HIERARCHY IN MERITOCRACY

Community Building and Code Production in The Apache Software Foundation

Master's Thesis

Oscar Castañeda
HIERARCHY IN MERITORCRY

Community Building and Code Production in the Apache Software Foundation

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

MANAGEMENT OF TECHNOLOGY

by

Oscar Castañeda
born in Guatemala City, Guatemala

TU Delft
Policy, Organization, Law and Gaming (POLG)
Faculty Technology Policy and Management (TPM)
Delft University of Technology (TU Delft)
Delft, the Netherlands
tbm.tudelft.nl
HIERARCHY IN MERITOCRACY

Community Building and Code Production in The Apache Software Foundation

Author: Oscar Castañeda
Student id: 1398946
Email: oscar.castaneda@gmail.com
Graduation Date: December 2010
Graduation Section: Policy, Organization, Law and Gaming (POLG)

Abstract

This research is about code production in top-level open source communities of The Apache Software Foundation (ASF). We extensively analyzed Subversion repository logs from 70 top-level Apache open source projects in the ASF from 2004 to 2009. Based on interactions in code production during one-year periods we constructed networks of file co-authorship that gave us access to the organization of Apache open source communities. This allowed us to measure graph level properties, like hierarchy and clustering, and their influence on the outputs of code production.

Apache communities are groups of individuals that organize their code production efforts in order to develop enterprise-grade open source software. The ASF explains the success of its communities and the software they produce by claiming to have instituted a meritocracy that brings contributors together in a way that significantly influences code production, namely by building communities instead of only focusing on technical properties of the source code like modularity. Self-organization theory has found that the role of institutions is minor. In this research we test and confirm the theory of self-organization, and find that the meritocracy institution does not influence code production.

Thesis Committee:

Chair and first supervisor: Prof.dr. M.J.G. van Eeten, Faculty TPM (Section POLG), TUDelft
Second supervisor: Dr.ing. V.E. Scholten, Faculty TPM (Section TSE), TUDelft
Third supervisor: Dr. R. van Wendel de Joode, Twynstra Gudde
Dedicated to my parents, Margarita and Oscar – my first teachers.
Acknowledgements for Scholarships and Stipends

- Supported by the Programme Al\textbeta an, the European Union Programme of High Level Scholarships for Latin America, scholarship No. E07M401381GT.

- Con el apoyo del Programa Al\textbeta an, Programa de Becas de Alto Nivel de la Unión Europea para América Latina, beca No. E07M401381GT.

- Supported by the Google Summer of Code program 2009.

- Supported by the Google Summer of Code program 2010.

- Supported by the TPM Study Abroad Fund for attending CASOS Summer Institute at Carnegie Mellon University.

- Supported by The Apache Software Foundation’s Travel Assistance Committee for attending ApacheCon NA 2010.

- Supported by the Programme Al\textbeta an travel subsidy for attending the 3\textsuperscript{rd} Al\textbeta an Student Conference.
I would like to thank my supervisors, Dr. Michel van Eeten, Dr. Victor Scholten and Dr. Ruben van Wendel de Joode. Thank you Michel for your mentorship, patience and for your nurturing of creativity as well as for encouraging me to focus and for providing direction to my project. Thank you Victor for your numerous insights on data analysis, for always receiving me with a smile, and for your support and mentorship. Thank you Ruben for cultivating my interest in open source and introducing me to the wonderful challenge of research in the field, as well as for guiding my research and providing insightful and motivating questions.

During my studies and research I benefited from a full scholarship from the Alβan programme of the European Union. I wholeheartedly thank the Alβan programme and the European Union for the financial support I received which enabled me to attend Delft University of Technology. Thanks also to my scholarship manager Margarida Prata for her kind help and support. Thanks to Dr. Roland Ortt for supervising and advising me on my scholarship and study progress.

During the summer periods of my research I benefited from project sponsorship from Google’s Open Source Programs Office as part of their Summer of Code program. I would like to thank Leslie Hawthorn, Carol Smith and Ellen Koe for believing in my alternate project proposals and for supporting my work. The Google Summer of Code program provided me with a fun way to learn about open source. Special thanks to Nitin Bhide, my project mentor for Google Summer of Code and founder of SVNPlot – the open source project that enabled my research.

As part of my research I attended the CASOS Summer institute at Carnegie Mellon University in June of 2009. I attended thanks to the support of the TPM Study Abroad Fund. I wish to thank the selection committee for their support. Thanks to Ms. Toke Hoek for helping me to study abroad. I would also like to thank the CASOS researchers and students for their insights into social networking analysis research.

As part of my research I attended ApacheCon Europe 2008 and 2009 in Amsterdam and ApacheCon North America 2010 in Atlanta, where I recently
presented my research in a conference session and at the BarCamp unconference. I wholeheartedly thank Charel Morris and the Stone Circle Productions team for receiving me as a staff volunteer. Attending ApacheCon was hugely beneficial for my research and an incredibly fun experience. I would also like to thank The Apache Software Foundation and The Apache Travel Assistance Committee, for providing me with travel assistance and the opportunity to attend ApacheCon North America.

In my research I benefited from the ideas, contributions and experience of various open source enthusiasts, evangelists, activists, academics and developers. I would like to thank Karl Fogel who suggested an initial topic for my thesis and helped me work out initial ideas. Thanks also to Adriano Crestani, Jean-Sebastien Delfino and Luciano Resende, committers of Apache Tuscany – the open source project where I learned about Apache and *The Apache Way*. I would also like to thank Sally Khudairi from her friendly feedback during the Media & Analyst Training at ApacheCon Europe 2009 in Amsterdam. Special thanks to Professor Oscar Bonilla, who introduced me to open source and has always been an enthusiastic supporter of my work and studies.

Finally, I wish to express appreciation to my classmates and friends, a diverse and multi-cultural group of talented and highly motivated individuals. I am delighted to have made so many friends during these two years. Thanks to Robin Benjamins and Hadi Asghari for their friendship and for their help with my research. I also want to thank Gustavo Mercado and Napoleon Cornejo, friends with whom I shared many great times. Thanks also to Isaia, whose sweetness brightened my days and helped me to focus on finishing my thesis. I also wish to thank my family for their unconditional support and love.

Oscar Castañeda
Delft, the Netherlands
November 29, 2010
Executive Summary

“Established in 1999, the all-volunteer ASF [Apache Software Foundation] oversees nearly one hundred fifty leading Open Source projects, including Apache HTTP Server — the world’s most popular Web server software, powering more than 130 Million Websites worldwide. Today, more than 300 individual Members and 2,300 Committers successfully collaborate to develop freely available enterprise-grade software, benefiting millions of users worldwide through thousands of software solutions distributed under the Apache License. The community actively participates in ASF mailing lists, mentoring initiatives, and ApacheCon, the Foundation’s official user conference, trainings, and expo. The ASF is funded by individual donations and corporate sponsors that include AMD, Basis Technology, Facebook, Google, HP, Microsoft, Progress Software, VMware, and Yahoo!. Additional sponsors include Matt Mullenweg, AirPlus International, BlueNog, Intuit, Joost, and Two Sigma Investments.”

-- Sally Khudairi <sk@apache.org>

Apache communities are groups of individuals that organize their code production efforts in order to develop enterprise-grade open source software. The Apache Software Foundation explains the success of its communities and the software they produce by claiming to have instituted a meritocracy that brings contributors together in a way that significantly influences code production, namely by building communities instead of only focusing on technical properties of the source code like modularity.

We find that meritocracy does not influence code production. Instead, our findings suggest that Apache open source communities are self-organizing systems that emerge from modularization in code production. Our results suggest that two extreme forms of modularization lead to the emergence of communities with a highly developed division of labor. These forms are parallel lines of development and interdependent modularization, and we find that both forms encourage the formation of community clusters that emerge through reputation-based tag-following behavior.
Contents

Acknowledgements for Scholarships and Stipends v
Preface and Acknowledgements vii
Executive Summary ix
Contents x
List of Figures xiv
List of Tables xvii

1 Introduction 1
1.1 Open source communities in the ASF . . . . . . . . . . . . . . . . 1
1.2 Organization in open source communities . . . . . . . . . . . . . 2
1.2.1 The role of institutions . . . . . . . . . . . . . . . . . . . . . 3
1.3 Organization in Apache open source communities . . . . . . . . . 4
1.3.1 The role of meritocracy . . . . . . . . . . . . . . . . . . . . . 5
1.4 Research gap . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.4.1 Research question . . . . . . . . . . . . . . . . . . . . . . . 8
Research sub-questions . . . . . . . . . . . . . . . . . . . . . . . . 9
1.4.2 Research questions remain unanswered . . . . . . . . . . . 10
1.5 Conceived model of code production . . . . . . . . . . . . . . . . 10
1.5.1 Distinguishing factors in the model . . . . . . . . . . . . . . 11
1.6 Relevance of this research . . . . . . . . . . . . . . . . . . . . . . 13
1.6.1 Scientific relevance . . . . . . . . . . . . . . . . . . . . . . 13
1.6.2 Management relevance . . . . . . . . . . . . . . . . . . . . . 13
1.7 Structure of the report . . . . . . . . . . . . . . . . . . . . . . . . 14

2 Theory 15
2.1 Conway’s Law . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.2 Self-organization . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.2.1 Decentralized coordination . . . . . . . . . . . . . . . . . . . 18
2.2.2 Bottom-up organization . . . . . . . . . . . . . . . . . . . . . 19
2.2.3 Irrelevance of hierarchy ........................................... 22
2.2.4 Modular division of labor ....................................... 24
2.3 Institutionalization .................................................... 27
  2.3.1 Centralized coordination .................................... 28
  2.3.2 Top-down organization ...................................... 29
  2.3.3 Relevance of hierarchy ..................................... 31
  2.3.4 Hierarchical division of labor ............................ 35
2.4 Reconciling self-organization and institutionalization ............ 38
  2.4.1 Clustering and Hierarchy .................................. 39
2.5 Theoretical model of code production ............................ 39

3 Model........................................................................ 41
  3.1 Factors of production ........................................... 41
  3.2 Factors of organization ....................................... 42
    3.2.1 Self-organization and code production .................. 43
    3.2.2 Institutionalization and code production ............... 43
  3.3 Operationalized model of code production ..................... 46
  3.4 Factors exogenous to the model .............................. 46
    3.4.1 Extent of relevance ...................................... 47
    3.4.2 Limitations related to exogenous factors .............. 48

4 Method ..................................................................... 51
  4.1 Introduction to ASF communities .............................. 51
  4.2 Research strategy ............................................... 53
    4.2.1 Testing for the influence of design principles ........ 54
  4.3 Research methodology ......................................... 56
    4.3.1 Data mining .............................................. 57
    4.3.2 Social network analysis ................................. 61
  4.4 Measurement of conceptual constructs ......................... 62
    4.4.1 Reliability and Validity ................................ 62
    4.4.2 Dependent variables .................................. 64
    4.4.3 Independent variables ................................. 65

5 Data Analysis ................................................................ 69
  5.1 Procedure ........................................................ 69
  5.2 Data Examination ................................................ 70
    5.2.1 Factors of production: hypotheses 1 and 2 .......... 71
    5.2.2 Factors of self-organization: hypothesis 3 ............ 74
    5.2.3 Factors of institutionalized organization: hypotheses 4, 5, 6 76
  5.3 Data Transformation ............................................. 81
  5.4 Multiple-Regression Analysis .................................. 86
    5.4.1 First model: explaining lines of code .................. 87
    5.4.2 Second model: explaining revision count .............. 88
    5.4.3 Alternative model specifications ....................... 89

6 Conclusions ................................................................ 91
  6.1 Research model ................................................. 92
6.1.1 Opposing streams of research .................................. 93
6.1.2 A model of code production .................................... 93

6.2 Main findings ....................................................... 94
6.2.1 Meritocracy does not influence code production ............. 94
Hierarchy does not influence code production ...................... 95
Modularity does influence code production ......................... 95
Answer to the research question .................................... 95
Open source communities are self-organizing ....................... 96

6.3 Examination of design principles ................................ 96
6.3.1 Conflict resolution mechanisms ............................... 96
6.3.2 Collective choice arrangements ............................... 98
6.3.3 Multiple layers of nested enterprise .......................... 99
6.3.4 Comparison to results from previous research ............... 100

6.4 Discussion of results ............................................. 101
6.4.1 Alternative explanations ...................................... 101
6.4.2 Limitations of this research .................................. 102
6.4.3 Generalizability of results .................................... 102
Generalizability of methods ....................................... 103

6.5 Implications and Reflection ...................................... 103
6.5.1 Implications .................................................... 103
6.5.2 Reflection ........................................................ 104

6.6 Future research .................................................... 105
6.6.1 Future research .................................................. 105
6.6.2 Future research on methods and techniques .................. 107

