<td>57.14% (8)</td>
<td>75% (9)</td>
<td>38.46% (5)</td>
<td>88.24% (15)</td>
</tr>
<tr>
<td>...high school</td>
<td>0% (0)</td>
<td>0% (0)</td>
<td>15.38% (2)</td>
<td>0% (0)</td>
</tr>
<tr>
<td><strong>Additional background knowledge (Q6)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...software engineering</td>
<td>80% (12)</td>
<td>100% (4)</td>
<td>73.3% (11)</td>
<td>88.24% (15)</td>
</tr>
<tr>
<td>...other natural sciences</td>
<td>40% (6)</td>
<td>50% (2)</td>
<td>60.0% (9)</td>
<td>17.65% (3)</td>
</tr>
<tr>
<td>...none of these</td>
<td>13.3% (2)</td>
<td>0% (0)</td>
<td>13.3% (2)</td>
<td>5.88% (1)</td>
</tr>
<tr>
<td><strong>No. of programming languages (Q7)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...none</td>
<td>0% (0)</td>
<td>0% (0)</td>
<td>6.7% (1)</td>
<td>11.76% (2)</td>
</tr>
<tr>
<td>...1 to 5</td>
<td>28.57% (4)</td>
<td>50% (2)</td>
<td>66.7% (10)</td>
<td>35.29% (6)</td>
</tr>
<tr>
<td>...6 and more</td>
<td>71.43% (10)</td>
<td>50% (2)</td>
<td>26.6% (4)</td>
<td>52.94% (9)</td>
</tr>
<tr>
<td><strong>Involvement in other OSS projects (Q8)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...yes</td>
<td>78.57% (11)</td>
<td>100% (4)</td>
<td>46.7% (7)</td>
<td>64.71% (9)</td>
</tr>
<tr>
<td>...no</td>
<td>21.43% (3)</td>
<td>0% (0)</td>
<td>53.3% (8)</td>
<td>35.29% (6)</td>
</tr>
<tr>
<td><strong>Roles in other OSS projects (Q9)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...maintainer</td>
<td>42.86% (6)</td>
<td>25% (1)</td>
<td>20% (3)</td>
<td>11.76% (2)</td>
</tr>
<tr>
<td>...core developer</td>
<td>50.00% (7)</td>
<td>25% (1)</td>
<td>20% (3)</td>
<td>17.65% (3)</td>
</tr>
<tr>
<td>...infrequent contributor</td>
<td>42.86% (6)</td>
<td>100% (4)</td>
<td>26.7% (4)</td>
<td>52.94% (9)</td>
</tr>
<tr>
<td>...lurker</td>
<td>28.57% (4)</td>
<td>50% (2)</td>
<td>13.3% (2)</td>
<td>29.41% (5)</td>
</tr>
<tr>
<td>...user</td>
<td>28.57% (4)</td>
<td>50% (2)</td>
<td>26.7% (4)</td>
<td>47.06% (8)</td>
</tr>
<tr>
<td><strong>Involvement in other OSS projects since (Q10)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...before joining this project</td>
<td>50% (7)</td>
<td>75% (3)</td>
<td>33.3% (5)</td>
<td>64.7% (11)</td>
</tr>
<tr>
<td>...since around the same time when joining this project</td>
<td>21.43% (3)</td>
<td>25% (1)</td>
<td>13.3% (5)</td>
<td>0% (0)</td>
</tr>
<tr>
<td>...only recently, after joining this project</td>
<td>14.29% (2)</td>
<td>0% (0)</td>
<td>6.7% (1)</td>
<td>0% (0)</td>
</tr>
<tr>
<td><strong>Involvement in overall OSS community (Q11)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...frequently or from time to time</td>
<td>50% (7)</td>
<td>50% (2)</td>
<td>40% (6)</td>
<td>100% (17)</td>
</tr>
<tr>
<td>...rarely or never</td>
<td>50% (7)</td>
<td>50% (2)</td>
<td>60% (9)</td>
<td>0% (0)</td>
</tr>
<tr>
<td><strong>Involvement in overall OSS community since (Q12)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...before joining this project</td>
<td>57.14% (8)</td>
<td>50% (2)</td>
<td>40% (6)</td>
<td>88.24% (15)</td>
</tr>
<tr>
<td>...since around the same time when joining this project</td>
<td>28.57% (4)</td>
<td>0% (0)</td>
<td>67% (1)</td>
<td>0% (0)</td>
</tr>
<tr>
<td>...after having joined this project</td>
<td>0% (0)</td>
<td>0% (90)</td>
<td>20% (3)</td>
<td>11.76% (2)</td>
</tr>
</tbody>
</table>

Table 3.24: Other survey responses, descriptive statistics.
**Hypotheses 3b, 4a&b.** These hypotheses were tested by means of a discriminant analysis, each for the TIKIWiki and the STEPMANIA project. All independent variables were tested for their separate relationship with the dependent variable, and they were tested collectively to test for the independent variables’ covariance. For the latter, the suggested relationships between all three independent (x1: amount of knowledge created post entry, x2: point of time of project entry, x3: project longevity) and the dependent variable (y: position in the hierarchy) will therefore be interpreted as a complete model, although they represent only a fraction of the original knowledge hierarchy model (chapter 2.3).

I tested for two alternative models in STEPMANIA and TIKIWiki, and the models differ with respect to the operationalization of the variables x2 and x3; **Model 1 (M1)** interprets the individuals’ first email submission as the point of time the contributors entered their community (x2), and their project longevity (x3) as the number of days between their first and last email submission. **Model 2 (M2)** does not focus on the contributors’ email submissions alone in order to operationalize x2 and x3, but it focuses on code or email contributions, whichever occurred first or last. Both models compute from group sizes and replace missing values with the mean. I will start with some descriptive statistics (tables 3.25 to 3.28).

The **descriptive statistics** illustrate differences in the communities’ heights of entry and vertical mobility barriers. Both communities’ vertical mobility barriers seem to be lower than the FREENET projects’, which were outlined in the first order analysis part of this study. At the same time, vertical mobility barriers are higher in STEPMANIA than in TIKIWiki. The finding becomes, on the one hand, clear through the communities’ high percentage of core developers (TIKIWiki: 51.63%, STEPMANIA: 17.04%). On the other hand, STEPMANIA has a high percentage of contributors who posted just a single email to the list (63.9%), but who could not or who wanted not to continue contributing to the ongoing project. STEPMANIA’s entry barriers therefore seem lower than those of TIKIWiki, again stressing the project’s character as a community for open source beginners.
These observations are additionally stressed through both projects’ accumulated knowledge bases. The accumulated number of emails in Stepmania (1,592) is only about one fourth of TikiWiki’s (6,671), which may enable for a faster entry to the project as a peripheral member. Their accumulated numbers of CVS commits, however, are vice versa and Stepmania’s CVS archive (39,546 commits) is almost twice as large as TikiWiki’s (23,311), which requires that an individual first absorbs more accumulated (technical) knowledge before he can actively contribute to the code repository (as a core developer).

**Hypothesis 3b** suggests a positive relationship between the amount of organization-specific knowledge a contributor continually creates after his or her entry to a community ($x_1$) and the individual’s position in the knowledge hierarchy ($y$). The separate testing of hypothesis 3b’s independent variable leads to the following findings. For TikiWiki, the analysis finds within-group means of 56.59 emails per core developer and 9.01 emails per peripheral member in the time frame of reference. In both models the variable has a highly significant, positive relationship of mediocre strength to the dependent variable, as was hypothesized (M1: $F_{25,631} = 25.631$, sig. 0.00; M2: $F_{22.746} = 22.746$, sig. 0.00). For Stepmania, the findings are comparable to TikiWiki’s, but stronger and they are highly significant for both models (M1: $F_{79,142} = 79.142$, sig. 0.00; M2: $F_{63.335} = 63.335$, sig. 0.00). **Hypothesis 3b cannot be rejected**, and the more emails an individual posts after his or her entry to a project, the more likely he or she is a member of the core developer group.

**Hypothesis 4a** suggests a negative relationship between the point of time an individual decides to officially enter a community ($x_2$), and his or her position in the knowledge hierarchy ($y$). The analysis’ results on both models vary for the two communities. In the Stepmania project, the core contributors posted their first email to the list on average on day 295 (M1: October 22, 2002) and their first contribution (code or email) was on average submitted on day 301 (M2: October 28, 2002), 6 days later. The peripheral members submitted their first email on average 9 months later to the list, on day 572 (M1 & M2: July 26, 2003). In both models, the variable is highly significant, representing a strong positive relationship (as hypothesized) to the individuals’ group memberships (M1: $F_{353,284} = 353.284$, sig. 0.00; M2:
F=360.435, sig. 0.00). For the StepMania project, hypothesis 4b cannot be rejected, and the earlier a contributor enters the project, the more likely he or she is to be a core developer.

For the TikiWiki project, the findings are different. For the TikiWiki project, the core developers contributed their first email on average on day 344 (M1: June 18, 2003), almost a year after the project’s registration on SourceForge. Their first contribution (email or code, M2) was, on average, 3 days earlier, which illustrates that some core contributors first posted code, before they had started discussing on the list. Peripheral members posted their first email on average on day 310 (M1&M2: May 15, 2003), about one month before the core contributors did so. Both models indicate a very weak and positive relationship (unlike hypothesized), which is of questionable significance (M1: F=4.740, sig. 0.030; M2: F=4.062, sig. 0.045). Hypothesis 4b must therefore be rejected for the TikiWiki project. An earlier entry to an open source community is consequently not advantageous for a higher position in TikiWiki’s knowledge hierarchy. Possible reasons for this finding could, for instance, be a difficult project start or heavy turnover of key people in the project’s early days.

The different results for the two communities in question again illustrate open source projects’ differences, which should be analyzed more closely in future work. It is likely that the relationship between the variables x2 and y is mediated by an additional variable, which accounts for the difference across projects. It may be, that the variable x2 is significant in the original knowledge hierarchy model that assumes such a mediator, namely the individual’s knowledge as a share of the entire organization’s accumulated knowledge.

Hypothesis 4b suggests a positive relationship between an individual’s longevity in an open source community (x3) and his or her position in the knowledge hierarchy (y). In the TikiWiki project, a core developer posted emails over an average timeframe of 111 days (M1) to the mailing list, and contributed emails and code over a timeframe of 120 days (M2). The peripheral members were, on average, active for only 39 days (M1&M2), which is about one third of the core developers’ project longevity. In both models, the variable is effective with mediocre strength and it is of high significance into the hypothesized direction (M1: F=27.336, sig. 0.00; M2: F=41.390, sig. 0.00).
For **Stepmania**, an average core contributor submitted both emails and code (M1&M2) over a timeframe of **210 days**, A peripheral member did so over an average of only **5 days**, which is about **2.3%** of the core contributors’ longevity in the project. In both models, the variable is thus highly significantly related to y in the hypothesized direction (M1; F=396.431, sig. 0.00; M2: F=428.442, sig. 0.00). For both projects, hypothesis 4b cannot be rejected and those individuals who are active on a project for a longer time frame are more likely to be a member of the core group than those who remain in the project shorter. For both communities, the variable explains most of the variance of the contributors’ position in knowledge hierarchy.

**Multivariate model**, Taken together, Model 1 and 2 have different explanatory values both for the two communities in study and between the two contributor groups. This becomes clear by way of their different discriminant functions:

**TikiWiki M1:** \[ y = -2.768 + 0.007x_1 + 0.007x_2 + 0.007x_3 \]

**TikiWiki M2:** \[ y = -3.207 + 0.002x_1 + 0.007x_2 + 0.011x_3 \]

**Stepmania M1:** \[ y = 2.726 + 0.001x_1 - 0.005x_2 + 0.009x_3 \]

**Stepmania M2:** \[ y = 2.335 + 0.02x_1 - 0.005x_2 + 0.008x_3 \]

For the TikiWiki project both limited models (M1&M2) could classify **58.2%** of the original cases correctly (explained variance). For the Stepmania project, the model could classify up to **87.7%** (M2) correctly. In both communities, the members of the peripheral member group could be classified extremely well with more than **90%** of correct classifications (TikiWiki: **91.2%**, Stepmania: **98.8%**, both M1). For the core developers’ behavior, however, the model is correct, but it has less predicting value (TikiWiki: **38.6%**, Stepmania: **34.6%** of correct classifications, both M2), which is also reflected in the models’ goodness indicators (tables 3.26 & 3.28).

These findings are entirely consistent with the knowledge hierarchy model as suggested by this study. For predicting the peripheral members’ behavior, the limited model is sufficient, since members of this group were found to be rather inactive across all variables (and therefore also less knowing) compared to the core developers, For the
core developers, the limited model is capable of partly explaining their hierarchical positions and their complex behavior, but it does not have the capability of predicting it entirely. There must consequently exist some further variables that could not be tested, but that are relevant in explaining who is a member of the core developer group, and who is not. Since the members of this group were found to be rather active than passive, the limited model tested here is not enough. It could be that the additional variables as suggested in the full knowledge hierarchy model of this study, which could not be tested here, account for the unexplained variance for the core developer group. A number of findings of the analysis additionally support these reflections,

firstly, the proposed models M1 and M2 are suitable to classify the StepMania contributors’ behavior much better than the TikiWiki members’. The reason for this is that StepMania has a greater percentage of peripheral members than TikiWiki, which is clearly explained by the models. Secondly, for both communities, model 2 is better suited to predict the core developers’ behavior than is model 1. Model 2 could integrate slightly more complexity in the variable operationalization, since it accounts for the core developer’s email or code contributions, and not only for their email postings in order to operationalize $x_2$ and $x_3$.

Finally, the core developer group has larger standard deviation values than does the peripheral member group in each of the models (TikiWiki: Mean 235.44, sd. 890.586, StepMania: Mean 651.87, sd. 2962.206), which reflects the core developers’ more complex and more diversified behavior. Supposedly, their code contribution behavior is most relevant to further differentiate between various hierarchical levels within the core developer group. In sum, the findings gained through the discriminant analysis support the suggested relationships of the knowledge hierarchy model generated by this work.
<table>
<thead>
<tr>
<th>Analysis case processing summary</th>
<th>TikiWiki Model 1</th>
<th>TikiWiki Model 2</th>
<th>Stepmania Model 1</th>
<th>Stepmania Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid cases N (Percent)</td>
<td>240 (78.4%)</td>
<td>248 (81.0%)</td>
<td>571 (90.9%)</td>
<td>581 (92.5%)</td>
</tr>
<tr>
<td>At least one missing discriminating variable</td>
<td>66</td>
<td>60</td>
<td>57</td>
<td>47</td>
</tr>
<tr>
<td>Total</td>
<td>306</td>
<td>306</td>
<td>628</td>
<td>628</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Standardized canonical discriminant function coefficients</th>
<th>TikiWiki Model 1</th>
<th>TikiWiki Model 2</th>
<th>Stepmania Model 1</th>
<th>Stepmania Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of emails (x1)</td>
<td>.513</td>
<td>.171</td>
<td>.047</td>
<td>.730</td>
</tr>
<tr>
<td>First email/contribution. (x2)</td>
<td>.774</td>
<td>.866</td>
<td>-.532</td>
<td>-.502</td>
</tr>
<tr>
<td>Longevity email/contribution. (x3)</td>
<td>.725</td>
<td>1.044</td>
<td>.603</td>
<td>.598</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Structure matrix (pooled within-group correlations)</th>
<th>TikiWiki Model 1</th>
<th>TikiWiki Model 2</th>
<th>Stepmania Model 1</th>
<th>Stepmania Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of emails (x1)</td>
<td>.270</td>
<td>.213</td>
<td>.397</td>
<td>.334</td>
</tr>
<tr>
<td>First email/contribution. (x2)</td>
<td>.627</td>
<td>.504</td>
<td>-.838</td>
<td>-.798</td>
</tr>
<tr>
<td>Longevity email/contribution. (x3)</td>
<td>.648</td>
<td>.679</td>
<td>.888</td>
<td>.870</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Canonical discriminant function coefficients (unstandardized)</th>
<th>TikiWiki Model 1</th>
<th>TikiWiki Model 2</th>
<th>Stepmania Model 1</th>
<th>Stepmania Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of emails (x1)</td>
<td>.007</td>
<td>.002</td>
<td>.001</td>
<td>.020</td>
</tr>
<tr>
<td>First email/contribution. (x2)</td>
<td>.007</td>
<td>.007</td>
<td>-.005</td>
<td>-.005</td>
</tr>
<tr>
<td>Longevity email/contribution. (x3)</td>
<td>.007</td>
<td>.011</td>
<td>.009</td>
<td>.008</td>
</tr>
<tr>
<td>Constant</td>
<td>-2.768</td>
<td>-3.207</td>
<td>2.726</td>
<td>2.336</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function at group centroids</th>
<th>Core developer</th>
<th>Peripheral member</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.661</td>
<td>.737</td>
</tr>
<tr>
<td></td>
<td>3.030</td>
<td>2.910</td>
</tr>
<tr>
<td></td>
<td>-2.911</td>
<td>-3.386</td>
</tr>
</tbody>
</table>

Table 3.25: Discriminant function summary.
<table>
<thead>
<tr>
<th></th>
<th>TikiWiki Model 1</th>
<th>TikiWiki Model 2</th>
<th>StepMania Model 1</th>
<th>StepMania Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wilks’ Lambda</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Wilks’ Lambda, F, df1, df2, sig.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Number of emails posted (x1)</strong></td>
<td>903</td>
<td>.915</td>
<td>.878</td>
<td>.901</td>
</tr>
<tr>
<td></td>
<td>25,631</td>
<td>22,746</td>
<td>79,142</td>
<td>63,335</td>
</tr>
<tr>
<td></td>
<td>1.238</td>
<td>1.245</td>
<td>1.569</td>
<td>1.579</td>
</tr>
<tr>
<td></td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
</tr>
<tr>
<td><strong>First email or contribution posted (x2)</strong></td>
<td>980</td>
<td>.984</td>
<td>.617</td>
<td>.616</td>
</tr>
<tr>
<td></td>
<td>4,740</td>
<td>4,062</td>
<td>353,284</td>
<td>360,435</td>
</tr>
<tr>
<td></td>
<td>1,238</td>
<td>1,245</td>
<td>1.569</td>
<td>1.579</td>
</tr>
<tr>
<td></td>
<td>.030</td>
<td>.045</td>
<td>.000</td>
<td>.000</td>
</tr>
<tr>
<td><strong>Email or contribution longevity (x3)</strong></td>
<td>.897</td>
<td>.855</td>
<td>.589</td>
<td>.575</td>
</tr>
<tr>
<td></td>
<td>27,336</td>
<td>41,390</td>
<td>396,431</td>
<td>428,442</td>
</tr>
<tr>
<td></td>
<td>1,238</td>
<td>1,245</td>
<td>1.569</td>
<td>1.579</td>
</tr>
<tr>
<td></td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
</tr>
<tr>
<td><strong>Eigenvalue</strong></td>
<td>.274</td>
<td>.266</td>
<td>.884</td>
<td>.979</td>
</tr>
<tr>
<td><strong>Canonical correlation</strong></td>
<td>.464</td>
<td>.518</td>
<td>.685</td>
<td>.703</td>
</tr>
<tr>
<td><strong>Wilks’ Lambda (function)</strong></td>
<td>.785</td>
<td>.732</td>
<td>.531</td>
<td>.505</td>
</tr>
<tr>
<td><strong>Chi-square</strong></td>
<td>57,254</td>
<td>75,793</td>
<td>359,460</td>
<td>393,712</td>
</tr>
<tr>
<td>(Chi-square, df, sig.)</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
</tr>
</tbody>
</table>

Table 3.26: Discriminant analysis, goodness tests.
<table>
<thead>
<tr>
<th></th>
<th>TikiWiki Model 1</th>
<th>TikiWiki Model 2</th>
<th>Stemania Model 1</th>
<th>Stemania Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std.</td>
<td>Valid N</td>
<td>Mean</td>
</tr>
<tr>
<td>Core developer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of emails (x1)</td>
<td>56.59</td>
<td>109.743</td>
<td>92</td>
<td>52.59</td>
</tr>
<tr>
<td>First email/contribution (x2)</td>
<td>343.83</td>
<td>123.986</td>
<td>92</td>
<td>340.38</td>
</tr>
<tr>
<td>Longevity email/contribution (x3)</td>
<td>111.40</td>
<td>138.582</td>
<td>92</td>
<td>120.23</td>
</tr>
<tr>
<td>Peripheral member</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of emails (x1)</td>
<td>9.01</td>
<td>25.639</td>
<td>148</td>
<td>9.01</td>
</tr>
<tr>
<td>First email/contribution (x2)</td>
<td>309.54</td>
<td>115.160</td>
<td>148</td>
<td>309.54</td>
</tr>
<tr>
<td>Longevity email/contribution (x3)</td>
<td>38.95</td>
<td>75.819</td>
<td>148</td>
<td>38.95</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of emails (x1)</td>
<td>27.25</td>
<td>74.346</td>
<td>240</td>
<td>26.47</td>
</tr>
<tr>
<td>First email/contribution (x2)</td>
<td>322.68</td>
<td>119.537</td>
<td>240</td>
<td>321.90</td>
</tr>
<tr>
<td>Longevity email/contribution (x3)</td>
<td>66.72</td>
<td>109.973</td>
<td>240</td>
<td>71.53</td>
</tr>
</tbody>
</table>

Table 3.27: Discriminant analysis group statistics.