Bibliography 113

A Additional Figures and Tables 121
A.1 Tables for Section 5.2 ............................................. 121
A.2 Tables for Section 5.3 ............................................. 128
A.3 Tables for Section 5.4 ............................................. 131
A.3.1 First (untransformed) model: explaining lines of code .... 134
A.3.2 Second (untransformed) model: explaining revision count 134
A.3.3 First (discarded) extended model: explaining lines of code per author count (productivity) .... 135
A.3.4 Second (discarded) extended model: explaining revision count per author count (productivity) .... 135

B Contributions to SVNPlot 137

C Research Proposals 143
C.1 Google Summer of Code 2009 .................................... 143
C.2 Google Summer of Code 2010 .................................... 143

D Posters and Presentations 155
D.1 Poster presented at AlBan Conference .......................... 155
D.2 Presentation at ApacheCon NA 2010 ............................ 155
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.2.1 Abstract</td>
<td>155</td>
</tr>
<tr>
<td>D.2.2 Presentation Slides</td>
<td>155</td>
</tr>
<tr>
<td>D.2.3 Blog Post</td>
<td>155</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 Conceived model of code production ........................................... 11
2.1 Apache Web of Trust ................................................................. 26
2.2 Theoretical model of code production ........................................... 40
3.1 Conceptualized model of code production .................................... 47
4.1 Apache HTTP Server community in 2009 ....................................... 56
4.2 Apache Hadoop community and sub-communities in 2008 ................. 57
4.3 Apache Hadoop community and sub-communities in 2009 ................. 58
5.1 Histograms of (a) Lines of Code and (b) Revision Count, both from Pooled data (2004-2009) ......................................................... 71
5.2 Histogram of File Count, from Pooled data (2004-2009) .................. 71
5.3 Scatterplots for (a) File Count vs. Lines of Code and (b) File Count vs. Revision Count, both for Pooled data (2004-2009) .............. 72
5.4 Histogram of Author Count, from Pooled data (2004-2009) ............. 72
5.5 Scatterplots for (a) Author Count vs. Revision Count and (b) Author Count vs. Revision Count, both for Pooled data (2004-2009) ....... 73
5.6 Histogram of Clustering Coefficient (Watts and Strogatz, 1998), for Pooled data (2004-2009) .............................................................. 74
5.7 Scatterplots for (a) Clustering Coefficient (Watts and Strogatz, 1998) vs. Lines of Code, and (b) Clustering Coefficient (Watts and Strogatz, 1998) vs. Revision Count, both for Pooled data (2004-2009) ....... 75
5.8 Histogram of Graph Connectedness (Krackhardt, 1994), for Pooled data (2004-2009) .............................................................. 77
5.9 Scatterplots for (a) Graph Connectedness (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Connectedness (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2009) ....... 78
5.10 Histogram of Graph Hierarchy (Krackhardt, 1994), for Pooled data (2004-2009) .............................................................. 78
5.11 Scatterplots for (a) Graph Hierarchy (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Hierarchy (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2009) ....... 79
5.12 Histogram of Graph Efficiency (Krackhardt, 1994), for Pooled data (2004-2009) .............................................................. 79
5.13 Graph Efficiency (Krackhardt, 1994) vs. Lines of Code, and Graph Efficiency (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2009)  
5.14 Histogram of Graph Hierarchy (Krackhardt, 1994), with and without outliers, for Pooled data (2004-2009)  
6.1 Theoretical model of code production  
6.2 Apache Hadoop community and sub-communities in 2008  
6.3 Apache HTTP Server community in 2009  
6.4 Apache Hadoop community and sub-communities in 2009  
A.1 Scatterplots for (a) File Count vs. Lines of Code and (b) File Count vs. Revision Count, both for Pooled data (2004-2009)  
A.2 Scatterplots for (a) Author Count vs. Lines of Code and (b) Author Count vs. Revision Count, both for Pooled data (2004-2009)  
A.3 Scatterplots for (a) Clustering Coefficient (Watts and Strogatz, 1998) vs. Lines of Code, and (b) Clustering Coefficient (Watts and Strogatz, 1998) vs. Revision Count, both for Pooled data (2004-2009)  
A.4 Scatterplots for (a) Graph Connectedness (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Connectedness (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2009)  
A.5 Scatterplots for (a) Graph Hierarchy (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Hierarchy (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2009)  
A.6 Scatterplots for (a) Graph Efficiency (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Efficiency (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2009)  
A.7 Stata output of gladder loc command  
A.8 Stata output of gladder rev command  
A.9 Stata output of gladder filecount command  
A.10 Stata output of gladder authorcount command  
A.11 Stata output of gladder hierarchy command  
A.12 Stata output of gladder connectedness command  
A.13 Graph matrix showing all independent variables.  
B.1 Code snippet (1) svnsqlite2ora.py  
B.2 Code snippet (2) svnsqlite2ora.py  
B.3 Code snippet (3) svnsqlite2ora.py  
B.4 Code snippet (4) svnsqlite2ora.py  
C.1 Google Summer of Code 2009 research proposal (pgs. 1,2/9)  
C.2 Google Summer of Code 2009 research proposal (pgs. 3,4/9)  
C.3 Google Summer of Code 2009 research proposal (pgs. 5,6/9)  
C.4 Google Summer of Code 2009 research proposal (pgs. 7,8/9)  
C.5 Google Summer of Code 2009 research proposal (pg. 9/9)  
C.6 Google Summer of Code 2009 research proposal (pgs. 1,2/10)  
C.7 Google Summer of Code 2009 research proposal (pgs. 3,4/10)  
C.8 Google Summer of Code 2009 research proposal (pgs. 5,6/10)  
C.9 Google Summer of Code 2009 research proposal (pgs. 7,8/10)  

xv
C.10 Google Summer of Code 2009 research proposal (pgs. 9,10/10) . . . . . . . . . 153
D.1 Poster presented at AlBan Conference . . . . . . . . . . . . . . . . . . . . . 156
D.2 Presentation abstract ApacheCon NA 2010 . . . . . . . . . . . . . . . . . . . . 157
D.3 Slides Presented at ApacheCon NA 2010 (pg. 1/5) . . . . . . . . . . . . . 157
D.4 Slides Presented at ApacheCon NA 2010 (pg. 2/5) . . . . . . . . . . . . . 158
D.5 Slides Presented at ApacheCon NA 2010 (pg. 3/5) . . . . . . . . . . . . . 158
D.6 Slides Presented at ApacheCon NA 2010 (pg. 4/5) . . . . . . . . . . . . . 159
D.7 Slides Presented at ApacheCon NA 2010 (pg. 5/5) . . . . . . . . . . . . . 159
D.8 Blog Post in Google's Open Source Blog . . . . . . . . . . . . . . . . . . . . . 160
List of Tables

3.1 Software Quality Metrics reviewed in Nagappan et al. (2008) ............... 47
3.2 Organizational Complexity Metrics proposed in Nagappan et al. (2008) .... 48

4.1 ASF Top-level Project Categories .................................................. 53
4.2 ASF Top-level Communities ........................................................... 54
4.3 Code Production in ASF Top-level Communities in 2009 ...................... 55

5.1 Variables selected for data transformation ......................................... 81
5.2 Box plot of graph hierarchy, graph connectedness, graph efficiency and clustering coefficient .......................................................... 84
5.3 Box plot of graph hierarchy, graph connectedness, graph efficiency and clustering coefficient after data transformation ......................... 84
5.4 Summary of Preliminary Findings and Statistical Tests Used ................. 85
5.5 Regression Model with log(Lines of Code) (log) as the dependent variable. 87
5.6 Regression Model with sqrt(Revision Count) (sqrt) as the dependent variable ................................................................. 88
5.7 Summary table of the outcome of multiple-regression analysis .............. 89

A.1 Correlation Table showing all independent variables .......................... 132
A.2 Regression Model with Lines of Code (loc) as the dependent variable .... 134
A.3 Regression Model with Revision Count (rev) as the dependent variable. 134
A.4 Regression Model with Log(Lines of Code per author) (log_per_author_log) as the dependent variable ............................................. 135
A.5 Regression Model with Sqrt(Revision count per author) (sqrt) as the dependent variable .......................................................... 135
Chapter 1

Introduction

Open source communities in The Apache Software Foundation (ASF) are groups of individuals with a common cause, who share leadership roles and together contribute in code production, which is the model of distributed collaboration used in the evolution of a software codebase (Weber, 2004) (pg. 94). Some contributors volunteer time and efforts while others are paid to develop software. They seldom meet but often interact by co-authoring source code files through shared infrastructure on the Internet, mainly revision control systems like Subversion.\textsuperscript{1} Despite problems of coordination associated with geographically distributed software development (Herbsleb and Grinter, 1999a) contributors in Apache are able to organize into communities that develop enterprise-grade software.

1.1 Open source communities in the ASF

Open source software is collectively developed by a community of participants who are distributed and collaborate over the Internet. Code production in these communities is the model of distributed collaboration used in the evolution of a project’s underlying software codebase (Weber, 2004) (pg. 94). Evolution happens through interactions in the codebase of an open source project, which give rise to a community. Such a community is organized through the production of software whose existence (or ‘being’) is represented by an open source project (Rideau, 2010). The resulting source code of a project is open, which means that it can be modified and freely redistributed, while its community exists and continues to evolve along with the software around which it formed.

One of the best examples of open source is the software produced in the ASF, a non-profit corporation created to provide an umbrella for projects of the Apache communities. The institutional infrastructure of the ASF has enabled Apache communities to produce software that is comparable and in many cases superior in market share\textsuperscript{2} and quality\textsuperscript{3} to that of closed source software systems,\textsuperscript{4} hence it is referred to as enterprise-grade software.\textsuperscript{5}

The highly successful software produced by Apache open source communities is used by corporations, universities, research organizations, non-profits, and individuals\textsuperscript{6} around the world. Many companies have joined Apache and
the open source revolution to sponsor, participate and benefit from the code it produces (Vixie, 1999). Some of these companies include Google, Yahoo, Facebook, Microsoft, Amazon and IBM. All contributions are owned by the ASF, mainly to safeguard individual participants from future legal claims (Van Wendel de Joode, 2005). The Apache community and the ASF are exemplar in open source organization and have been the subject of extensive academic research and related publications (Mockus et al. (2000); Mockus et al. (2002); Van Wendel de Joode (2005); Van Wendel de Joode et al. (2003); O’Mahony (2003a); O’Mahony (2003b); González-Barahona et al. (2004); Fogel (2005); Weber (2004); Weiss et al. (2006)).

The ASF explains the success of its communities and the software they produce by claiming to have instituted a *meritocracy* that brings contributors together into communities in a way that significantly influences code production, namely by building communities instead of only focusing on technical properties of the source code like modularity. The claim is that ‘community-over-code’ distinguishes the Apache communities from other open source projects like those found in Sourceforge or Google code. Projects on these developer sites often do not have a community around them that ensures the evolution of a software codebase. Consequently, many of these projects cannot ensure sustainable and organized code production. Instead, contributors in these projects only focus on the modularization of their codebase. Another negative consequence is that the lack of a community hinders firms’ (social) participation and adoption of the software these open source projects produce (O’Mahony and West, 2008) (pg. 13); (Stam and van Wendel de Joode, 2007) (pg. 509)). For instance it forces firms to take up a management role in open source projects in order to compensate for the lack of a community (Capra et al., 2009).

The relevance of claims regarding the institutional success of the *meritocracy* governance model (Erenkrantz and Taylor, 2003; O’Mahony and Ferraro, 2007) can be explained and better understood by analyzing such claims through an organizational model of open source. Understanding how open source communities are organized and how they sustain themselves can also explain whether the *meritocracy* model has an influence on community building and code production, as explained previously, or if such a model has been created primarily based on incentives (Van Wendel de Joode, 2005). For instance, one incentive to create such models is to give an explanation to poorly understood phenomena.

### 1.2 Organization in open source communities

Recent organizational research (Van Wendel de Joode, 2005), found that open source communities are primarily self-organizing. This means that internal processes of interaction in the communities, which are not guided or managed from an outside source, give rise to collective patterns of behavior. Van Wendel de Joode (2005) points out that the self-organizing nature of open source communities limits their malleability. Such limitation is mainly because of the ‘light-handed’ influence of institutions, which are defined as the *rules that un-
The extent of influence of institutions is observable through the behavior of software developers. This in turn has implications for the purposeful design and creation of open source communities around institutions. Van Wendel de Joode (2005) adopted a framework from research on community-managed common pool resources, which explains how more traditional communities, such as huertas and communities sharing water resources, have managed to self-organize around institutions. Based on this framework he created an organizational model of open source “(...) to serve as input to discussion and reflection and as a guide for further research on the subject” (Van Wendel de Joode, 2005) (pg. 37). The model consists of eight design principles that reconcile the two main strands of research on the organization of open source communities. Such strands correspond to two extreme views on organization, namely self-organized anarchies and institutionalized communities (Van Wendel de Joode, 2005). These views contribute opposing ways to explain organization:

1. A bottom-up explanation that focuses on mechanisms that explain self-organization.
2. A top-down explanation centered on institutions that exert influence on the communities.