<table>
<thead>
<tr>
<th></th>
<th>TikiWiki Model 1</th>
<th>TikiWiki Model 2</th>
<th>Stemania Model 1</th>
<th>Stemania Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Original</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Count</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core developer</td>
<td>43</td>
<td>115</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>Peripheral member</td>
<td>13</td>
<td>135</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>Percent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core developer</td>
<td>27.2</td>
<td>72.8</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>Peripheral member</td>
<td>8.8</td>
<td>91.2</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>Percent of cases correctly classified</td>
<td>58.2</td>
<td>58.2</td>
<td>86.9</td>
<td>87.7</td>
</tr>
</tbody>
</table>

Table 3.28: Discriminant analysis classification results.
**Hypotheses 8&9.** Hypotheses 8 and 9 suggest a positive relationship between an open source community’s age and size and the amount of the community’s accumulated knowledge base. Both hypotheses could not be tested, firstly, since no data was available to operationalize their dependent variable. Secondly, the open source communities in question were sampled with respect to their size and age, which are the independent variables of the hypothesized relationships. Testing the hypotheses for the selected communities would end up in a circular argumentation. In order to test the hypotheses, some additional data are required, preferably cross-sectional data on a large number of comparable open source communities.

I will, however, describe the relationship between the independent and dependent variables as observed for Stepmania and TikiWiki. The results are exclusively based on the created knowledge after the contributors’ project entry, and they reflect only a part of the hypotheses’ content, since data on their knowledge absorption behavior before and past their project entries are missing. The figures below illustrate that for both communities, Stepmania and TikiWiki, the accumulated number of emails and code commits steadily increased with the projects’ growing age and accumulated number of contributors. Unlike in the Freenet project, the number of code commits is larger in both communities than the number of emails submitted to the developer mailing lists.
Figure 3.29: *Stepmania*’s accumulated number of CVS commits, emails, new contributors over time.

Figure 3.30: *TikiWiki*’s accumulated number of CVS commits, emails, new contributors over time.
4. Discussion and Concluding Remarks

Research questions and findings. The study aimed at finding answers to the following research questions: Firstly, does the hierarchical principle exist in (seemingly democratic) open source software development communities? Secondly, upon which foundation is the hierarchy in open source software projects grounded, and what influences the hierarchical position of individuals in this setting? To answer these guiding questions, I employed a two-step analytical and methodological process. In the first step, I employed the grounded theorizing method and conducted an inductive single-case study on the FREENET project. The outputs of this first step were hypotheses and a model on a knowledge hierarchy for the setting in question. In a second step, I chose to test the previously gained findings by means of a quantitative data analysis. As data sources served a web-based survey filled in by members of four open source communities, CRYSTALSPACE, HSQLDB, STEPMANIA, TIKIWIKI, and publicly available, archival data on STEPMANIA, TIKIWIKI.

My findings suggest the prevalence of a hierarchy in the open source setting, since the project members differ broadly with respect to their informal ranks, their decision rights, their degrees of freedom to act, the attention they attract from their colleagues and their reputation. The hierarchy prevailing in the communities studied, differs from what we know from traditional organizations on the market-firm continuum. In traditional organizations, a hierarchy is explicitly given, and it is grounded in formal authority and on centralized property rights. In the open source setting, the hierarchy is informal, emergent, and it is grounded in the contributors’ project-specific knowledge. I refer to this type of hierarchy as a ‘knowledge hierarchy,’ which is, at the same time, responsible for entry, exit, vertical and horizontal mobility barriers in the project under study.

An individual’s position in the knowledge hierarchy is influenced by a number of factors. It is influenced by how much organization-specific knowledge an individual could absorb before his or her entry to the community, by his involvement to the specific community after the entry to the project, and by the time span the individual had spent on the project. In addition, the knowledge hierarchy model suggests that an individuals’ share of accumulated individual project-specific knowledge of the entire knowledge that was accumulated in the project serves as a mediator variable.
**Limitations and future research.** The study suffers from a number of methodological limitations and it opens up avenues for future research. *Firstly*, the study’s theory-generating part is based on a single case study which made it possible to cope with the topic’s novelty and complexity. I was able to analyze the FREENET project’s informal hierarchy in an in-depth way. One outcome of this work is that open source projects are very different, a finding also confirmed by other authors (e.g., Ghosh, 2002; Krishnamurthy, 2002). All findings of this study, although they were (partly) tested in other open source settings, can only be generalized to a limited extend. In order to learn more about open source communities’ characteristics, a cross-sectional comparison of a large number of projects would be very helpful.

*Secondly*, due to a lack in variance between the web-based survey respondents, some hypotheses and the full knowledge hierarchy model could not be tested in the study’s second part. Although all findings of the quantitative model-testing part of this work were in line with the hypothesized model, its validity is not yet clarified and still open for future research. Of greatest interest for future research is how much variance the mediator variable in the model is able to explain, since the quantitative findings refer exclusively to a direct relationship between the independent and the dependent variables.

*Thirdly*, the dependent variable (an individual’s position in the knowledge hierarchy) in this study was operationalized in a rather broad way. I distinguished only between two hierarchical levels, namely between higher-level core developers and lower-level peripheral members. The findings indicate that the structure of the core developer group is complex and calls for a stronger differentiation. It can be assumed that the informal hierarchy in open source communities refers not only to two groups, but rather to a continuum on which we can find contributors who occupy a broad array of positions in the informal hierarchy.

*On the one hand*, future research should therefore search for options about how the individuals’ hierarchical positions can alternatively be operationalized. *On the other hand*, the knowledge hierarchy as discovered in this work makes it difficult to speak of the ‘one typical’ open source contributor. A more differentiated perspective would be helpful in order to understand their different motivations, rewards, project preferences. Few
researchers who do research on open source projects are yet in a position to take in a more differentiated view (e.g. von Krogh et al., 2003a; Benkler, 2003).

Fourth, although the theory-generating part of this study revealed that *time and the communities’ evolution play crucial roles* in the informal knowledge hierarchies of open source communities, the theory-testing part of this project referred to only one point in time (the entire period under study). Since communities in general seem to be capable of simultaneously offering a stable yet adaptable environment, these communities could be an interesting model for organizing innovative output. It would be most interesting to analytically separate the dynamic from the stable elements in open source communities and to compare the findings with other forms of organizing.

**Discussion and contributions.** Only recently has the deep analysis of virtual organizations been able to attract considerable attention, and prior studies are primarily located in the field of research on open source online communities (see paper 2) or on new forms of (networked) organization and communication (e.g. Hansen & Haas, 2001; Ahuja & Carley, 1999; Finholt & Sproull, 1990; DeSanctis & Jackson, 1994; Lipnack & Stamps, 1997; Sproull & Kiesler, 1986). Authors have characterized virtually working organizations as extraordinarily democratic and decentralized, having stressed the informal character of these types of organizations. In prior work on virtual organizations, however, a hierarchy has been widely denied or ignored (e.g. Camillus, 1993; Mills, 1991). This study contributes in two respects to these ongoing research streams, since it discovers, firstly, that a hierarchy *is* in existence in the virtual setting under study, and secondly, the study suggests a mechanism in which the hierarchy in place is grounded.

These findings open new topics and questions of various kinds. Firstly, it is striking that the open source contributors, either consciously or unconsciously, decided to adopt for a *hierarchical project composition* grounded upon their project-specific knowledge. It is not that the principle of a hierarchy in general is entirely new; we are already familiar with it from traditional organizations where it is effective in combination with other formal and informal structuring mechanisms, such as authority or formal coordination instruments. The question is, ‘Why do individuals who are a part of, and
who create a new system from scratch for which they could freely choose any imaginable structuring mechanism, decide to rely on a hierarchical form?

*One explanation* is that contributors to open source projects have a life outside their software development work, in which they are frequently confronted with and are a part of many different hierarchies. As a consequence, they learn and transfer the principle of any kind of a hierarchy to the open source setting without being aware of their actions and without further questioning the mechanism. Open source software development projects are thus not located in any open space, but are (path-dependently) bound to their outside societal setting. A *second explanation* is that the principle of a hierarchy is, per se, a helpful structuring mechanism capable of successfully reducing complexity or an overload of information. Prior work has stressed that in systems of a given size and complexity, hierarchies require less information exchange than do teams (Simon, 1977), and hierarchies tend to be more robust in situations of changing, fluid membership (Carley, 1992), which are both the case for open source communities. In addition, this explanation is also stressed by the fact that not only humans, but also most animals are organized in hierarchies, although they are not able to consciously decide on their group structures.