Van Wendel de Joode (2005) combines these explanations to understand how bottom-up behavior in open source arises in the presence of top-down influence. One of his main findings is that “(...) the role of institutions is minor compared to individual behavior (...)” (Van Wendel de Joode, 2005) (pg. 207). Instead of an explanation based on institutions, he explains self-organization through emergent patterns that are the result of individualistic behavior of participants in the communities. Van Wendel de Joode (2005) explains emergence through a small number of individual behavioral rules that result in collective behavior.

Collective behavior is different than the individualistic form from which it emerges, and results in innovation in the software produced in open source communities. Combined with mechanisms related to the design principles mentioned previously, individual behavioral rules are claimed to be sufficient to understand how open source communities are organized and how they sustain themselves. Therefore, Van Wendel de Joode (2005) finds that a bottom-up explanation is sufficient to understand organization in open source.

1.2.1 The role of institutions

Institutions can be informal, in which case they are typically emergent and organic ((Hodgson, 2003), cited in Van Wendel de Joode (2005)). Such institutions include norms and behavioral codes of conduct ((Jong, 1999) cited in Van Wendel de Joode (2005)). On the other hand, institutions can also be formal, in which case they consist of rules such as procedures, arrangements, contracts and laws (Van Wendel de Joode, 2005). It is also possible, although not mentioned by Van Wendel de Joode (2005), to have the presence of hybrid institutions that combine informal and formal aspects:
“Institutions, whether conceived as groups or practices, may be partly engineered, but they have also a “natural” dimension.” (Selznick, 1984)

Van Wendel de Joode (2005) concentrates on formal institutions. He gives three explanations of incentives to establish such institutions, despite their minor role in self-organization. First, institutions serve as a simple reference for individualistic participants to understand how complex collective mechanisms, like governance and decision-making, are carried through. Second, institutions are a means for open source communities to gain external recognition and counter common criticism for their chaotic nature. And third, institutions are employed to explain the success of open source communities and the software they produce (as with the ASF). However, beyond the explanation of incentives to create institutions, Van Wendel de Joode (2005) specifically mentions the ASF as an exception.

“Some institutions do have an important role in the communities. The role of the Apache Software Foundation in the Apache community, for instance, is not at all marginal. The ASF performs the important function of safeguarding participants in the community from future legal claims.” (Van Wendel de Joode, 2005)

In addition, Van Wendel de Joode (2005) puts forth a series of propositions, of which three relate to the role of institutions in open source. The first proposition states that intelligence at the organizational level is marginally attributable to collective institutions. Instead, the second proposition affirms, such intelligence emerges from individual behavior. The third related proposition is apparently connected to the aforementioned exception as it focuses on institutions, such as the ASF, that do play a significant role in the organization of open source communities. The proposition reads as follows:

“The institutions in open source communities that do play a significant role do so because they (i) increase the external recognition of communities and (ii) are a means to protect the communities from outside pressures.” (Van Wendel de Joode, 2005) (pg. 214)

1.3 Organization in Apache open source communities

The Apache communities are organized under a software foundation, ‘The ASF’, which was created through negotiation of rights and interests between Apache communities and corporate business firms. The result was a set of social arrangements that were new to both communities and firms (O’Mahony, 2003a). One of those collective arrangements is a meritocratic governance model ((Erenkrantz and Taylor, 2003); (O’Mahony and Ferraro, 2007)) for open source
software development that has facilitated the organization of Apache communities under the ASF. The resulting structure of projects comprise a loosely-organized group of communities that are referred to as the ‘central decision-making organizations of the Apache world.’ These communities share an institutional infrastructure that is embodied in the principle of meritocracy, an emergent and hence informal institution in the ASF which states that:

‘The more you do, the more you are allowed to do.’

1.3.1 The role of meritocracy
The principle of meritocracy can be defined as a rule that influences behavior in code production. The principle of meritocracy has been extended to an institutionalized model of organization in open source which is defined as the decentralization of leadership roles that are distributed among members of a community (Erenkrantz and Taylor, 2003). Distributed leadership has been found to include both technical and organization-building contributions (O’Mahony and Ferraro, 2007) (pg. 1100). Therefore, as an institution, meritocracy does not appear to be ‘light-handed.’ On the contrary, it appears that meritocracy encourages increases in quantity and quality of technical contributions. Furthermore, meritocracy plays an organizational role in redistributing power among contributing members of the community in a way that is accepted by such members. Therefore, meritocracy combines both technical and organization-building aspects of contribution in open source. It appears that meritocracy is the keystone that holds together encouraged technical behavior and the organizational power redistribution that results from such behavior. Thus the presence of meritocracy leads to the belief that Apache communities are not completely self-organized. This implies that Apache communities can to some extent be purposefully designed in accordance to The Apache Way, which is the set of guiding principles of the ASF that are essentially based on meritocracy.

One way to understand The Apache Way is to study the behavior of Apache software developers. The ASF claims that what these developers do says much about how the Apache communities are organized (Fielding, 1999). Developers come together into a community by doing. Primarily this means co-authoring source code, which is a medium of expression that gives developers “(...) equal opportunity to succeed or fail based on their merits.” (Hecker, 2010). Leadership roles emerge from developers’ reputation, which is based on how good they are as coders (Weber, 2004) (pg. 180). Therefore, to understand The Apache Way we first need to understand meritocracy. And to understand meritocracy we need to study behavior in code production.

The role of meritocracy is believed to be that of recognizing coding patterns based on a context-specific understanding by the community of how merit is conceptualized (O’Mahony and Ferraro, 2007). Recognized patterns are interpreted into a reputation system which grants allowances in terms of power and control. In other words, meritocracy is initially an informal institution that over time leads to a formal one, where informal patterns of social behavior emerge to become the official (formal) way of doing things. This happens until the
patterns change and a new official (formal) way emerges. An example of this process can be found in the credit that individual contributors receive in an open source project’s website. Furthermore, these same resources often distinguish contributors who in addition of being committers are also recognized as PMC members. This latter recognition distinguishes such individuals as members who also play an organizational role in the communities. Such recognition and roles are direct results of meritocracy. This clarifies how meritocracy is initially emergent but is then enforced top-down.

The role of meritocracy is then to recognize patterns and reinterpret them as the new official way of code production. In this process a hierarchy of decision-making emerges which is continuously reconceptualized into a structure of roles that has been described as pyramidal flow in decision-making (Weber, 2004) (pg. 187). Such pyramidal flow resembles the hierarchy wheel described in de Bruijn and ten Heuvelhof (2008) (pg. 14).

The hierarchy of roles created through meritocracy includes: user, developer, committer, PMC member, PMC chair, and ASF member. These roles are chosen by self-selecting individuals that gain power by sustained contributions over time (Erenkrantz and Taylor, 2003). The resulting roles in Apache have been found to have a direct influence on code production (Mockus et al. (2000); Mockus et al. (2002)). Furthermore, such roles are structured as a voluntary hierarchy (Weber, 2004), that functions as a decision-making system for the management and coordination of releases (Fogel, 2005).

Therefore, meritocracy seems to play a significantly influential role in the organization of Apache open source communities. It appears to be an informal institution that displays convergence in behavior, which ultimately results in the formation of a social hierarchy (Valverde et al., 2006). However, it is not clear whether meritocracy influences code production especially when compared to the natural presence of competing explanations like modularity.

1.4 Research gap

Van Wendel de Joode (2005) seems to only attribute an external role to formal institutions in the self-organization of open source communities. The absence of an internal function indicates that the role of institutions in organization is minor. Instead, Van Wendel de Joode (2005) finds that individual behavior plays a more substantial role. For instance, he proposes that intelligence at the organizational level emerges from individual behavior (Van Wendel de Joode, 2005) (pg. 214).

Consequently, Van Wendel de Joode (2005) takes for granted the possibility that behavior emerges into an informal institution as a result of previous behavior. This implies that he only considers emergent patterns of behavior that are not shaped in any way by previous efforts. In other words, it is possible that the behavior that precedes emergence, might have been shaped by previous behavior that in turn had emerged into an informal institution. An example would be a pattern of code production that was repeated over time and became a mold for behavior that emerged subsequently. Another way to say this is
Introduction

1.4 Research gap

that informal institutions are a mold upon which behavior emerges and leads to further repetitions of similar behavior. Following this hypothetical line of reasoning, what would happen if emergent patterns of behavior did converge on informal institutions? One answer is that informal institutions could then help in understanding how open source communities are organized.

As mentioned previously, informal institutions are typically emergent. They include norms and behavioral codes of conduct, both of which are patterns of social behavior (Axelrod, 2006). Such patterns are described by researchers through rules, which differ based on the object of analysis (Van Wendel de Joode, 2005) (pg. 26). For some researchers the object of analysis describes actual behavior while for others it describes individual behavior. However, regardless of whether patterns of social behavior describe actual behavior or just individual behavior, the informal institutions to which such behavior patterns are bound still describe behavior that has arisen in the absence of an external ordering influence.

This opens the possibility for a principal role for informal institutions. Such role would be primarily organizational, therefore internal, and thus beyond the external role ascribed to formal institutions. Furthermore, the possibility of a significant organizational role of informal institutions has not been addressed in literature on self-organization and institutionalization in open source, including in Van Wendel de Joode (2005). Identifying the absence of such an explanation broaches a research gap which can be bridged by investigating the extent of influence of informal institutions on code production.

Code production is believed to be relevant because it is directly connected to sustainability and is one of the principal aims of organization in open source, namely to produce open source software in a sustainable way despite internal and external pressures. Furthermore, code production is relevant in non-profit foundations that mediate pressures between open source communities and firms (O’Mahony, 2003a). Therefore, the research gap identified here is that in open source communities with an associated non-profit foundation, such as Apache communities with the ASF, informal institutions can plausibly explain how open source communities are organized and how they sustain themselves, as can be observed by code production.

Code production is defined in this research as the model of distributed collaboration used in the evolution of a project’s underlying software codebase (Weber, 2004) (pg. 94). This model of distributed collaboration can be based on the modularization of source code or on the institutionalization of rules such as meritocracy. In addition, distributed collaboration can also come as a result of non-organized behavior and simply arise from events that have no relation to how an open source community is organized. Therefore, we need to further specify code production as a measurable aspect that can allow us to distinguish between behavior that comes as a result of modularization, institutionalization or which is simply just not organized in a particular way.

The more refined notion of code production is obtainable by focusing on two measures that are found to be at different levels of granularity: the number of lines of code and the number of revisions (revision count) that are produced in an open source project by the members of its community (therefore, by the
committers of the project who are those allowed to make changes to source files on the source code repository). In both measures code production happens as a result of co-authoring source code files. Therefore, these properties are observable at the individual file level but can also be used for reflection at the project level (with a project understood as a collection of files). In addition, these aspects of code production give us access to a social graph of collaboration in the production of open source software.

How we propose to study code production and organization in open source (as will be explained in more detail in Chapter s 3 and 4) also puts forth a research gap (which we address in this research). We claim that the organization of open source communities has not been studied with a focus on code production, which we identify as the primary activity of open source communities. Other research has focused on analyzing mailing list traffic, of which we are critical because ‘sending emails’ is not what open source developers do. What they do is ‘write code’ and therefore it seems more appropriate to study code production rather than other aspects.

1.4.1 Research question

Emergence and the role of institutions are at the center of competing explanations of organization in open source: self-organized anarchies versus institutionalized communities (Van Wendel de Joode, 2005) (pg. 12). The core issue between these competing explanations is how they approach the problem. Self-organization relies on synthesis – it is a bottom-up explanation of organization. Institutionalization, on the other hand, employs analysis and uses a top-down approach to understand structure.

According to Van Wendel de Joode (2005), and literature on self-organization, the role of institutions is minor. On the other hand, the ASF claims that the meritocracy institution explains success in code production and community building. This comes in agreement to the views expressed in literature on institutionalization in open source, namely that institutions are central in the explanation of how open source communities are organized. The core issue behind competing bottom-up and top-down explanations represents our main research question:

Does meritocracy influence code production?

Encouraging bottom-up growth and grooming contributions top-down seem to be part of meritocracy, as can be intuitively perceived from its motto – ‘the more you do, the more you are allowed to do.’ However, neither self-organization nor institutionalization literature explain the emergence and role of meritocracy in code production. Therefore we will use the organizational model of open source of Van Wendel de Joode (2005) since it combines both explanations – it examines institutions around which communities have been known to self-organize. Furthermore, such model provides “(...) a way to decide where to look for mechanisms to understand how the communities are organized” (Van Wendel de Joode, 2005) (pg. 38).
We have selected three design principles from the organizational model of open source (Van Wendel de Joode, 2005) because of their relevance on decision-making and governance. Namely these principles are collective choice arrangements, conflict resolution mechanisms and multiple layers of nested enterprise. These design principles are valid for this research because they represent aspects of hierarchy, a structured ordering in a network that will be defined shortly, and these are aspects that can be readily operationalized and measured. Furthermore, the design principles are relevant for this research because they provide “(...) a way to decide where to look for mechanisms to understand how the communities are organized” (Van Wendel de Joode, 2005) (pg. 38).

We believe the measures that will be used (as will be explained in Chapter s 3 and 4) capture extreme forms of the organizational instruments mentioned previously, therefore giving us an understanding of the extent to which these instruments have an influence on community organization (as observed by influence on code production). Therefore, we have equated meritocracy with hierarchy in a network, which can be further decomposed into connectedness, asymmetry and redundancy. These are aspects of hierarchy in a network that can be measured and based on which we can reflect on the organization of open source communities.

**Research sub-questions**

As will be explained in Chapter s 2 and 3, the intersection of the three design principles, from the organizational model of open source (Van Wendel de Joode, 2005), that are studied in this research seems to lie in structural mechanisms that can be observed in measures of hierarchy. Our research sub-questions therefore refer to the extent to which such phenomena influence code production. Different predictions are found in literature on self-organization and institutionalization with regards to such influence. We will examine these predictions in Chapter 2.

The research sub-questions focus on two types of emergence of organization in open source communities. The first is a consequence of modularity which is called clustering. It is a form of emergence that leads primarily to self-organization and stands for bottom-up mechanisms. The second form of emergence is associated with hierarchy. This form is believed to lead to the institutionalization of emergent patterns of social behavior into decision-making roles structured as a ‘pyramidal flow hierarchy’ (Weber, 2004). In other words, the belief is that social patterns of behavior converge and become the official way of doing things. Hence the expectation is that this form of emergence leads shortly to top-down institutional instruments, and thus emergent hierarchy represents the opposite of bottom-up organization. The resulting structure leads to the following sub-questions of this research:

1. **Does modularity influence code production?**

2. **Does hierarchy influence code production?**
The first question is not related to meritocracy but offers an alternative to it, namely through modularity which is a phenomenon in code production that arises organically (and results in clustering as will be explained in more detail in Chapter 2). On the other hand, the second sub-question is related to meritocracy in several ways. As will be explained in more detail in Chapter 2, hierarchy is a conceptual construct that is sub-divided into three aspects that focus on the asymmetry, level and redundancy of connections in a graph. These aspects are believed to represent extreme forms of the collective institutions that were selected for this research, namely collective choice arrangements, conflict resolution mechanisms and multiple layers of nested enterprise. These institutions are believed to be relevant to meritocracy because they allow us to make this vaguely defined concept more concrete by relating it to mechanisms that have been established to have an influence in other communities and which have been a central part of research into the organization of open source communities (Van Wendel de Joode, 2005).

1.4.2 Research questions remain unanswered

Success in code production also depends on the appropriate management of internal pressures such as the motivation of individuals to contribute, the threat of incompatibility and the conflicts that arise in the production of open source software (Van Wendel de Joode (2005) (pg. 2); Fogel (2005)). The previously mentioned exception and proposition only provide answers that relate to outside pressures and therefore leave our research questions unanswered.

1.5 Conceived model of code production

Based on the research gap identified in this Chapter we have conceived a first model of code production. Figure 1.1 shows this model, with hierarchy as a top-down instrument that exerts influence on code production and clustering (which is the result of modularity) exercising a bottom-up influence. Both of these are factors of organization, as will be explained in Chapter 3, that are believed to positively influence code production. More concretely our belief is that a combination of bottom-up mechanisms and top-down instruments, which we also refer to in engineering terms as analysis and synthesis, will have a positive consolidated influence on code production.

Respectively we will refer to the bottom-up and top-down phenomena as factors of self-organization and factors of institutionalized organization. Additionally, Figure 1.1 shows the number of files and the number of authors also positively influencing code production. These are factors of production that are assumed to play a role in all open source communities as will be explained in Chapter 2. Furthermore, these factors of production will be used to represent the lack of organization in the model constructed in Chapter 3.

The interactions shown in Figure 1.1 are suggestive of the expectations we have articulated previously. However, some clarification is useful in an attempt to make these relationships more intuitive, and specifically will help to discard the possibility of circular reasoning. As was described in section 1.4, we specified
Conceived model of code production

1.5.1 Distinguishing factors in the model

As will be explained in Chapter 2, clusters of developers arise as a result of splitting up complex software into different activities (Van Wendel de Joode, 2005) (pg. 134). We consider clustering as a phenomenon that is brought about...
by modularity. Modularization may have other effects as well. For instance, it can increase the number of files as a result of splitting up source code into modules. However, the effect of splitting up code will not be noticeably different in terms of the outputs, namely lines of code and revision counts, since splitting up code is meant to encourage participation but does not necessarily entail increased participation.

On the other hand, the result of modularization which leads to the formation of clusters of authorship (Ghosh, 2004) (pg. 18) (as will be explained in more detail in Chapter 2), does entail increased participation and therefore leads to increases in code production measured as lines of code and revision count. The reason why this happens will be explained in Chapter 2, but we include a summary here. Modularization leads to the formation of clusters of authorship because developers come together through the bottom-up aggregation of individual decisions, like those to co-author source code files, which has been reported to arise from artificial properties called tags (Van Wendel de Joode, 2005), which developers copy and follow.

The predictions found in literature are that increased participation results in higher code production and that increased participation leads to the formation of clusters of authorship which in turn result in higher code production. Both of these are influenced by modularization because it leads to increased participation, but not because it leads to an increase in the number of files (since by itself, that is not enough to entail increased participation). In other words, when modularization has an organizational influence, then it does lead to increases in code production, but otherwise it does not.

We claim that the distinction between modularization that leads to clustering and modularization that only leads to splitting-up of source code (e.g. as found in a specific kind of software architecture) discards the possibility of circular reasoning, since we are making a clear distinction between factors that are organized and those that are not. Furthermore, we claim that the former influences code production, while the latter by itself is not enough to have this level of influence. We argue that the influence of factors of production on code production does not reflect the same reasoning as that applied when we consider the influence of factors of organization on code production. Therefore, separating the factors of production from the factors of organization reduces the risk of biasing the results of this research.

Moreover, our reasoning aims to separate factors of production from factors of organization, as much and as cleanly as possible, in order to enable a comparison between organizational architectures (institutionalized vs. self-organized architectures). We acknowledge that this separation is not entirely possible because the constructs we use to reflect on organization are built on the basis of factors of production, therefore the separation is not as clean as would be desired. Nevertheless, as will be explained in Chapter 4, our method enables us to compare measures on the same scale and to interpret the results as the percentage of presence of a particular organizational measure. And by also taking factors of production into account, we believe to have substantially reduced (or eliminated) the risk of taking those factors for granted.
1.6 Relevance of this research

1.6.1 Scientific relevance

This research is scientifically relevant because it constitutes an empirical test of the organizational framework of Van Wendel de Joode (2005). Furthermore, adding to its relevance, this research is an empirical investigation into the influence of meritocracy on code production in the ASF. Explanations from theory and practice appear to be opposites and through its empirical approach this research aims at making a meaningful inference taking practice as rival theory.

This research brings together management science and computer science, thus increasing its scientific relevance. Furthermore, directions for future research in this project propose to employ methods of artificial intelligence, a branch of computer science that is relevant to management science. In addition, this research addresses two directions for future research put forward in Van Wendel de Joode (2005). These directions propose to investigate in more detail:

- “(...) emergence and role of institutions in self-organizing communities (...)” (Van Wendel de Joode, 2005), and

- “(...) aspects of [...] organization [in] open source communities in relation to the software they create (...)” (Van Wendel de Joode, 2005)

1.6.2 Management relevance

This research has many applications in management of technology and innovation. The analysis of formal and informal institutions can be employed to get a better understanding of the influence business firms can have on open source communities. One relevant example would be the role of strategy in corporate venturing and its influence on the strategic renewal of software companies and open source communities participating in the production of open source software. An application is the use of open source incubators as tools for corporate venturing.

Another application of this research is in the management of volunteers in the production of open source software (Fogel, 2005). Companies looking to invest in open source and “give back to the community” might also be interested in having their contributions become sustainable. As well of interest is to provide a substantial understanding of the role their own software developers play in an open source community. In addition, open source can be used as a management tool for recruitment and this research is relevant in that respect because it provides a framework, methodology and software tools to better understand the role and contributions of volunteer software developers in open source communities.
1.7 Structure of the report

This report is structured as follows. Chapters 2 and 3 address the two most important theories concerning organization in open source, namely self-organization and institutionalization. Chapter 2 addresses theoretical aspects which are surveyed and used to construct a conceptual model. This model is presented in Chapter 3 and is constructed out of expectations drawn from theory. Then in Chapter 4 there is a description the data mining software that was written to collect, pre-process and represent open source communities as networks of file co-authorship. Furthermore, Chapter 4 discusses how other software was used for social network analysis and statistical analysis of relationships present in the collected networks of file co-authorship.

Chapter 5 provides a detailed account of the results of this research. It describes the results in relation to the design principles that were identified in this introductory Chapter. Following from results, Chapter 6 focuses on discussion and conclusions. Finally, section 6.6 proposes directions for future research. There are also various appendices included in this report. The appendices cover additional material, which includes tables referenced in the main text, two accepted project proposals based on this research, a poster presented at a student conference, and the abstract and slides presented at a practitioner conference.
Chapter 2

Theory

In this Chapter relevant literature on organization in open source will be reviewed. Its main purpose is to survey opposing explanations of organization in open source that come from the research strands of self-organization and institutionalization. Specifically the focus will be on institutions and their influence on the sustained organization of code production in open source communities. Different predictions about the extent of code production will be examined.

The outcome of this Chapter will be a theoretical framework that brings together bottom-up and top-down explanations of organization in open source. This framework will meaningfully relate modularity and hierarchy to predictions about code production. In the next Chapter hypotheses will be formulated based on the theoretical framework and a conceptual model will be constructed that combines the expectations drawn from these hypotheses. In this research Conway’s Law is assumed to hold, this is explained in Section 2.1. Much recent research in software engineering coordination, scientific computing, reverse software engineering and open source organization, shows that Conway’s Law is relevant and continues to be tested.

In the following sections we concentrate on three design principles from the organizational model of open source (Van Wendel de Joode, 2005). We split these design principles into their institutional and self-organizational components so as to draw expectations about the level of code production from predictions found in literature. The sections on bottom-up and top-down organization (sections 2.2.2 and 2.3.2) focus on the design principle of collective choice arrangements. The next sections, 2.2.3 and 2.3.3, are about the irrelevance and relevance of hierarchy, and are centered on the design principle of conflict resolution mechanisms. Following this, sections 2.2.4 and 2.3.4, respectively on modular and hierarchical division of labor, pay particular attention to the design principle of multiple layers of nested enterprise.

To clarify the argument, we have included small summaries at the start and end of every section, that can be read independently, and which give the main ideas we are trying to present. Furthermore, we will start with theory on self-organization because it will be useful to build a foundation for theory of institutionalization, which in many ways can be considered to come as a response to claims made in self-organization.
2.1 Conway’s Law

An assumption made in this research is the existence of an isomorphism between the architecture of an open source project’s codebase and the organization of its associated open source community. The rationale behind this assumption comes in relation to the claim of ‘community-over-code’ that is part of how the ASF explains the success of its communities and the software they produce (as explained in Section 1.1). Furthermore, this assumption is related to the direction of future research proposed by Van Wendel de Joode (2005), to study “(...) aspects of [...] organization [in] open source communities in relation to the software they create (...)” (Van Wendel de Joode, 2005) (as explained in Section 1.6.1).

One of the first observations connecting organization to architecture was made by Conway (1968). His observation became known as Conway’s Law, a principle which establishes that:

“Any organization that designs a system will produce a design whose structure is a copy of the organization’s communication structure.” (Conway, 1968)

Brooks (1995) was the first to cite Conway, and named the principle Conway’s Law. The “Documentary hypothesis” in Brooks (1995), is based on this law. This hypothesis relates a reduced set of documents to the success of a software project. These so-called “formal documents” become intertwined with the organizational structure of software projects, specifically in terms of roles and ownership, and are considered important because they actively register and communicate decision-making (Brooks, 1995).

“Amid a wash of paper, a small number of documents become the critical pivots around which every project’s management revolves.” (Brooks, 1995)

In addition, and as a consequence of the isomorphism from the structure of a system to the structure of its design organization, Conway (1968) identified the dynamic nature of the communication structure in organizations in relation to prevalent system design. This means that the “(...) need to communicate at any time depends on the system concept in effect at that time.” (Conway, 1968)

In other words, if the ‘formal documents’ of Brooks (1995) undergo change, so will the system concept in effect at any one time, and as a result organizational structure will unavoidably evolve. Therefore, Conway (1968) concludes, “flexibility of organization is important to effective design.”