*Secondly,* the ongoing discussion on democracy in organization has posed the question of whether a hierarchy, per se, is harmful to organizations or not. According to Harrison and Freeman, democracy entails “*that members of an organization or society participate in processes of organizing and governance*” (Harrison & Freeman, 2004: 49). Democratic organizations have thereby been found to have advantages, such as the members’ increased commitment to the organization and to their own decisions, their greater responsibility for their work and their work outcomes, Democratic organizations have also been found to have disadvantages, since democratic decisions take a lot of time and a democracy bears the danger that lower-level people in the organization may choose a path that is not advantageous for the organization (Harrison & Freeman, 2004).

It is consequently not enough to generalize about democracy in organizations, and we should distinguish between different types of hierarchies (as they were found to be in place by this work) and between the contingencies under which a certain type of hierarchy makes sense or not (Kerr, 2004). In general, a democracy has been found to be
most suitable for organizations confronted with an ever-changing environment and that aim at generating innovative outputs, since a democracy allows for greater variance in human networks and knowledge relationships, in resources, or in peoples’ cognition and ways of thinking (e.g., Weick & Sutcliffe, 2001). The typical modern organization, however, seems to rely on the traditional organizational model that is based on formal structures and on an authority-based hierarchy (de Jong & Witteloostuijn, 2004). This fact is especially striking, since it is questionable if there is any organization in developed economies that is not situated in a fast-paced environment and that is not urged to generate innovations. Future work should take a closer look at why so many organizations are reluctant to adopt more democracy for their own work. It may be that there are other crucial factors in place that hinder organizations from becoming democratic that have as yet not been taken into consideration,

In addition to this, it would be most interesting to compare the ranking that emerges from the traditional principle of a hierarchy based on bureaucracy, authority, and on bundled property rights as we know it from many firms, to the type of hierarchy as it was discovered in this study. Do both mechanisms lead to the identical hierarchical structures, and do the people who know the most about their organizations occupy the highest positions in traditional firms? If not, what could be the reasons for and consequences of this finding? If yes, do firms really need all of their bureaucratic administrative processes? One possible result of such research might be that various organizational forms are less different than we are actually aware of to date, or that their differences are grounded in factors that have not yet been analyzed.

Finally, the findings of this study have revealed that we can speak of a sort of hierarchy in the open source setting, and of a relatively stable group of people who are at its pinnacle. This characterization comes close to Hambrick’s definition of a corporate top management team, which is a “relatively small group of most influential executives at the apex of the organization” (1994: 173). Even if it is not intended, the individuals who are at the top of the organization exercise more power than others, which (again) puts the open source projects’ entirely democratic character in question. In addition, this finding poses the question of whether leadership takes place in this setting, or whether the composition of open source communities’ ‘top teams’ is relevant for the projects’ success, which are both promising topics for future research.

246
Endnotes Part 3

1 For abridged version of the theory-generating part of this paper see Kugler 2005 a&b.

2 In order to avoid unnecessary overlaps between the chapters of this dissertation, I kept the first part on the description of the methods as employed for this study (inductive single case on the FreeNet project) at its minimum. Data sources and gathering overlap with those as described in Paper 2 of this dissertation. For more detailed information on the methodological proceeding see the methods part there.

3 I could not find any significant differences in attention in an analysis of the threads initiated by core developers versus peripheral members. In 2000, a total of 1,714 threads were initiated, The 30 core developers initiated 1,047 (61.09%) of the threads, of which 276 (26.36%) received no reply, The 326 peripheral members, in comparison, initiated 667 (38.91%) email threads, of which 186 (27.89%) received no reply. Within the core developer group slight differences could be found. Of the four most active (upper-level) core developers, on average 27.5% of email threads remained unanswered, whereas of the remaining 26 core developers 32.6% of email threads received no reply.

4 The contrasting discussion of core developer behavior and characteristics versus that of peripheral project contributors aims at demonstrating tendencies. Within both member type groups however, differences between the contributors could be found, which indicates that their hierarchical positions in the project take place rather on a continuum than in two discrete, hierarchical groups.

5 No exact figures have been available on the number of new general documents, such as FAQ, papers, presentation foils, or other publications on FreeNet’s general direction in 2000.

6 FreeNet’s email posts and CVS commits in 2000 evolved as follows:

<table>
<thead>
<tr>
<th>Month 2000</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emails posted to the list /month</td>
<td>269</td>
<td>485</td>
<td>676</td>
<td>1,371</td>
<td>785</td>
<td>1,193</td>
<td>710</td>
<td>1,199</td>
<td>1,220</td>
<td>1,467</td>
<td>1,051</td>
<td>784</td>
</tr>
<tr>
<td>Accumulated no. of emails</td>
<td>269</td>
<td>754</td>
<td>1,430</td>
<td>2,801</td>
<td>3,586</td>
<td>4,779</td>
<td>5,489</td>
<td>6,688</td>
<td>7,908</td>
<td>9,375</td>
<td>10,426</td>
<td>11,210</td>
</tr>
<tr>
<td>Code logs added to the CVS / month</td>
<td>44</td>
<td>105</td>
<td>138</td>
<td>229</td>
<td>67</td>
<td>43</td>
<td>83</td>
<td>147</td>
<td>120</td>
<td>191</td>
<td>51</td>
<td>26</td>
</tr>
<tr>
<td>Accumulated no. of code logs</td>
<td>44</td>
<td>149</td>
<td>287</td>
<td>516</td>
<td>583</td>
<td>626</td>
<td>709</td>
<td>856</td>
<td>976</td>
<td>1,167</td>
<td>1,218</td>
<td>1,244</td>
</tr>
<tr>
<td>Ratio (%) of each accumulated no. of emails and no. of code logs</td>
<td>6.11</td>
<td>5.06</td>
<td>4.98</td>
<td>6.95</td>
<td>6.15</td>
<td>7.63</td>
<td>7.7</td>
<td>7.81</td>
<td>8.1</td>
<td>8.0</td>
<td>8.56</td>
<td>9.0</td>
</tr>
<tr>
<td>Ratio (%) of each absolute no. of emails and no. of code logs</td>
<td>6.11</td>
<td>4.62</td>
<td>4.91</td>
<td>5.99</td>
<td>11.72</td>
<td>27.74</td>
<td>8.55</td>
<td>8.16</td>
<td>10.17</td>
<td>7.68</td>
<td>20.61</td>
<td>10.15</td>
</tr>
</tbody>
</table>

Table 3.31: Monthly increases and accumulated numbers of FreeNet’s emails and CVS code logs in 2000.

7 Von Krogh et al. (2003b) also report about the joiners (which are equal to peripheral members) levels and types of activity, which are crucial dimensions of their joining scripts, through which they become a member of the developer community.
Within the core developer group, there is great variance with respect to the amount of created knowledge: Email contributions on the developer mailing list; The four most active core developers together account for 50.48% (sd, 18.48) of all email contributions (63.82% of all core developer emails); thread initiation; The four most active core developers initiated 34.19% (sd, 118.56) of all threads on the developer mailing list (55.97% of all threads initiated by core developers); CVS commits; The four most active core developers account for 52.73% (sd, 13.47) of all code commits (by core developers) in 2000. The four most active core developers additionally entered the project on average on day 28 of the year 2000 (Jan, 28), with three of them writing their first email in the first five days of the year, and the fourth contributor submitted his first email on April 12.

The problem of organization-specific knowledge that vanishes when a developer leaves the organization was also reported to occur in software development firms. One contributor has illustrated: “At the companies I’ve worked at, I have had the luxury of not having to deal directly with the several million lines of code that other people have to deal with day-to-day. In my experience in industry, software engineering is nonexistent, and this would be hard to fix. A guy often starts the project, and writes very high quality code, very well separated, and nearly bug-free. But then the project grows, the guy leaves the company, and other people are left with the worst possible way, just because nobody had such a good grasp of the entire code base, except for the guy who left. If you have a good discipline and write up interface guidelines, design by contract, and so on, what often happens when the head guy moves on is that all goes straight out of the window, because the poor chaps who have to deal with the code don’t know enough to do it right. It just doesn’t work at the places I’ve been at.” (Peripheral member #1).

Von Krogh et al. (2003b) report more in detail on Freenet’s weekly number of posters.

Both knowledge absorption and creation after a contributor’s entry to a project cover firstly, their time investments and, secondly, the number of knowledge transfer media they used therefore, which I will take in consideration for the concrete variable operationalization.

Read; Question no. 3 in the web-based survey. All survey questions are available in the appendix.

Read; Average, [range].

Each of the four open source communities studied (CrystalSpace, HSQLDB, StepMania, TikiWiki), uses various mailing lists and/or forums for their discussions, which can all be found on their websites; I focused exclusively on the communities’ developer mailing lists in order to compare the study findings between these projects, and to compare the findings with the results from the first order analysis (Freenet project) of this paper, I assume therefore, that the members of different open source communities use their mailing lists and forums in a comparable way.

For the knowledge hierarchy model, the mediator variable can be aggregated as described below. In fact, I could not calculate the aggregated variable due to a lack of variance in the responding group, as is described in the paper’s paragraphs on data analysis.

\[ m_\text{ag} = \frac{\sum x \sum a_x + \sum x \sum b_x}{\sum x (\sum a_x + \sum b_x)} \]  

\[ m_\text{ag} \]: Share of an individual x’s accumulated organization-specific knowledge (absorption & creation) of the entire in the project accumulated amount of organization-specific knowledge at a certain point of time t (time investment in hours),

\[ t \]: Point of time during life cycle of OSS project, in days since project initiation, \( t = [1] \).

\( x_i \): Individual contributor to the organization \( x = [1,n] \); \( n = \) total of all contributors to the organization.
a.: Time invested in knowledge absorption activities before and after project entry by individual contributor x.*

b.: Time invested in knowledge creation activities before and after project entry by individual contributor x.*

k: Knowledge transfer medium, k = [1, 2, 3, 4]; 1: mailing list; 2: code repository; 3: general documents; 4: other.

16 Since only 4 of HSQLDB’s contributors returned the survey, I will mainly refer to the other three projects.

17 In the two-sample case, the discriminant function equals a regression function and it reads:

\[ y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 \]

with:

- \( y \): discriminant/canonical variable
- \( b_j \): discriminant coefficient (j)
- \( x_k \): independent variable (j = 1, 2, 3)
- \( b_0 \): constant.