Weber (2004) argues that in open source, in contrast to closed source software, technical architecture drives thinking about organization. Informally, through free and uncontrolled establishment of communication links, the software architecture “(...) sets a stake in the ground around which formal organizational structures take form and evolve” (Weber, 2004). For this reason, “(...) organization should follow [architectural] evolution rather than lead it.” (Weber,
Problems arise otherwise, for example when an organization becomes difficult to change and thus negatively influences a system’s software architecture (Weber, 2004). The desired direction of influence, according to Weber (2004) (‘technical architecture drives thinking about organization’), leads to a positive effect of organization on software architecture. This argument draws on the consequence of the isomorphism from the structure of a system to the form of its design organization.

In this research we assume Conway’s Law to be true. Therefore, we will interchangeably use terms that refer to the technical architecture and organization of open source communities. In addition, this assumption allows us to make a meaningful connection between structural properties like hierarchy and modularity and organizational concepts like meritocracy and clustering. Furthermore, we believe that reflections on organization can be extended to investigate the influence of a given software architecture on code production. This is discussed in more detail in Chapter 6.6.

Research in software engineering coordination ((Herbsleb and Grinter, 1999a), (Herbsleb and Grinter, 1999b), (Herbsleb and Mockus, 2003)), has consistently acknowledged the validity and relevance of Conway’s Law. And although such research has produced results that show the relevant presence of other dimensions of decision-making in software engineering, research in software engineering coordination has consistently strived to theoretically characterize and make predictions about coordination on the basis of Conway’s Law. Research in scientific computing and reverse software engineering ((Aranda et al., 2008), (Bowman and Holt, 1998)), has tested Conway’s Law. Recently, Aranda et al. (2008) found commonalities in the structure of scientific software development groups including boundaries, coordination, and project leadership, which were reflected in the structure of the software produced by theses groups. Bowman and Holt (1998), introduced the idea of an ownership architecture and showed that it is a good predictor for the underlying software architecture. Such research findings can be considered an empirical validation of Conway’s Law. Lastly, Conway’s Law has also been considered in innovation management literature, for example in (MacCormack et al., 2008). It has also been adopted in organizational and political science research on open source, particularly by (Weber, 2004), who employs Conway’s Law as a foundational argument against self-organization.

2.2 Self-organization

Self-organization focuses on how coordination is achieved in decentralized phenomena.

Much recent research argues that open source communities are self-organizing systems (Raymond (1999); Kuwabara (2000); Axelrod and Cohen (2001); Madey (2002); Van Wendel de Joode (2005); Robles et al. (2005); Heylighen (2007)). A common thread in self-organization literature is focused on explaining how coordination is achieved in spite of the lacking influence of collective institutions. Coordination appears to be the crux of the matter of self-organization since it
is centrally discussed in literature. It shows up in explanations of bottom-up organization, arguments about the irrelevance of hierarchy and descriptions of modular divisions of labor.

It seems commonsensical to focus on coordination, especially since the otherwise centralized mindset\(^{24}\) needs some explanation for how decentralized phenomena can achieve the type of coherence found in self-organization. However, there is a surprising lack of agreement about the extent to which centralized coordination can be allocated across decentralized phenomena and how much of an impact that would have on distributed code production.

Another reason to address the lack of coordination is because of the apparent abundance of local interactions between contributors. Due to the apparently high number interactions, contributors have been compared to social insects ((Valverde et al., 2006), Van Wendel de Joode (2005)). The study of social insects like ants, termites and bees is one of the foundations of self-organization (Bonabeau et al., 1999). Observations of these insects indicate that their collective productions are the result of myriads of local nonlinear interactions and not the outcome of centralized coordination and control (Theraulaz et al., 2003). Therefore, self-organization is a metaphor that aims to explain how code production works in open source communities because these communities are characterized by the apparent lack of centralized coordination and control.

### 2.2.1 Decentralized coordination

Decentralized coordination has been studied in different ways, yet what is common in all self-organizing systems is the phenomenon of clustering.

Van Wendel de Joode (2005) explains how global order is achieved through an organizational model of open source. He writes, “(...) open source communities are self-organizing; the local interactions of individuals emerge into collective patterns of behavior” (pg. 212). The claim there is not that centralized institutions like hierarchy are completely absent, but rather that their role in the organization of open source communities is minor when compared to individual behavior (Van Wendel de Joode, 2005) (pg. 207). There is much agreement among proponents of self-organization regarding the irrelevance of hierarchy as a centralized coordination instrument in open source communities ((Bonaccorsì and Rossi, 2003); (Garzarelli et al., 2002); (Raymond, 1999)). However, there are various organizational aspects of coordination that must be fulfilled somehow.

Van Wendel de Joode (2005) describes mechanisms like elegance and modularity that relieve the need for coordination (pgs. 84-85). He also discusses technical mechanisms, like revision control systems, used to coordinate massive amounts of individual efforts (pgs. 86-93). And the organizational mechanisms are said to arise from individual choice and behavior, and aggregated into collective patterns of behavior (pg. 207). Therefore, it follows that there are patterns of behavior that produce better coordination designs than others. In other words, self-organization is purposeful to the extent that it does not result
Theory

2.2 Self-organization

in randomness. Perhaps it can be combined with other aspects to result in more purposiveness (and thus less randomness). We will revisit these expectations in the next sections in relation to predictions found in literature.

Other researchers argue along similar lines, for instance claiming that evolution solves the problem of coordination. For example, Kuwabara (2000) argues that coordination in open source communities is achieved through evolutionary processes that lead to self-organization. Yu (2008) shows how self-organization leads to adaptation in functional and quality requirements, a process which in closed source software relies almost completely on centralized coordination. Another line in this argument is that of emergence. Some authors believe the function of coordination in self-organizing systems emerges from a ‘self-correcting spontaneous’ organization of work (Garzarelli et al. (2002) (pg. 4); Raymond (1999) (pg. 52)).

In contrast to evolution and emergence, Crowston et al. (2007) pay attention to individual contributors and find that ‘self-assignment’ of tasks is the most common coordination mechanism in self-organizing communities. Others focus on how collective behavior is influenced by changes in the environment and how that leads to coordination (Robles et al., 2005). Similarly, Heylighen (2007) finds that coordination is achieved through stigmergy (Grassé, 1959), a collective type of hand-waving that efficiently brings contributors and tasks together, directly or indirectly (Bonabeau et al., 1999) (pg. 14). Here communication is based on individuals modifying their environment and in turn adapting to the modifications of others.25

Researchers who study the Apache development process argue that coordination is present but has been minimized in the face of pressures that include expectations for success and the threat of fragmentation through divergent lines of development ((Fielding, 1999) (pg. 42); (Mockus et al., 2000) (pg. 271); (Mockus et al., 2002) (pg. 310)). Therefore to understand decentralized aspects of self-organization we will focus on a phenomenon that is common in all self-organizing systems. This phenomena is known as clustering, it is primarily a bottom-up process of organization.

2.2.2 Bottom-up organization

Clustering is a bottom-up process of organization that is brought about by modularity.

Bottom-up is an organizational approach for software development in which the composition of sub-communities gives rise to larger community systems in an emergent fashion. In such systems centralized coordination, for instance embodied in collective choice arrangements, is believed to be absent. Instead individual behavior is said to be more influential in the organization of open source communities (Van Wendel de Joode, 2005) and therefore in code production. The bottom-up practice is often characterized as working from the grassroots, where decision-making by individuals co-authoring source code files brings them together into clusters of authorship (Ghosh, 2004) (pg. 18). These clusters are also referred to as swarms (Van Wendel de Joode, 2005) (pg. 136-
2.2 Self-organization

137), which are sub-communities that become interlinked to others in a process of clustering. The result of such interlinking is an overall structure that comprises a top-level community.

The bottom-up aggregation of individual decisions, like those to co-author source code files, has been reported to arise from tags (Van Wendel de Joode, 2005). Tags are artificial properties of an agent that are detectable and can be copied (Axelrod and Cohen, 2001). Examples of tags include elegance of source code, reputation of developers and the level of activity in a project (Van Wendel de Joode, 2005) (pg. 130-132). Van Wendel de Joode (2005) describes how these mechanisms influence individual decisions and result in the non-random formation of clusters of developers. These clusters are comparable to loosely coupled organizations (Freeman, 1991) (pg. 512). Furthermore, the phenomenon of clustering is related to the expectation of non-randomness that was presented in Section 2.2.1.

Clusters of developers arise as a result of splitting up complex software into different activities (Van Wendel de Joode, 2005) (pg. 134). As developers work on these activities they take on different roles and expand their reputation. The result is that the level of activity in the (sub-)community changes and (more) developers come together to form clusters of authorship (Ghosh, 2004) (pg. 18). Such occurrences have been said to be an organizational consequence of modularity. For instance, modularity has been proposed as “(...) the engineering corollary to what organizational theorists call “loose coupling.”” (Weber, 2004) (pg. 172). This explains why one of the research sub-questions put forth in Section 1.4.1 could also have been phrased in terms of clustering (in place of modularity). In other words, in this research we consider clustering as a phenomenon that is brought about by modularity. And from here onwards we will use both terms interchangeably, unless otherwise noted.

Predictions about the influence of clustering on code production

Clustering results from modularity, which is mainly recognized as a mechanism for divergence (Van Wendel de Joode, 2005) (pg. 123). Therefore, expectations of increased code production related to divergence may also apply to clustering. Divergence leads to the expectation of increased code production: many branches, forks and variants (Egyedi and Van Wendel de Joode, 2004). Therefore, the general prediction is that divergence leads to increased code production.

There are also aspects of convergence in clustering, for instance the formation of clusters of authorship (Ghosh, 2004) (pg. 18). Convergence leads to expectations of coherence that prevent incompatibility and avoid fragmentation (Egyedi and Van Wendel de Joode, 2004). For instance, high levels of activity are associated with convergence of software releases, which leads to its inclusion in distributions and lists (Van Wendel de Joode, 2005) (pg. 134). This in turn attracts more individuals and companies to participate in code production (Van Wendel de Joode, 2005) (pg. 134). Therefore the general prediction is that convergence leads to increased participation which results in higher code
self-organization.

Relevant predictions and observations related to the aforementioned tagging mechanisms are also found in literature. For instance, Crowston et al. (2003) propose that the number of active developers in an open source project can predict success in code production. Others associate reputation with quality. Raymond (1999) for example, writes that:

“There’s a very strict meritocracy (the best craftsmanship wins) and there’s a strong ethos that quality should (and indeed must) be left to speak for itself.” (Raymond, 1999) (pg. 89)

Another relevant observation about quality is the so-called Linus’s Law; this law states that “Given enough eyeballs, all bugs are shallow” (Raymond, 1999) (pg. 30). This observation suggests that higher (quality) code production comes as a result of increasing levels of activity. Other authors combine expectations and propose that reputation leads to higher quality and increased levels of activity (Weber (2004) (pg. 142); West and O’Mahony (2005)). Similarly to convergence, there is a prediction of higher levels of code production in these expectations, and this is because such beliefs arise from observations of increased participation.

Many of the predictions that are found in literature are based on increased participation. Many researchers show that increased participation is manifested in quality and elegance of source code, expanded reputation of contributors and higher levels of activity, all of which indicate higher levels of code production. Increased participation has also been shown to lead to the formation of clusters of developers. This phenomenon is also generally associated with higher code production. Therefore, the general predictions are that increased participation results in higher code production and that increased participation leads to the formation of clusters of authorship which in turn result in higher code production.

What is not clear in literature is the relative importance of mechanisms that rely mainly on factors of production and those that result in factors of organization. For instance, which factor is more influential in code production: non-organized, potentially random, increases in participation or self-organized (non-random) clusters of participation? More concretely, the question is whether increases in the factors of production by themselves constitute a good enough explanation for increases in code production or if there are more comprehensive explanations? The following quote summarizes what has been recognized as missing from explanations found in literature:

“Eric Raymond famously said about open source, “with enough eyeballs all bugs are shallow.” What is clearly missing from that statement, and is ultimately important, is how those eyeballs are organized.” (Weber, 2004) (pg. 234)

To this we add that what is ultimately important is to understand how open source communities are organized and how organization influences code
production. Such an understanding could for example improve firm participation in open source communities. It could also lead to more efficient hybrid forms of organization that combine the best aspects of various organizational approaches for software development. Another possibility would be to actively monitor the bottom-up organizational aspect of open source communities to understand how evolution influences code production.