18 Although the sample covers both core developers and peripheral members in my findings, I will refer to the core developers only. Firstly, they make 84% of the respondents and, secondly, I assume that the participating peripheral group members are rather comparable to the core developers than to the typical peripheral group members, otherwise they had not participated in the study.

19 The figures in brackets reflect the average result gained through the survey (see the appendix).

20 The correlations between use of the various knowledge transfer media for knowledge absorption before project entry are as follows (coefficient / (D.F.) / 2-tailed sig., * sig. level .05, ** sig. level .01):

<table>
<thead>
<tr>
<th></th>
<th>mailing list</th>
<th>CVS</th>
<th>general docs</th>
<th>private disc</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailing list</td>
<td>1.0</td>
<td>1.716 (44)</td>
<td>1.187 (44)</td>
<td>1.629 (44)</td>
<td>4.762** (44)</td>
</tr>
<tr>
<td>CVS</td>
<td>1.0</td>
<td>1.399 (44)</td>
<td>2.356 (44)</td>
<td>0.800 (44)</td>
<td></td>
</tr>
<tr>
<td>general documents</td>
<td>1.0</td>
<td>4.378** (44)</td>
<td>3.730* (44)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>private discussion</td>
<td>1.0</td>
<td>3.172* (44)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 332: Correlations between the use of knowledge transfer media for knowledge absorption before project entry.

21 The correlations between use of the various knowledge transfer media for knowledge absorption post project entry are as follows (coefficient / (D.F.) / 2-tailed sig., * sig. level .05, ** sig. level .01):

<table>
<thead>
<tr>
<th></th>
<th>mailing list</th>
<th>CVS</th>
<th>general docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailing list</td>
<td>1.0</td>
<td>4.457** (47)</td>
<td>2.639 (47)</td>
</tr>
<tr>
<td>CVS</td>
<td>1.0</td>
<td>0.065 (47)</td>
<td></td>
</tr>
<tr>
<td>general documents</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 333: Correlations between the use of knowledge transfer media for knowledge absorption post project entry.
Dissertation Summary

&

Conclusions
Dissertation Summary & Conclusions

The aim of this dissertation was to shed light on open source software development projects, a new form of organizing that seems to be capable of bringing about innovative outputs. Thereby I focused on the issue of coordination, since in this setting integration cannot be generated by means of traditional formal or informal mechanisms we are already familiar with from organizations on the market-firm continuum. I structured my work in three separate, but complementary parts, each of which has the character of a separate paper that (theoretically) could stand alone.

In the first part of this dissertation, I approached the overall topic in question from a theoretical perspective in order to explore the peculiarities of coordination in open source software development, I used existing theory in the tradition of economics, organization theory, and social network theory. The discussion leads to conclude the following, firstly, formal coordination mechanisms are bound to fail in the open source setting, since their output is a public instead of a private good. Secondly, the total amount of coordination that is required to achieve unity of effort in an average open source project is smaller when compared to an average software development firm. The reason therefore is that the contributors to an open source project focus exclusively on explorative tasks, whereas supportive or administrative tasks play no crucial role. Finally, the specific open source setting enables the combination of stability and volatility, which are the integral components of dynamic capabilities (Teece et al., 1997). The different roles and behaviors of two developer types (core developers and peripheral members) to open source communities enable this.

The dissertation's second part approached the overall topic to be analyzed through a largely unprejudiced, inductive proceeding. By means of analyzing the single case of the Freenet open source project I could generate the following findings, coordination in this setting is entirely informal and it is generated by means of the contributors' shared (implicit and explicit) project-specific knowledge of rules and standards. The knowledge covers three distinct dimensions, namely technical standards, community rules and the project's overall direction. Knowledge that is capable of
coordinating an organization comes thereby to existence through an evolutionary process of knowledge creation, capturing & replication, and transfer.

For the dissertation’s third part I employed a two-step methodological procedure to generate and partly test a model of a knowledge hierarchy in open source software development projects. My findings indicate that in this setting a hierarchy is prevailing that is entirely informal, and that is grounded in the contributors’ amounts and types of knowledge about the community they are contributing to. In general, the earlier in a project’s life cycle a contributor enters a certain community, and the more knowledge he or she absorbs and creates while contributing to it, the higher will be the position in the knowledge hierarchy.

These findings all could contribute to a better understanding of the relationship between coordination and innovation in open source software development. But also, the findings lead to a number of questions on a more general level of analysis, I will briefly discuss the following, firstly, ‘are open source software development projects a new, special type of organization or institution?’ Secondly, ‘what are the learnings and consequences of the findings derived out of this dissertation for organization theory?’ Finally, what are the consequences for traditional forms of profit- or non-profit organizations on the market-firm continuum?’ I will limit this section to my own thoughts.

Are open source software development projects a new, special type of organization or institution? This question is especially interesting, since open source projects have characteristics we are already familiar with from other settings, for example, project work (management consulting), communities (sports clubs, fan clubs), virtuality (email contacts to colleagues or friends overseas or even next door), voluntariness (working for sports clubs or with kids), the production of a public good (art museums), a focus on knowledge and innovative output (research & development), just to name a few. So what is really new about open source projects?

Although the above-mentioned elements are well-known to us, it is their new combination and merging and the resulting complexity, which characterizes open source communities, and which is new for us. In open source projects, a number of well-known
elements of other working environments are being meshed, so that something new could emerge (which is, in fact, also the idea of the generation of innovations). These communities are therefore both new and well-known to us.

One crucial prerequisite for the success of open source communities has been the rise and commercialization of the internet and of computer technology in the late 1980s and 1990. Therefore, it is not surprising, that Linux’ outstanding growth and success paralleled the rise of the internet. Those types of collaboration that the computer and internet technologies enabled, however, are still relatively young when compared to the traditional modes or working, and therefore, we are less familiar with their use or consequences that result out of it, Expectably, other new characteristics of organizing will come up in the future, and we will also be confronted with alternative new combinations of these characteristics and the resulting new organizational forms. We will therefore permanently face the challenges and opportunities of their functioning and the need to learn how, why and under what conditions new forms of organizing will make sense.

What are the learnings and consequences of the findings derived out of this dissertation for organization theory? The first part of this work approached the dissertation’s topic by means of a number of traditional theories. These could help to frame the open source phenomenon from different loosely-coupled perspectives and so help to better understand this form of organizing. These traditional theories could not, however, explain how open source projects are coordinated, nor could they indicate the existence of a knowledge hierarchy. Traditional theories that were build through observations in traditional organizations are therefore valuable instruments to explain a new phenomenon, but they can only tell half of the truth, since they cannot grasp it in its entirety. New phenomena can often not entirely be captured through old constructs, frames or expressions.

We have, consequently, to be careful in adopting old theories for new concepts, since they are retrospective in nature, and they have only limited value in explaining phenomena that are different from what we had before. Categorizing new phenomena wrongly or incompletely can, in fact, lead to misleading conclusions and, finally, to wrong decisions. Using old theories for new things seems thus to make sense only, if new
things are not really new. Alternatively, there is a need to adapt our traditional organization theories to the new reality. The occurrence of new phenomena calls therefore for exploratory research methods that are more suitable for the generation of new constructs, new models, or new theories that are capable of better describing things we have not faced have before, as it was done in this dissertation.

*What are the consequences for traditional forms of profit- or non-profit organizations on the market-firm continuum?* Traditional organizations, profit-oriented or not, can learn from open source projects that not only the output they generate can be subject to innovative ideas, but also the way they are structured or organized. And in fact, these characteristics seem to be closely interrelated. Observing open source software development, traditional organizations can also learn that less bureaucracy (and therefore, less costs) is possible for organizations that are involved in processes of technological innovation. These aspects are not new, nevertheless, organizations are still having difficulties changing, taking up or adapting to new environments and realities. This challenge is most salient in established, large organizations that become more rigid with growing size, established routines, linkages to external partners or cognitive inertia. Organizational change processes require a lot of resources and they may lead to a trade-off between an organization’s short- and longer-term success. The question is thus not if organizations should change, rather it is how they can best do so permanently, and less bureaucracy or more democracy seem to be valuable instruments to achieve these goals.

Additionally, the dissertation has illustrated, that open source communities are subject to very specific rules, structuring mechanisms, and laws. Any individual or firm that wishes to participate in this type of software generation must necessarily observe and understand what is going on in this setting, and for what reasons. To become respected as a full member of a community, and to influence its direction, individuals need to capture open source communities’ very nature, and this work could shed some light on the subtle mechanisms that are in place in this setting. Active community members, project administrators or initiators should do the same, since they must be aware of what determines the continuous flow and a longer-term stock of contributors to a certain project, as well as what enables or hinders individuals to contribute to a community.
References
References


German Human Genome Project (2003): www.dhp.de/


Osterloh, M.; S. Rota; B. Kuster (2003): Open Source Software Production: Climbing the 
Shoulders of Giants. Working Paper, University of Zurich, Switzerland.

Vol. 25, 129-141.

Version 3/01.


Voices from the Open Source Revolution. O’Reilly and Associates; Sebastopol, CA, 
171-188.

standtogether.html.


Companies, Harper & Row, New York, NY.

Pfeffer, J. (1981): Management as symbolic Action; The Creation and Maintenance of 
organizational Paradigms. L.L. Cummings; B.M. Staw (eds.): Research in 


Popo, L.; T. Zenger (2002): Do formal Contracts and relational Governance function as 

B.M. Staw; L.L. Cummings (eds.): Research in Organizational Behavior Vol. 12. JAI 

Projekt Gutenberg Website (2003); http://gutenberg.net.


Raymond, E.S. (1999b): The Cathedral and the Bazaar: Musings on Linux and Open Source by an accidental Revolutionary, O’Reilly and Associates, Sebastopol, CA.


Weick , K.E. (1979): The social Psychology of Organizing. Addison-Wesley, Reading, MA.


Appendices
Appendix 1
The Open Source Definition

The distribution terms of open source software must comply with the following criteria:

1. Free Redistribution
The license may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a loyalty or other fee for such sale.

Rationale: By constraining the license to require free distribution, we eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure for cooperators to defect.

2. Source Code
The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost – preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

Rationale: We require access to un-obfuscated source code because you can’t evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification must be made easy.

3. Derived Works
The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

Rationale: The mere ability to read software isn’t enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.

---

1 The Open Source Definition is published at www.opensource.org/docs/definition.html.
4. Integrity of the Author’s Source Code

The license may restrict source code from being distributed in modified form only if the license allows the distribution of ‘patch files’ with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software build from modified source code. The license may require derived works to carry a different name or version number from the original software.

Rationale: Encouraging lots of improvements is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they’re being asked to support and protect their reputations. Accordingly, an open-source license must guarantee that source be readily available, but may require that it be distributed as pristine base sources plus patches. In this way, ‘unofficial’ changes can be made available but readily distinguished from the base source.

5. No Discrimination against Persons or Groups

The license must not discriminate against any person or group of persons.

Rationale: In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open source license from locking anybody out of the process. Some countries, including the United States, have export restrictions for certain types of software. An OSD-conformant license may warn licenses of applicable restrictions and remind them that they are obliged to obey the law; however, it may not incorporate such restrictions itself.

6. No Discrimination against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Rationale: The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it.
7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

Rationale: This clause is intended to forbid closing up software by indirect means such as requiring non-disclosure agreement.

8. License must not be specific to a Product

The rights attached to the program must not depend on the program’s being part of a particular software distribution. If the program is extracted from the distribution and used or distributed within the terms of the program’s license, all parties to whom the program is distributed should have the same rights as those that are granted in conjunction with the original software distribution.

Rationale: This clause forecloses yet another class of license traps.

9. License must not contaminate other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open source software.

Rationale: Distributors of open source software have the right to make their own choices about their own software. The GPL is conformant with this requirement, GPLed libraries ‘contaminate’ only software to which they will actively be linked at runtime, not software with which they are merely distributed.
Appendix 2

Interview Partners: Name and Date

**Software Industry and Entrepreneurship Experts (Face-to-face and telephone interviews):**

Michael Jäger, CEO Hilogix GmbH  
(Software & Software Consulting), CH  
February 4, 2002

Hermann Arnold, CEO Brains-to-Ventures  
(V Venture Capitalist & Software), CH  
February 8, 2002

Patrik Steinmann, Software Engineer, Brains-to-Ventures  
(V Venture Capitalist), CH  
February 11, 2002

Dr. Daniel Schlegel, Partner Accenture (Consulting), CH  
February 11, 2002

Unda Karshaus, Consultant Accenture (Consulting), D  
February 14, 2002

Fabian Hediger, CEO Beecom AG (Software), CH  
February 14, 2002

Prokop Kazil, COO Beecom AG (Software), CH  
February 14, 2002

**Contributors to the Freenet OSS Project, first Round (Telephone interviews):**

Ian Clarke (Project Initiator)  
October 25, 2000

Oskar Sandberg  
November 15, 2000

Scott Miller  
November 24, 2000

Adam Langley  
December 26, 2000

Stephen Blackheath  
December 27, 2000

Steven Hazel  
January 1*, 2001

Paul Kappler  
January 23, 2001

Sebastian Späth  
January 25, 2001

---

2 Interviews with the Freenet core developers (first and second interview rounds) were led in the context of a research collaboration between the University of St.Gallen and MIT. They were led by the following persons: Simon Grand, Eric von Hippel, Karim Lakhani, Georg von Krogh, Petra Kugler, Sebastian Späth (in alphabetical order), who asked the questions listed in appendix 3.
Contributors to the Freenet OSS Project, second Round
(Telephone and face-to-face interviews):

Ian Clarke (Project Initiator) March 2nd, 2001
Scott Miller March 1st, 2001
Steven Hazel March 1st, 2001
Theodore Hong March 1st, 2001
Mr, Bad May 2nd, 2001
Sebastian Späth November 20, 2001

Contributors to the Freenet OSS Project, third Round
(Open questions survey and online chat):

Sebastien Loisel September 8, 2003
Paul Jimenez September 9, 2003
Jeremiah Wessels September 15, 2003
Joseph Solbrig September 19, 2003
Marc Schneiders September 21, 2003
David Schutt September 28, 2003
Peter Harkins September 29, 2003
Lawrence Leung September 29, 2003
Aaron Hamid September 30, 2003
Jay McCarthy September 30, 2003
Andy Rabagliati October 6, 2003
Tom Brown October 6, 2003
Arend Lammertink October 8, 2003
Fred Salzer October 9, 2003
Jim Gallagher October 20, 2003
David Weekly October 20, 2003
Michael Rogers October 22, 2003
Roger Dingledine October 27, 2003
John Cole October 29, 2003
Will Dye November 6, 2003
Appendix 3

Guidelines for Interviews with Freenet Core Developers (First and Second Round)

1. Personal Background
Could you give us some information about your background, education and current situation of employment?
How old are you?
Do you have any experiences in software programming?
What programming languages are you familiar with?
Did you have to learn any new programming languages or other capabilities to be able to contribute to Freenet and to understand what is going on in the project?

2. Motivation to contribute to an OSS project in general and more specifically to Freenet
What is your motivation to contribute to the project (e.g. fun, the final product, technical challenge, contacts within the community, etc.)?
Does a contribution to the project create any costs for you?
Could you describe these costs?
Do you have any personal benefits from contributing?
What benefits do you have?
Do you have any personal use for the final software product?
What is it that makes contributing to this specific project so exciting for you personally?
How did you learn about the project in the first place?
Why did you decide to dedicate your contributions to this specific project in the universe of OSS projects?
Do you also contribute to other OSS projects? To which ones?
What distinguishes this project from other OSS projects?
Under what circumstances would you leave the project?
Would you turn to another open source project? To which one?
3. Role and involvement in Freenet
Could you briefly describe your role in the project?
How much time do you usually spend working on the project?
How do you spend this time?
What type of work is most time-consuming for you?
Do you feel personally responsible for the code that you are contributing?
What is it from a technical perspective that makes the project challenging for you?
Do you prefer working with many people or with a small group of contributors?

4. Entry and exit to/ from Freenet
Do you have a complete overview of the project?
Could you describe the process of becoming a core developer?
About how long did this process take?
Do you remember any major events during this process?
How long did the process of learning about the project take?
Do you see any entry barriers for newcomers to the project?
Could you name some entry barriers?
How could one overcome these entry barriers?
Are there any exit barriers, if so which ones?
What would happen if someone left the project?

5. Coordination and Community
Could you briefly describe the decision-making process in the project?
What happened if conflicts come up?
How are conflicts resolved?
Do you feel well-coordinated in Freenet?
How would you describe Freenet’s organization and coordination?
Have you ever used other than electronic devices to communicate with your fellow project contributors?
Have you ever met anybody in person?
Do there any procedures, rules or routines exist in the project?
Do you think that every contributor to the project is familiar with these procedures, rules or routines?
Could you give an example of these procedures, rules or routines?
How does communication usually take place in Freenet?
Which channels are used therefore?
Do you think that some kind of hierarchy is prevailing in the project?
Could you describe the prevailing hierarchy?
How would you describe the ‘typical way’ that a thread/piece of code has to take before it is integrated into the CVS?
Do you think that the project is currently proceeding into a coherent direction?
Does communication in the project take also place on the basis of private emails?
What types of topics do you usually communicate on a private basis and for which topics do you use the public mailing list?
Do you get any feedback on your code or email contributions?
How does feedback happen?

6. Community-related aspects and identification of distinct groups
Do you think that the project would work similarly in a competitive/company environment?
What is different, what is similar between Freenet and a firm?
What would happen if the community grew strongly and extremely fast?
Are there any contributor groups, which you are able to identify?
How would you characterize the project’s core developers?
What types of behaviors are typical for the members of this group?
What distinguishes a core developer from a non-core developer?
How would you describe central roles within the project?
How do you treat free riders/lurkers within the project?
Is this project modular, so that some kind of specialization can take place?
Does any specialization take place in the project?
How do you identify a good programmer?
What would you do if you recognized a bad programmer?
Who are the most important (key) individuals within the project?
Appendix 4

First Contact Email to Freenet Peripheral Contributors

Dear …

I have found your email address on the development mailing list of the Freenet OSS project to which you have contributed in a rather infrequent manner.

I am working on a research project at the Swiss University of St.Gallen (www.unisg.ch), which seeks to find out how coordination is achieved in OSS without relying on formal or traditional coordination mechanisms we can usually find in firms. It could be that the way OSS projects organize themselves fosters organizations’ innovation capability. However, there is still a great need to find out how this process functions in detail—and this subject is a core part of my doctoral dissertation.

In collaboration with Prof. Eric von Hippel and Karim Lakhani from MIT and Prof. Georg von Krogh, who is my thesis supervisor, I have already looked at Freenet for quite some time, and we have talked to some of the current and former Freenet core developers, among others Ian Clarke, Oskar Sandberg, Scott Miller, Theo Hong. For my work it is, however, important also to give the rather infrequent contributors a voice. Please be sure that my work is absolutely serious research, and you might already know our OSS research website that came out of the above-mentioned collaboration (http://opensource.mit.edu).

I would appreciate it a lot if you answered a handful of questions on your Freenet contributions, even if they have been small in number (this is exactly what I am interested in!). I want to invite you to getting in touch either by phone (no longer than half an hour), by online chat or I could simply send my questions to you by email, whatever you prefer. If you wish so, I will also guarantee that your responses will be treated absolutely anonymously.

Please let me know, if you are interested in some additional information of who I am or on what I am doing. In any case I look forward to hearing from you.

Kind regards, Petra Kugler
Appendix 5

Open Questions Survey sent to Freenet Peripheral Members (Third Round)

Most of the following questions are rather open, Feel free to answer whatever you think you have something to say to, and say as much as you wish to answer. Some questions will simply ask for a ‘yes’ or ‘no’, and for others a more differentiated answer will be appropriate. Feel free to write into the file and to use as much space as you require therefore. Be sure that, no matter how detailed your answers will be, I do appreciate it a lot!