*Understanding the influence of clustering on code production gives insights into its role in how open source communities are organized and how they sustain themselves.*

### 2.2.3 Irrelevance of hierarchy

*Proponents of self-organization argue that hierarchy is absent and irrelevant across decentralized phenomena and does not explain how open source communities are organized and how they sustain themselves.*

Hierarchy is an important notion in self-organization for it symbolizes centralized coordination. In self-organization centralized coordination is believed to be lacking or completely absent due to the decentralized nature of emergent phenomena. However, *conflict resolution* in the absence of hierarchy poses a problem for explanations of self-organization. Open source communities have been said to be rout with conflicts that emerge from high levels of diversity and interdependency in the participants and their activities (Van Wendel de Joode, 2005) (pg. 103-104). Therefore, the role of coordination for conflict resolution must be fulfilled somehow.

In light of the propensity for conflict in open source communities, mechanisms have been sought in self-organization that reduce the consequences of high levels of diversity and interdependency. Van Wendel de Joode (2005) finds such mechanisms in *modularity*, the *culture of doing* and *parallel lines of development*. Modularity has been found to take away interdependencies and result in the localization and isolation of conflicts (Van Wendel de Joode, 2005) (pg. 105). In terms of software architecture, modularization results in an enormous variety of software architectures which have arisen informally:

> “It is interesting to note that we do not have named software architectures. We have some intuition that there are different kinds of software architectures, but we have not formalized, or institutionalized, them.” (Perry and Wolf, 1992) (pg. 41)

The culture of doing, which appears to be related to *meritocracy* (see Section 2.3.3), has been found to be more influential than hierarchical bodies in Apache like the ASF and the Project Management Committee (PMC) (Van Wendel de Joode, 2005). And the other remaining mechanism, namely parallel lines of development, has been found to be effective in deflecting conflict (Van Wendel de Joode, 2005) (pg. 111). Therefore, mechanisms other than hierarchy have been found to be more influential in conflict resolution. And these mechanisms are mainly related to clustering.
Arguments against hierarchy

Proponents of self-organization argue strongly against hierarchy. They claim that hierarchy in open source is irrelevant. For example, Bonaccorsi and Rossi (2003) argue that the bottom-up approach favors decentralization and completely abandons hierarchy:

“In the new approach, the hierarchically organised and top down planned structure adopted in all productive processes is abandoned in favour of a new kind of bottom up structure, which is non-coercive and largely decentralised.” (Bonaccorsi and Rossi, 2003) (pg. 1244)

Madey (2002) affirms that the lack of central control and planning are essential properties that make open source a “(...) prototypical example of a decentralized self-organizing process” (pg. 1807). Other researchers agree with this characterization. For instance, Garzarelli et al. (2002) point out that self-organization and the absence of hierarchy are intrinsic organizational characteristics of open source. Furthermore, they write, “The open source philosophy assures a ‘self-correcting spontaneous’ organization of work” (Garzarelli et al., 2002) (p. 4).

Raymond (1999) equates self-organization in open source with “Unix traditions of modularity, APIs and information hiding” (pg. 221). He identifies these as elaborate and efficient means missing in organizational prescriptions that resolve to hierarchy. Raymond (1999) opposes traditional arguments that favor centralized coordination, like for example Brooks’ Law. Furthermore in opposing such arguments he takes a stance against hierarchy:

“The Linux world behaves in many respects like a free market or an ecology, a collection of selfish agents attempting to maximize utility, which in the process produces a self-correcting spontaneous order more elaborate and efficient than any amount of central planning could have achieved.” (Raymond, 1999) (pg. 52)

Predictions about code production in the presence of hierarchy

Raymond (1999) attributes low levels of code production to hierarchy, especially when compared to the self-correcting spontaneous order created through modularity, and therefore through clustering. Expectations of increased code production that are relevant to both modularity and clustering include parallel lines of development (Van Wendel de Joode, 2005) (pg. 111), the identification of optimal performance frontiers (MacCormack et al., 2008), and highly scalable collaboration (Vixie, 1999).

There are also predictions with regard to the culture of doing, which is mainly related to clustering. Mockus et al. (2002) find that a small core of developers contribute most new functionality and code (80% and 88% respectively) (pg. 321). They also find that a larger core than 10 to 15 people leads to a separation of labor (clustering) into several related open source projects.
The culture of doing is also self-fulfilling. In other words, the culture of doing implicitly predicts higher code production, where increased ‘doing’ implies higher code production. This is also true with parallel lines of development, where more parallel lines of development implies higher code production.

The predominant influence of clustering on code production, over that of hierarchy, gives insights into the non-hierarchical organization of open source communities in relation to their sustained code production. In other words, organization can be understood through code production and since clustering is argued to play a predominant role over hierarchy, this means that, if the argument holds, open source communities are predominantly self-organized.

2.2.4 Modular division of labor

Clustering comes as a result of modularity and in turn induces and reinforces a modular division of labor. This means that labor is divided by modules across communities. Since modularity leads to clustering, the formation of clusters gives insights into the organization of open source communities in relation to their sustained code production. High-level coordination across clusters is achieved through reputation.

Modularity is a principal notion in self-organization for it relieves the need for centralized coordination (Van Wendel de Joode, 2005) (pgs. 84 - 85). Modular task decomposition has been argued to reduce coordination costs ((Parnas, 1972); (Moon and Sproull, 2000); (Dafermos, 2001); (Kogut and Metiu, 2001a); (Garzarelli et al., 2002), (Lerner and Tirole, 2002) (pg. 28); (Van Wendel de Joode, 2005) (pg. 161-162)). Modularization has been defined to be “(...) a responsibility assignment rather than a sub-program” (Parnas, 1972) (pg. 1054).

This implies that modularization has organizational consequences, because it involves commitments between individuals, and one of the consequences is a reduced need for coordination. Despite the benefits, modularization poses some problems:

“The first problem is to decide how to divide the software into smaller modules and how to decide what part of the software should belong in what module.” (Van Wendel de Joode, 2005) (pg. 162)

“The second problem is that of coordination between modules and across communities.” (Van Wendel de Joode, 2005) (pg. 162)

In a seemingly paradoxical way, problems that result from modularity are addressed by modular design. Modularity has been argued to reduce complexity and lead to the creation of roles (Van Wendel de Joode, 2005). Splitting up complex software into different roles that end up being performed through a variety of roles seems to be an organizational consequence of the technical principle of modularity. For instance, modularity has been proposed as “(...)
the engineering corollary to what organizational theorists call “loose coupling.”” (Weber, 2004) (pg. 172). Modularization improves communication between software developers and makes software code more manageable (Weber, 2004). Furthermore, it enables an unplanned division of labor that has been said to be emergent, based on self-selection (Van Wendel de Joode, 2005).

Still, however, the role of coordination between modules and across communities must be fulfilled somehow. Van Wendel de Joode (2005) proposes that distributions and the level of reputation are tertiary coordination mechanisms that achieve the necessary high-level coordination. Arguably this also may occur in an emergent fashion.

### Tertiary coordination through reputation

Tertiary coordination enables coordination between modules and across communities. For instance, it allows for the coordination of changes outside of core functionality between users, developers and core developers (O’Reilly, 1999) (pg. 37). Some authors claim this is one of the reasons open source software is more modular than closed source software ((O’Reilly, 1999); (Vixie, 1999); (MacCormack et al., 2008)). Specifically, the advantage is that tertiary coordination takes advantage of the loose coupling induced by modularity to provide coherence in a cooperative participation environment where centralized control is absent. A similar approach has been recognized in multiagent research on Artificial Intelligence, where multiagent systems that exhibit emergent behavior are believed to be *manageable without the need for centralized control* (Russell and Norvig, 2010) (pg. 429-430).

Some authors argue that tertiary coordination is a high-level task that is performed by the leader(s) of an open source community. Raymond (1999), for instance, claims that one of the preconditions for bazaar style development is the presence of a ‘coordinator’ that can “(...) recognize good design ideas from others” (pg. 47). Prestige in such cases is the factor that attracts attention and procures cooperation from contributors (pg. 84). Other researchers explicitly suggest that there are higher level mechanisms for coordinating contributions. For example, Mockus et al. (2002) report that in Apache contributions among core developers are coordinated through mutual trust (pg. 325). Figure 2.1 illustrates this point by showing the web of trust formed by Apache developers who have signed each other’s cryptographic public keys. These are keys used to produce digital signatures and for encrypting messages. Signing another person’s public key signifies trust in that person (Garfinkel, 1994).

"The underlying keyring [from Figure 2.1] is based on an accumulated version of all the different KEYS files used on apache.org. It currently consists of 78 keys (77 valid, 1 revoked) and a total number of 3254 signatures (1396 valid, 1764 unknown, 94 revoked)."

Another tertiary coordination mechanism proposed is the level of reputation (Van Wendel de Joode, 2005) (pg. 162). This mechanism is arguably connected
to the creation of roles since it is through specialization that contributors gain reputation and get assigned a role.

**Predictions about code production in a horizontal division of labor**

Open source software has been found to be inherently more modular than proprietary software (MacCormack et al., 2008). As a result of modularization, open source has leveraged massively parallel and distributed software development (Narduzzo and Rossi, 2003). Modularization encourages a horizontal division of labor, whose emergence comes as a result of contributors choosing their own activities (Van Wendel de Joode, 2005). This includes selecting the modules where they wish to contribute and deciding how to split up the source code to accommodate those contributions. Modular design increases the number of possible activities and enlarges the space for participation, it is said to be a required property for code production in open source (Vixie, 1999).

Narduzzo and Rossi (2003) argue that horizontal and vertical division of labor are apparent in separate levels of a system’s software architecture. They portray two levels: user space and kernel space. These levels are present in any type of software mainly as core and non-core components. Narduzzo and Rossi (2003) portray the development of non-core components as anarchic, decentralized and distributed. They argue that code production of non-core components is mainly horizontal – without any need for coherence but instead “(...) fostered by the highly modular architecture” (Narduzzo and Rossi, 2003) (pg. 21).
Raymond (1999) portrays open source as a ‘great babbling bazaar’ (pg. 21) that produces code horizontally through modularity and results in increased code production.

Others agree that modularity increases code production. It does so by reducing complexity (Parnas, 1972), decomposing complexity (Van Wendel de Joode, 2005) and by enabling software developers to understand code interdependencies and structure within the limits of bounded rationality (Haefliger et al., 2009). Differences in modularity have been found to account for improvements in code production. For example, MacCormack et al. (2008) reveal significant differences in modularity that indicate the tendency of larger more distributed teams to develop products with more modular architectures. Consequently, the result is increased code production where collaboration scales more efficiently (Vixie, 1999). Moreover, a horizontally prominent division of labor leads to higher code production (Raymond, 2003).

In general the predictions about modularity found in literature are that modularity results in higher (code) code production. Such expectations are related to the many advantages of modularity in aspects like design, management of complexity, task decomposition, reduction of cost, among others. However, none of the predictions found in literature are specific about how modular structure leads to different expectations about code production. One exception is MacCormack et al. (2008), who show how the existence of performance frontiers leads to different expectations about code production. Such performance frontiers are shown to be influenced by the structure and level of modularity. These properties of modularity are observable in clustering, the factor discussed in Section 2.2.2 which has been synthesized from literature and believed to be an influential factor of organization in open source communities.

The predominant influence of clustering on code production, over that of hierarchy, is argued to come as a result of horizontal (non-hierarchical) division of labor induced by modularization. The degree of clustering gives insights into the level of modularity in a software codebase and thus into the structure of that codebase. Since we assume Conway’s Law to be true (as explained in Section 2.1), this means that clustering also gives insights into the level of modularity in the organization and structure of an open source community represented as a network graph (as will be explained in Chapter 3).

2.3 Institutionalization

Institutionalization focuses on how coordination is achieved in centralized phenomena through rules.

There is another stream of organizational research in open source. It is spearheaded by researchers who explain organization through the institutionalization of collective mechanisms that have been found to be instrumental in coordination. Many researchers in this stream often criticize explanations
based on self-organization. They attribute a more significant role to hierarchy and favor explanations that show how open source communities organize around institutions. These researchers particularly focus on institutions that exert top-down influence on the communities, many of which have been found to be influential in code production. Here we will focus on mediation and arbitration, project leadership and revisited voting systems, and roles and their associated level of reputation.

2.3.1 Centralized coordination

Centralized coordination indicates the presence of hierarchy.

A large number of researchers focus on instruments that are central in achieving coordination in open source. The main distinctions between such instruments are based on how centralized they become. Some communities are more centralized than others. For example, the development process in Linux has been reported to be highly centralized, distributed and hierarchical ((Kogut and Metiu, 2001a); (Weber, 2004); (Lerner and Tirole, 2002)). In other communities, such as Apache, strong centralization of authority is vested in committees that resolve disputes through voting or consensus ((Fielding, 1999); (Lerner and Tirole, 2002)). Particularly emphasized in the state-of-the-art is the distributed role of leadership in Apache ((Fielding, 1999); (Bonaccorsi and Rossi, 2003)).

Many researchers of centralized instruments in open source criticize self-organization. One of the primary criticisms is that self-organization fails to explain how a global order is achieved (Weber, 2004). Another source of criticism stems from the metaphor that equates open source communities to self-organizing systems found in nature (Valverde et al., 2006). For instance Weber (2004) states that arguments of self-organization are precluded by the ‘unnatural state’ of interactions in open source, because “[t]here is no state of nature on the Internet.” (pg. 132). He further stresses that open source communities interact with non-self-organized organizations such as “corporations and lawyers and governments and bureaucracies” (Weber, 2004) (pg. 132). Such interaction makes open source communities also subject to non-local events that permeate their organization.

Centralized coordination has been shown to be relevant in production communities, such as open source communities (O’Mahony and Ferraro, 2007). For instance, it has been shown that coordination and linking of individual efforts to community goals is vital to leadership in mature and large-scale open source communities (O’Mahony and Ferraro, 2007) (pg. 1100). This finding comes from the observation that “(...) leaders embraced more organization-building behaviors over time (...) [and that a] developer’s [technical] effort alone did not have the same effect” (O’Mahony and Ferraro, 2007) (pg. 1100). It can be argued that community-building efforts come together in institutions for mediation and arbitration like the ASF and the Project Management Committee (PMC) of each community.

The findings of O’Mahony and Ferraro (2007) imply that success in code production can be attributed to community-building efforts aimed at conflict reso-
2.3 Institutionalization

Such findings enlarge the role of meritocracy to also include organization-building behaviors. Thus the culture of doing (Van Wendel de Joode, 2005) frames a relevant role for a technical hierarchy with organizational functions. It also poses an alternative to the reduction of interdependencies through modularity (Van Wendel de Joode, 2005). The difference is mainly that modularity brings about divergence while the community-building is an organized production that results in convergence. Furthermore, such findings carry the expectation of increased code production through centralized coordination.

The reported patterns of non-local evolution indicative of the presence of hierarchy in open source networks have been claimed to be related with a centralized core of developers that posses a global view of the system (Valverde and Solé, 2007). One of the main findings is summarized as follows:

“The observed community organization indicates that even distributed systems develop internal hierarchies, thus suggesting that some amount of centralized, global knowledge might be inevitable.” (Valverde and Solé, 2007)

2.3.2 Top-down organization

Top-down organization is brought about by collective choice arrangements such as voting systems and leadership because merit permeates the system.

Top-down is an organizational approach towards software development in which communities are decomposed into sub-communities by relying on centralized coordination. The process is frequently portrayed as planned or hierarchical, where community structure is determined from the top. But in open source the control aspect of top-down processes is much more detached and loose. In other words, top-down organization is a governance model which is about “(...) setting parameters for voluntary relationships among autonomous parties” (Weber, 2004) (pg. 172).

In Apache, the governance model is meritocracy (Erenkrantz and Taylor, 2003). Contributors buildup a reputation and in the process gain influence over a project. Such influence is recognized as a top-down instrument by the community, which means contributors become undisputed leaders and there is an official organization that institutionalizes this role (Kogut and Metiu, 2001a) (pg. 253). The institution is meritocracy and it is present in all open source communities of the ASF (Erenkrantz and Taylor, 2003). Here individual behavior is important in its own right, but only becomes influential in its individually aggregated (not collective) form over time.

Leadership in the Apache communities is determined through meritocracy. The way it is determined is by the quantity and quality of contributions: “the more work you have done, the more you are allowed to do” (Fielding, 1999) (pg. 43). In this sense, meritocracy is an instrument for collective choice since individuals affected by the outcome of a decision-making process, namely that of vying for leadership, are able (and encouraged) to participate (Ostrom, 1990).
Furthermore, the outcome is a form of collective choice that binds and restricts members of the collective (Van Wendel de Joode, 2005) (pg. 122). To be specific this happens through the work contributors have performed and what and how much that entitles them to do (hence influence) in return.

In addition, the voting system in Apache is binding in the sense that only votes cast by Apache Group members are considered binding (Fielding, 1999) (pgs. 42-43). Votes bind such contributors to the collective judging from the amount of work they have previously performed. Therefore, the real binding votes are those that are expressed through contributions in the form of source code.

“(...) the basic rule is that only PMC members have binding votes, and all others are either discouraged from voting (to keep the noise down) or else have their votes considered of an indicative or advisory nature only.”

Predictions about the influence of collective choice arrangements on code production

Mockus et al. (2002) write that “[v]otes are generally reserved for major changes that would affect other developers who are adding or changing functionality” (pg. 317). Furthermore, they suggest a relationship between voting and committing source code:

“Each AG member can vote on the inclusion of any code change, and has commit access to CVS (if he or she desires it).” (Mockus et al., 2002) (pg. 317).

This implies that votes have the potential to influence code production through the (non-)inclusion of source code contributions. And since votes are generally reserved for major changes (Mockus et al., 2002), such (non-)inclusion events would represent significant amounts of source code, thereby increasing the potential for substantial influence on code production.

Voting systems are doubly-binding when they are considered as contributions to source code and indications of the willingness to be bound. Such a double-binding nature constitutes a prediction about the influence of voting on code production:

“In some cases and communities, the exercise of a vote carries some responsibilities that may not be immediately obvious. For example, in some cases a favourable vote carries the implied message ‘I approve and I’m willing to help.’ Also, an unfavourable vote may imply that ‘I disapprove, but I have an alternative and will help with that alternative.’”

More concretely the expectation is that the exercise of the right to vote brings with it an implied willingness to contribute. Thereby, the collective
choice arrangement of voting systems in Apache seems to have a positive influence on code production. The prediction is therefore that voting systems are generally associated with increased or at least sustained code production.

Leadership is a collective choice arrangement that is relevant in open source communities (Van Wendel de Joode, 2005). The nature of leadership has been found to be an important determinant of project success. For instance, Lerner and Tirole (2002) argue that leadership is a key component in the prevention of forking (pg. 23). Forking is “(...) the ability of anyone to take a copy of the source code and use it to start a competing project” (Fogel, 2005) (pg. 88).

Forking indicates coordination failure (Narduzzo and Rossi, 2003). It may significantly reduce the number of contributors in an open source community, for instance from participants that migrate to other communities (Weiss et al., 2006). Therefore, the expectation drawn from literature is that leadership prevents forking by ensuring coordination. The prediction is that leadership is generally associated with increased or at least sustained code production.

2.3.3 Relevance of hierarchy

Proponents of institutionalization argue that hierarchy is present and relevant in explaining how open source communities are organized and how they sustain themselves. For instance, hierarchy is relevant because it is a determinant factor in conflict resolution.

The presence of hierarchy in conflict resolution mechanisms

Many researchers have called attention to conflict resolution mechanisms in open source (Fielding, 1999); Raymond, 1999); (Erenkrantz and Taylor, 2003); (Elliott and Scacchi, 2003); (Weber, 2004); (Van Wendel de Joode, 2005). Some authors claim that these mechanisms for resolving disputes are present in two implicit rules in open source from which higher level mechanisms follow. Other researchers find that the mediation and arbitration mechanisms that follow from the two implicit rules are not relevant in conflict resolution. Both stances are ultimately derived from the principle of meritocracy, which Weber (2004) and Hecker (2010) identify in open source licenses. Often in literature, the case of Apache is used as a differentiating example (e.g. Van Wendel de Joode (2005)). We find that this is because it is a case where leadership and conflict resolution are distributed (Fielding, 1999).

The first of the two implicit rules is a rule that determines who is entitled to have decision-making rights on a particular piece of source code. Raymond (1999) claims there is a territorial rule in effect in which ‘authority follows from responsibility’ (pg. 103). This means that a contributor who takes responsibility for a piece of source code is entitled to have decision-making rights on that piece of code. Such rights originate from the ownership claim that (groups of) contributors can have on (parts of) an open source project (Raymond, 1999). Namely this claim is the right to distribute modified versions of the source code (Raymond, 1999), which basically means that contributors abide by an open
source license. Rideau (2010) adds that it is not the ideas that are owned in open source, but the space of participation these ideas consume:

“Software Projects are as intrinsically ownable (and own and – to be owned) as any physical phenomenon; they have an identity in the physical world, independent of the parts of the noosphere that have been explored already and that are intended to be explored later; they consume physical resources, mainly hackers’ hard work.” (Rideau, 2010).

Raymond (1999) explains the three ways that exist for groups to acquire ownership in an open source project. The first is to start the project from scratch. The second way is to take ownership through handover from another contributor. And the third way is almost identical to the second, with the slight difference that ownership is acquired by taking it from someone else. It is important to note that ownership claims are not restricted to individual contributors, projects may be owned by groups (Raymond, 1999) (pg. 73). Also important is that ownership is granular to the file level, it does not necessarily refer to ownership of an entire open source project. These two aspects have implications on authority, which relates to the implicit rule described previously, and on seniority which refers to the implicit rule that will be described next.

The second implicit rule focuses on contributions over time. Raymond (1999) claims this is a spanning rule in which ‘seniority wins disputes’ (pg. 103). This rule becomes apparent to contributors through the span of their involvement in open source communities. This means that a contributor who takes responsibility for a piece of source code over an extended period of time is recognized as an authoritative figure that can resolve disputes relating to that piece of code. Again, such rights originate from the ownership claim that (groups of) contributors can have on (parts of) an open source project (Raymond, 1999).

Weber (2004) clarifies the origins of the two implicit rules discussed previously, by adding that authority not only follows but also derives from responsibility (pg. 163). He adds that coding ability and level of effort are measures of seniority that can be used in resolving disputes (pg. 163). But since these measures are not usually carried out during conflict resolution, the rules remain implicit and their enforcement relies on the tacit knowledge of contributors gained through participation. In addition, these rules originate from merit (Hecker, 2010) and are thus implied from the principle of meritocracy:

“The more work an individual contributes and the more she takes responsibility for a piece of the project, the more decision-making authority she gains.” (Weber, 2004) (pg. 163)

The two implicit rules for conflict resolution are claimed to be widely recognized in open source communities, as collectives, and not only by some of the individuals that comprise them (Raymond, 1999). Therefore it would seem possible to have roles in a community that are based on authority and seniority.
When the implicit rules that give shape to roles have been recognized by the community the next step is to enforce them. The formal creation of roles indicates a willingness by the community to enforce and abide by rules (O’Mahony, 2003a). Mediation and arbitration have been suggested as explicit higher-level mechanisms to enforce implicit rules:

“(…) develop written codes of good practice for resolving the various sorts of disputes that can arise in connection with open-source projects, and a tradition of arbitration in which senior members of the community may be asked to mediate disputes.” (Raymond, 1999) (pg. 110)

In the Apache communities these higher level mechanisms have been codified into institutions, namely the ASF and the Project Management Committee (Van Wendel de Joode, 2005) of each open source community. The key aspect of these institutions is the principle of meritocracy, which enables various roles that are determined by the level of merit and the extent of ownership in code that a contributor has accumulated over time (Fielding, 1999); (Erenkrantz and Taylor, 2003)). These are indicators of authority and seniority and are therefore also suggestive of the presence of hierarchy in conflict resolution. More specifically, the presence of hierarchy in code production can most readily be perceived in conflict resolution roles that influence the release of source code.

**Conflict resolution roles**

The presence of hierarchy is also apparent in the determination of roles for conflict resolution. Some believe that contributors who have amassed enough merit also increase their ownership stake in source code (in the conception of ownership that comes from open source licensing) ((Raymond, 1999) (pg. 76); (Weber, 2004) (pg. 180-181); (Hecker, 2010)). Following the implicit rules for conflict resolution (Raymond, 1999), those same contributors will also increase the influence of their opinions when it comes to releases. This applies especially to pieces of source code where a particular conflict may be isolated or localized, as has been said to occur with modular code (Van Wendel de Joode, 2005). It also applies to conflicts that are limited to sets of source code files—the so-called commit wars that happen when “(…) people working on the same parts of code keep overwriting each others changes” (Bauer and Pizka, 2003) (pg. 5). Furthermore, influential opinions in relation to conflict and releases may also be observed in forks (Fogel, 2005) (pg. 88), which happen when conflict extends to the entirety of an open source project and becomes endemic to the community.

**Conflict resolution for releases**

Fogel (2005) describes two forms for conflict resolution for releasing source code, namely dictatorship by release owner and change voting for releases. The former is more readily associated with Linux where there is a benevolent dictatorship at work. In the case of Apache, there is a voting system in place
2.3 Institutionalization Theory

for conflict resolution (Fielding, 1999). However, despite the presence of a voting system to distribute authority and control, there still is the single role of ‘release manager’, which is a voluntarily contributor who temporarily receives code ownership (Kogut and Metiu (2001b) (pg. 254); (Mockus et al., 2000)). This individual contributor is mainly responsible for keeping track of changes and approvals, following-up with unfinished tasks and merging batch changes to source code (Fogel, 2005) (pg. 177). As the name implies, such a role is managerial whereas the leadership role in Apache is shared (Fielding, 1999).