1. Personal Background

Could you give some short information about yourself, your education and your current situation of employment or your studies?

What is your current position of employment or education? How old are you?

Do you have any experiences in software engineering or in a related field? Which programming languages do you know?

Did you already have these experiences when contributing (code, email discussion, or any other thoughts) to Freenet?

If yes, was this knowledge capable of helping you to understand Freenet? How? In what terms could it not help any further?

2. Motivation for choosing Freenet

I am interested in your motivation for contributing to an OSS project in general, and more concretely for choosing the Freenet project.

Why did you decide to contribute (code, email, or any other thoughts) to an OSS project?

Did or do you also contribute to or use any other OSS projects? If yes, how strong was or is your involvement in these projects?

Why did you pick Freenet out of the universe of OSS projects? How did you first learn about it? Do you also remember about when this was?

Do or did you (intend to) use the software developed in Freenet? Did you get any other benefits from the project in return (e.g., learning effects, contacts, reputation, etc.)?
What were the reasons for you to stop contributing (code, email discussion, or any other thoughts) to Freenet? Are you intending to re-enter the project?

3. Your Role and Involvement in Freenet

*I would like to learn a bit about the intensity and type of your involvement into the Freenet project.*

Could you briefly describe the role that you took over in Freenet? What other roles could you recognize in the project?

Did you focus on some specific tasks or topics? Which ones? Why these?

Did you feel that you had fully understood the entire project or did you rather understand some parts of it? What is it that you could not understand? Why not?

About how much time did you spend on Freenet (as a whole or per week, per month, etc.)? How did you spend the time (reading code/emailing, or contributing, etc.)?

4. Entry to Freenet, Entry and Mobility Barriers

*The goal of the following questions is to learn more about the process of how and when you had entered the project and how you had learned what was going on in Freenet.*

Could you briefly describe how you had entered the project? What did you do after you had learned about Freenet and after you had found it interesting?

Did you first read the email discussion, the code or other documents like Ian’s thesis or newspaper articles? If yes, do you remember how long this took? What was it that you have been able to learn (technical stuff, rules of behavior, Freenet’s goal, etc.) at that time?

Do you think there are some essential things that you can only learn while being on the project? What is it?

Do you think the members who have been contributing to Freenet since the project’s very initiation in early 2000 had any advantages over later entrants? Could you name these advantages?

Did you feel that there were any entry barriers for new developers? Which ones? How could one overcome the barriers?

Could you also recognize any mobility barriers (e.g., taking over a certain task, becoming a core developer, etc.)?

What would one have to do to become a core developer?
5. Coordination

I am interested in the way coordination took/ takes place in Freenet, in the prevalence of any kind of (formal or informal) hierarchy and in the kinds of rules or norms that you have been able to recognize.

How do you think does/ did coordination take place in Freenet? Did you feel well informed about what was going on?

What would happen, if a contributor missed reading the CVS or the discussion list for some time?

Could you find any hierarchy to be in place in Freenet? Did you always respect it? In what terms?

Who would you say was at the top of the Freenet hierarchy? How did you know that these persons were at the top? How would you characterize those at the top?

Have you been able to recognize any implicit or explicit rules or norms that guided Freenet? Could you give an example? How did you learn about these rules or norms? Are they the same in other OSS projects?

Is there anything else you would like to mention?

Thank you very much!!!
## Appendix 6

**Comparison of Qualitative Findings: Differences between Freenet Core Developers and Peripheral Members**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Peripheral members (less coordinated)</th>
<th>Core Developers (well coordinated)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Access to the code repository/ CVS</strong></td>
<td>No direct access.</td>
<td>Direct access,</td>
</tr>
<tr>
<td><strong>Project coordination</strong></td>
<td>Feel not or feel badly coordinated, know often not what, when or how to do,</td>
<td>Feel well coordinated, know what, when or how to do,</td>
</tr>
<tr>
<td><strong>Knowledge of project-specific rules in general</strong></td>
<td>No or only partial knowledge of project-specific rules,</td>
<td>Complete knowledge of project-specific rules,</td>
</tr>
<tr>
<td></td>
<td>Have predominantly knowledge about explicit rules.</td>
<td>Have knowledge about explicit and implicit rules,</td>
</tr>
<tr>
<td><strong>Recognition of rules, which can only be learned while being on the project</strong></td>
<td>No/ partial recognition,</td>
<td>Full recognition.</td>
</tr>
<tr>
<td><strong>Integration of the contributors' efforts to the project</strong></td>
<td>Often no/ bad integration,</td>
<td>Mostly good, full integration.</td>
</tr>
<tr>
<td><strong>Knowledge about the project’s overall direction, technical standards, community rules</strong></td>
<td>Generally no knowledge of all three knowledge types/ dimensions.</td>
<td>In general knowledge of all three knowledge types/ dimensions,</td>
</tr>
<tr>
<td></td>
<td>No recognition of any necessity for knowledge about all three types/ dimensions.</td>
<td>Recognition of the necessity for knowledge about all three types/ dimensions.</td>
</tr>
<tr>
<td></td>
<td>Know ‘what’.</td>
<td>Know ‘what and why’.</td>
</tr>
<tr>
<td><strong>General vs. specific knowledge of project-specific rules</strong></td>
<td>Either very broad project overview, but insufficient knowledge of project details, or very detailed knowledge of the project, but no project overview.</td>
<td>Broad project overview and deep knowledge about the project’s details.</td>
</tr>
<tr>
<td>Use of all types of knowledge storage and transfer media in Freenet: mailing list; code repository/ CVS; general docs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Only partial (active and passive) use of knowledge storage and transfer media, mostly no use of code repository/ CVS.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Active and passive) use of all knowledge and transfer media,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution to creation of project-specific knowledge about community rules, overall direction, technical standards</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Only minor contributions and knowledge but about all knowledge types, creation of path-dependent knowledge.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Major and minor contributions, mostly knowledge of all knowledge types, creation of path-creating knowledge.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Point of time of entry to the project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project entry during very early project stages,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project entry during later project stages,</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Appendix 7

**Comparison of Quantitative Findings: Differences between Freenet Core Developers and Peripheral Members**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Peripheral members (less coordinated)</th>
<th>Core Developers (well coordinated)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of contributors in this group in 2000</strong></td>
<td>326</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>(91.6%)</td>
<td>(8.4%)</td>
</tr>
<tr>
<td><strong>Total number of emails submitted to the developer mailing list in 2000</strong></td>
<td>4,459</td>
<td>6,751</td>
</tr>
<tr>
<td></td>
<td>(39.78%)</td>
<td>(60.22%)</td>
</tr>
<tr>
<td><strong>Average number of emails per contributor in 2000</strong></td>
<td>13.68</td>
<td>225.03</td>
</tr>
<tr>
<td><strong>Number of threads initiated in 2000</strong></td>
<td>667</td>
<td>1,047</td>
</tr>
<tr>
<td></td>
<td>(38.92%)</td>
<td>(61.08%)</td>
</tr>
<tr>
<td><strong>Average number of threads initiated per contributor in 2000</strong></td>
<td>2.05</td>
<td>34.9</td>
</tr>
<tr>
<td><strong>Number of threads participated in (2000)</strong></td>
<td>2,784</td>
<td>8,370</td>
</tr>
<tr>
<td><strong>Average number of threads participated in, per contributor (2000)</strong></td>
<td>8.54</td>
<td>279</td>
</tr>
<tr>
<td><strong>Average entry date to the project in 2000 (first email submitted to the developer mailing list)</strong></td>
<td>Day 197.3 (July 14)</td>
<td>Day 109.8 (April 18)</td>
</tr>
<tr>
<td><strong>Average longevity on the project in 2000, no. of days between first and last email submitted to the developer mailing list</strong></td>
<td>24.84 days</td>
<td>185.97 days</td>
</tr>
</tbody>
</table>

---

3 A part of the figures listed here emerged out of a research collaboration between the University of St.Gallen and MIT, and the following persons contributed to calculate them: Eric von Hippel, Karim Lakhani, Georg von Krogh, Petra Kugler, Sebastian Späth (in alphabetical order),
Appendix 8

First Contact Email to Project Maintainers:
Questionnaire on a Knowledge Hierarchy

Dear …

We are working on a large research project at the Swiss University of St.Gallen (www.unisg.ch), and the project aims at understanding how coordination is achieved in OSS projects. We seek to finding out what role the contributors’ shared and individual knowledge plays in this process both within and between projects. And finally, we are interested in learning more about the longer-term consequences of knowledge in OSS projects, One part of this research will be a central part of a doctoral dissertation.

Please be sure that our work is absolutely serious research, and you might want to take a brief glance at comparable studies at opensource.mit.edu. This forum for OSS researchers emerged out of a research collaboration between Eric von Hippel and Karim Lakhani from MIT and our team here in Switzerland, which is supervised by Georg von Krogh.

We have started with an extensive analysis of Ian Clarke’s Freenet project, and in the near future we will move beyond that community. So we came across your TikiWiki project, we found it very interesting, and we would be excited to take a closer look at your work. We prefer, of course, to do so with your ‘approval’ and support, since your knowledge of the project and your voice are of greatest value for our work and for us!

We seek to take a closer look at individuals’ submissions to the email discussion list, at CVS entries, but also at who has access to the CVS at which point in time. We are also planning to use a web-based survey. In ‘exchange’ to that, we would like to offer you the knowledge that we will gain during the research process, which reflects a neutral and external scientific perspective to your work.

So what is it that you could do for us? Not much, as we are entirely aware of the time constraints of OSS developers. It would be great, if you could forward a link of our survey to the contributors who are subscribed to your project developer mailing list. Also, we would like to ask you for your approval to analyze the project’s mailing lists and the source code. Please be sure that we will not uncover any contributor names, if a contributor wishes to remain anonymous, since every publication of our work will only refer to aggregated or anonymized results.
Please let us know if this is OK for you, or if you would like to receive further information on who we are or on what we are doing! We look forward to hearing from you,

Kind regards, Petra Kugler & Sebastian Späth
Appendix 9

Web-based Questionnaire Introduction

Who we are and what we are doing

We are a research team at the Swiss University of St,Gallen and conducting a large research project and a doctoral dissertation on OSS communities. Our work aims at understanding firstly, how coordination is achieved in OSS communities without relying on the traditional formal mechanisms we know from firms. We assume that knowledge that is shared by the project members plays a crucial role to bring together the work of individual project members. Secondly, we seek to understanding how, when and what kinds of knowledge are re-used within and across projects and in how far the knowledge re-use simplifies the process of coding software.

We have already discussed on both questions with members of the open source community, and they have demonstrated considerable interest in these topics. We would also like to give our knowledge back to you and would be delighted to forward our findings (e.g. as a paper) to all interested developers who contributed to our study as soon as our work will have reached its conclusion.

Privacy concerns

Our work is absolutely serious research and we will treat your answers anonymously, which means that no individual’s answer will be set in relation to any contributor name or email address, and every result of our work will appear only in an aggregated way. We must, however, ask you for your (nick-)names and email addresses, which we will only use for correlating purposes and for matching different types of data (e.g. email contributions, CVS entries, etc.).

Comparable work

You will find further examples of comparable scientific research on the OSS subject on opensource.mit.edu, a website you might already know. The website emerged out of a research collaboration between Professor Eric von Hippel and Karim Lakhani from Massachusetts Institute of Technology (MIT) and our research group in Switzerland which is supervised by Professor Georg von Krogh,
To whom we are addressing

Please note that all answers and comments of every individual who is involved in your project are of greatest value to us, no matter how much or how little you donate to the project, how frequent or infrequent you do so, or what type of contribution it is (discussion, code, bug reports, writing papers or FAQ, etc., or even just lurking or learning from the project). We also want to give a voice to the rather infrequent contributors and listen to what they have to say.

How you can help us

Some of the survey questions are rather open in nature. Just fill in as much as you feel you have to say, there is of course no limit. Other questions are closed in nature and they will require you to simply tick one or several of the small boxes. To complete the entire questionnaire will take about 20 minutes. In case of any questions or comments you might have, please refer to Petra Kugler, University of St.Gallen, Institute for Management, Dufourstrasse 40a, CH-9000 St.Gallen, petra.kugler@unisg.ch.

We look forward to reading your comments and answers, and we appreciate your help a lot. Thank you very much! :-) 

Petra Kugler, Stefan Häfliger, Sebastian Späth

---

4 The survey introduction is available at: http://blogs.ifb.unisg.ch/survey/notes.html,
The survey included questions that are no part of this dissertation (Q18-21).
Appendix 10

Web-based Questionnaire to OSS Project Members

Knowledge Sharing, Re-use and Coordination in OSS,
A Questionnaire to all Crystal Space Project Members,

Please find our questions below, there is a total of 24 questions of interest. Feel free to fill in all answers where you think you have something to say to, and leave a question out if you cannot give an answer. We will appreciate your responses and comments a lot! For additional information please feel free to take a look at:

http://blogs.ifb.unisg.ch/survey/notes.html

A. Personal Background and Related Knowledge.

Please indicate your name and email address. Although we will treat your answers entirely anonymous, we need these statements to correlate and match your comments with other project-related data, such as code entries or email contributions. No names or email addresses will appear in the results,

1. Your name:  

2. Your email address  
   (as used for the project):

3. Your nationality:

4. Your age:

5. Could you give us some brief information on your current situation of employment or your student status? Is your work/your studies in the field of software engineering? (Question is optional)

6. Do you have any additional background knowledge (e.g. prior studies, work, interns, or other involvement etc.) in the fields of:
   - Software engineering, coding work
   - Other natural sciences, e.g. Maths, Physics, Chemistry, etc,
   - None of these
7. How many programming languages do you know?

- None
- 1 programming language
- 2 - 5 programming languages
- 6 - 10 programming languages
- 11 and more programming languages

8. Besides the Crystal Space project, are you also involved in other Open Source projects?

- No, I am only involved in the Crystal Space project
- Yes, I am also involved in other OSS projects

9. If you are involved in additional OSS projects, besides Crystal Space, please indicate your role or roles in these other projects (tick one or several):

- Maintainer
- Core developer
- Infrequent developer
- Lurker
- User
- Other roles

10. If you are also involved in other OSS projects (besides Crystal Space), please think briefly when your involvement started, and indicate: "I did already contribute to other OSS projects..."

- Before joining the Crystal Space project
- Since around the same time when joining the Crystal Space project
- Only recently, after having joined the Crystal Space project

11. Do you take part or are you interested in other OSS-related activities and visit OSS conferences, read books or articles on OSS, take part in an ongoing discussion on OSS, etc.?

- Frequently
- From time to time
- Rarely or never

12. In case you do show interest in other OSS activities, please indicate what fits best to your situation: "I have this interest since about..."

- Before joining the Crystal Space project
- Since around the same time when joining the Crystal Space project
- Only recently, after having joined the Crystal Space project
B. Your Entry to the Crystal Space Project

13. Please think briefly and as good as possible of the time before you have entered the project through subscription to the mailing list. How did you learn about what was going on in the Crystal Space project? Please indicate also about how much time you have spent on each of the options listed below in to gather this information. (This is very important!)

1: overall 10 hours and below; 2: 11 to 50 hours; 3: 51 to 100 hours; 4: 101 to 150 hours; 5: 151 or more hours; 6: not at all/0 hours

1 2 3 4 5 6

... I browsed the developer mailing list
... I went through the source code/CVS
... I looked at the homepage, at the FAQ, at newspaper or online articles, foils and presentations, or at other general information on the project
... I communicated with other contributors on a private basis or chatted with them
... Other sources of information

14. "Looking back, these activities (Question 13) helped me to understand..."

1: excellent .... 5: not at all

1 2 3 4 5

... the Crystal Space project’s vision, its overall aims, and longer-term direction?
... desired and less accepted behavior within the Crystal Space project (unspoken or explicit)?
... project-specific technical standards?
... the Crystal Space project’s architecture and most critical features?
... the roles, ranks and specialization of individual project members, and of their contributions to the project?

C. Your Role and Involvement to the Crystal Space Project

15. Now being involved to the Crystal Space project, do you have a full overview and an understanding of what is going on/ changing in the project with regard to...

1: complete understanding; ....; 5: no understanding at all

1 2 3 4 5

...the project’s vision, its overall aims, longer-term directions, and changes made to it?
...changes or newer agreements on technical standards of the Crystal Space project, e.g., on what is considered as good code?
...changes or newly emerging rules, norms, or regulations
on the contributors’ behavior to be accepted as a full member of the Crystal Space project?

16. When reading and browsing the email list, source code or general documents, ...
(Please tick the option that applies best to you)

☐ ...I do focus on new and current contributions
☐ ...From time to time I do go back to older contributions, especially when I lack a specific piece of information
☐ ...I do frequently search or browse older contributions as it contains a lot of valuable information

17. Please remember now the time frame since your entry to the Crystal Space project (subscription to the mailing list). On average, how much time per week do you spend on ...

1: 10 hours and below per week; 2: 11-20 hours; 3: 21-50 hours; 4: 51-100 hours; 5: more than 100 hours; 6: no time spent/ 0 hours

...reading the development mailing list?
...posting to the development mailing list?
...going through the source code, reading it?
...writing code and patches, fixing bugs?
...reading general documents on the project, e.g. FAQ, newspaper or online articles, papers, foils, etc.?
...writing general documents on the project, e.g. FAQ, newspaper or online articles, etc.?

D. Knowledge ‘re-use’ between Projects (This section [Questions 19-22] applies only to contributors who donate code to the project.)

18. Do you frequently ‘re-use’ some code, patches or algorithms from other OSS projects? What kind of code do you ‘re-use’? When do you do so?

19. How do you search for appropriate pieces of code to be ‘re-used’? Where do you search for these pieces of code? And, how do you know that these pieces are appropriate for your own work? Could you briefly describe this process?

20. I you do ‘re-use’ some code, do you usually donate any kinds of credits to the ‘giving’ project or developer? If yes, how does that look like? And, will you usually quote the giving project or developer anywhere?
21. Could you state a brief example of where or when you have ‘re-used’ some code or algorithm?

E. Final Comments

22. Would you like to give any additional comments or do you have any questions on the OSS subject to which you would like to get an answer, and on which we should conduct future research?

23. Please indicate finally: "I am interested in receiving the results of the completed study by email..."

☐ Yes, please  
☐ No, thanks

Thank you very much for your help!

If you have further questions or comments, feel free to contact: 
Petra Kugler, Sebastian Späth, Stefan Häfliger
Nationality: German
Date of birth: 22. September 1968
Languages: German, English, Italian, French
Email: pekugler@web.de

Academic Education

*University of St. Gallen* (Switzerland)  
Institute for Management, *Doctoral thesis*

2000-05

*University of California, Berkeley* (USA)  
Institute of Industrial Relations, *Visiting Scholar*

2002

*Augsburg University* (Germany)  
*Diplom, Vordiplom & Baccalaureat* in Business Administration

1991-97

*Università degli Studi di Bari* (Italy)  
Faculty of Business Administration & Economics, *ERASMUS Scholar*

1993-94

Work Experience

*University of St. Gallen* (Switzerland)  
*Teaching and Research Associate*

1999-01

*Augsburg University* (Germany)  
*Teaching and Research Associate*  
*Student Teaching and Research Assistant*  

1998  
1996-97

*Nambu Kneissl Ski School, Hachi-Kogen* (Japan)  
*Seasonal Ski Instructor* teaching Japanese Children  

95, 97, 98

*Grey Advertising Agency, Düsseldorf* (Germany)  
*Summer Associate* (Consultant)

1994

*Schmittgall Advertising Agency, Stuttgart* (Germany)  
*Junior Consultant* (Text, Concept, Consulting) and *Apprentice*

1989-91

*K & K Advertising Agency, Stuttgart* (Germany), *Apprentice*

1989

*Alfred Kärcher GmbH & Co., Winnenden* (Germany)  
*Intern, Marketing & Advertising*

1988

*Evangelischer Reisedienst (ERD), Stuttgart* (Germany)  
*Ski instructor and coach* for children and adult camps across Europe  

Winter 1991-01