The release owner is the “(...) person [who] has final say over what changes make it into the release” (Fogel, 2005) (pg. 175). This individual contributor does not need to be the same person as the project leader or benevolent dictator (Fogel, 2005). Therefore, in a meritocracy like Apache, where the leadership role is shared (Fielding, 1999), every contributor is essentially the release owner for the source code over which he or she has ownership. Tasks that come up during the time of a release are offloaded to a release manager (Fogel, 2005). And conflicts that escape the authority and influence acquired by merit are resolved through voting. Thus change voting for releases coupled with release ownership on segments of an open source project and release managers for the entire project constitute a ‘pyramidal flow’ (Weber, 2004), “(...) a hierarchical structure where decision-making flows through a fairly well defined pyramid” (Weber, 2004) (pgs. 185-186).

Predictions about code production in the presence of hierarchy

Kogut and Metiu (2001b) observe a substantial influence of hierarchical organization on code production. They describe how tasks in large modules of Apache are assigned to developers who manage the project. Their prediction is that the presence of hierarchy to a large extent explains code production:

“(...) the community is far more hierarchically organized for the actual development of software code than suggested by the metaphor of a population of interacting agents. For the contribution of large modules, Apache and Linux both assign these tasks to developers who manage the project.” (Kogut and Metiu, 2001b) (pg. 260)

Mockus et al. (2002) show that the presence of hierarchy explains the majority of code production in one Apache project. Their observations can be readily extended into a prediction for all Apache projects, that the presence of hierarchy is a forecast to increased code production. However, Mockus et al. (2002) also observe that there is a threshold that leads to fragmentation in a community producing code. Thus their prediction can be extended to include threshold limits on increases in code production.

Weiss et al. (2006) show that the presence of hierarchy in preferential attachment leads to increases in the number of developers. Such increases in the factors of production have been distinguished from increases in the factors of organization in Section 2.2.2. Here we make the same distinction and we note that increased participation results in higher code production and that
increased participation leads to the formation of hierarchical structures in authorship, so-called *pyramidal flows* (Weber, 2004), which in turn result in higher code production.

What is not clear in literature, as also reported in Section 2.2.2, is the relative importance of mechanisms that rely mainly on factors of production and those that result in *factors of organization*. We repeat our previous question, which factor is more influential in code production: non-organized increases in participation or hierarchically structured pyramidal flows of decision-making participation? To further concretize, we emphasize the question — do increases in the factors of production by themselves constitute a good enough explanation for increases in code production or are there more comprehensive explanations?

### 2.3.4 Hierarchical division of labor

*Hierarchy comes as a result of meritocracy and in turn induces and reinforces a hierarchical division of labor. This means that labor is divided by merit across communities. Since meritocracy leads to hierarchy, the formation of nested communities gives insights into the organization of open source communities in relation to their sustained code production. High-level coordination across nested layers is achieved through merit.*

As discussed in Section 2.2.4, some authors claim that an organizational consequence of the technical principle of modularity is the partition of software into activities that end up being performed through a variety of roles. This claim has been taken towards seemingly opposite directions in literature. For instance, it has been used to argue for emergence in literature on self-organization while in other publications, such as those from the research stream of institutionalization, the claim is employed to propose that modularization can be used for organizational planning.

Some researchers argue that modularity is a mechanism to decompose complexity and that it leads to an emergent division of labor that is based on self-selection (Van Wendel de Joode, 2005). Others, such as Weber (2004) suggest that modularization can be formally planned in a way that has organizational implications. Weber (2004) writes that modularity is “(...) the engineering corollary to what organizational theorists call “loose coupling.”” (pg. 172). Modularity has been said to be an instrument to attract talent to an open source community, hence firms that sponsor open source projects invest significant resources in creating or increasing modularity (O’Mahony and West, 2008) (pg. 9).

Many researchers explain the importance of modularity in relation to technical design and its implications on organization. They use the example of the re-design of the Apache HTTP Server, which increased modularity and simplified participation for contributors (Mockus et al. (2000); Van Wendel de Joode (2005) (pg. 85); Weber (2004) (pg. 110)). The specific consequences on organization that researchers write about are fostered parallel development, improved software extensibility, and increased efficiency in distributed development. The
2.3 Institutionalization Theory

re-designed architecture was published as a set of design considerations (Thau, 1996), and is centrally mentioned on the website of the Apache HTTP Server project. Below are some of the mentions of such modular re-design found in literature.

“In 1995 Robert Thau re wrote the entire NCSA server to make it entirely modular. The server became much easier to maintain. You could work on part of the server, without having to worry that you would damage the rest... [Developers] could work in parallel without stepping on each other’s toes.” (Van Wendel de Joode, 2005) (pg. 85)

“Robert Thau reworked much of the software to improve the modularity of the code so distributed development in an open source process could proceed more efficiently.” (Van Wendel de Joode, 2005) (pg. 85)

“Robert Thau designed a new server architecture (code-named Shambhala) which included a modular structure and API for better extensibility, pool-based memory allocation, and an adaptive pre-forking process model.”

These excerpts from literature show that modular design can be used by an individual to plan for increased efficiency. Such a re-design would be comparable to a benevolent dictatorship where final decision-making authority rests with one person (Fogel, 2005). A division of labor would not be apparent in such a case and a hierarchy without any span would be found to be at work. However, a division of labor would be apparent in a consensus-based democracy, where decision-making authority is distributed among members of a group (Fogel, 2005). Hierarchy would also be present in the sense described in section ??.

This is the case in Apache, where leadership has been distributed among a group of members that have acquired authority through the merit of their contributions (Fielding, 1999). In such a meritocracy, ownership is shared and authority distributed among committers and members with organizational roles. In this way, the modular design of Apache software constitutes a hierarchical division of labor which is reflected in the modular design of its communities.

Still, however, the role of coordination between modules and across communities must be fulfilled somehow. In open source communities, tertiary coordination has been shown to be achieved through the emergence of governance (O’Mahony and Ferraro, 2007).

Tertiary coordination through governance: meritocracy

As stated in Section 2.2.4, tertiary coordination enables coordination between modules and across communities. There it was argued that tertiary coordination takes advantage of the loose coupling induced by modularity and how loose coupling has been recognized as a way to manage emergent behavior without the
need for centralized control (Russell and Norvig, 2010) (pg. 429-430). Furthermore, it was argued that reputation is a high-level coordination mechanism that can be used in managing loosely-coupled community systems. Trust was also mentioned as a coordination mechanism among Apache contributors (Mockus et al., 2002).

Some authors believe the repute and trust (Raymond, 1999) behind roles are parts of the recognition acquired through the principle of meritocracy, which says that ‘the more you do, the more you are allowed to do.’ In fact, the ASF affirms that meritocracy enables various roles. Roles are a recognition in the community of the merit an individual contributor has amassed. Furthermore, roles are a high-level coordination instrument as can be observed in the list of members of the ASF and PMCs. All members there have gained recognition in one or more Apache communities through the merits of their contributions.

Mockus et al. (2002) distinguish mainly two types of roles in code production, core developer and non-core developer. They find that the top 15 developers in Apache contribute most new functionality and code (80% and 88% respectively) (pg. 321). The ASF defines these roles as developer and committer. Furthermore, the ASF claims that what distinguishes these roles from each other is the level of reputation, or merit, that a contributor has accumulated in the community. The ASF refers to this governance principle as meritocracy.

“The Apache project is a meritocracy – the more work you have done, the more you are allowed to do.” (Fielding, 1999) (pg. 43)

Predictions about code production in a vertical division of labor

Narduzzo and Rossi (2003) argue that horizontal and vertical division of labor are apparent in separate levels of a system’s software architecture. They portray two levels: user space and kernel space. These levels are present in any type of software mainly as core and non-core components. Narduzzo and Rossi (2003) portray the development of non-core components as anarchic, decentralized and distributed. They argue that code production of non-core components is mainly horizontal – without any need for coherence but instead “(...) fostered by the highly modular architecture” (Narduzzo and Rossi, 2003) (pg. 21).

Development of core components displays a vertical division of labor. Narduzzo and Rossi (2003) categorize the development of these components as being regulated, integral, coherent, structured and hierarchical. Furthermore, they write that development of core components is “(...) fueled by a highly structured and hierarch[ical] process of review and selection (albeit not based on formal authority but rather on competence and reputation)” (Narduzzo and Rossi, 2003) (pg. 22). The predictions about code production in a hierarchical division of labor are based on the efficiency of the hierarchical process that determines the vertical distribution of such labor.


2.4 Reconciling self-organization and institutionalization

The state-of-the-art literature on organization in open source is based on two opposing research streams, namely that of self-organization and institutionalization. The former argues bottom-up and favors mechanisms that lead to emergence while the latter argues top-down and favors instruments that exert organizational influence. The foundation of these arguments appears to be found in the contrasting engineering approaches of analysis and synthesis. Therefore it seems reasonable to attempt a type of reconciliation that can lead to a consolidated model of organization in open source communities, especially since such a model would combine aspects from both explanations.

Attempts to combine and consolidate new models from competing explanations can be found in qualitative and quantitative research on open source ((Van Wendel de Joode, 2005); (Valverde and Solé, 2007)). There is however, a lack of unified qualitative and quantitative research that combines opposing explanations from self-organization and institutionalization. Such research is believed to be important because it can complement existing qualitative models of organization in open source by testing them quantitatively. For this reason we believe that reconciling competing explanations should focus on findings from quantitative research, such as those of Valverde and Solé (2007).

Self-organization

Valverde and Solé (2007) distinguish different roles that a community can play in terms of how it is organized. In self-organized systems, higher level properties are known to emerge from local (bottom-up) interactions among network components. Here the community functions as a distributed intelligence. As previously discussed this is a view that is promoted in literature on self-organization in open source ((Raymond, 1999), (Kuwabara, 2000), (Axelrod and Cohen, 2001), (Madey, 2002), (Van Wendel de Joode, 2005), (Heylighen, 2007), (Robles et al., 2005)).

Institutionalization

In hierarchical systems, on the other hand, Valverde and Solé (2007) explain that decisions dominate structure and function in a top-down fashion. Such systems are commonly designed from a centralized core often referred to as the ‘kernel.’ The presence of a centralized ordering influence in design, such as a set of collective arrangements influencing the actions of individuals, is advocated in literature that explains organization in open source through institutions ((Weber, 2004); (Kogut and Metiu, 2001b); (Lerner and Tirole, 2002); (O’Mahony and Ferraro, 2007); (O’Mahony and West, 2008)).
2.4.1 Clustering and Hierarchy

Valverde and Solé (2007) find that both institutionalization and self-organization are present in open source social networks. Their view represents a third strand in the state-of-the-art literature on organizational open source. This research strand explains organization through and through. It centers on the combination of top-down and bottom-up processes of interaction ((Valverde et al., 2006), (Valverde and Solé, 2007), (Crowston and Howison, 2005), (Crowston et al., 2006), (Lerner and Tirole, 2000)). Such view is clearly articulated below:

“Our analysis reveals the interplay between bottom-up, distributed decision making periphery in the OSN [open source network] involving many agents and a top-down, centralized small core of agents.”

(Valverde and Solé, 2007)

2.5 Theoretical model of code production

Based on the research streams of institutionalization and self-organization we have constructed a theoretical model of code production. This model is shown below in Figure 2.2. The model shows three of the design principles from the organizational model of open source (Van Wendel de Joode, 2005), as top-down instruments that influence organization. The design principles are included in the model because we are using the organizational model of open source of Van Wendel de Joode (2005) as a guide for research into the organization of open source communities, and because the design principles provide “(...) a way to decide where to look for mechanisms to understand how the communities are organized” (Van Wendel de Joode, 2005) (pg. 38). However, we will not measure particular aspects of the design principles (as will be explained in Chapters 3 and 4), even though we do note they are connected in interesting ways. We discuss this connection in more detail in Chapter 3 (Section 3.2.2).

The theoretical model of code production also shows the bottom-up influence of clustering, a mechanism which as discussed is the result of modularity. In this case the inclusion of clustering is not complicated by issues of operationalization of theory, since we employ a measure that directly refers to the degree of clustering which is itself is a theoretical construct.

Both instruments and mechanism are believed to comprise the factors of organization. Additionally, the model shows the influence of labor and capital through factors of production. Namely these are the number of files and the number of authors found to be involved in code production.
Figure 2.2: Theoretical model of code production
Chapter 3

Model

The subject matter of this Chapter is the discussion of results that are expected from empirical research. Following from the expectations set forth in the theoretical framework constructed in Chapter 2, our focus is now on the formulation of hypotheses that are supported by relevant theory. These expectations combine mechanisms and instruments found in established theories that explain organization in open source communities. The expectations are synthesized into a conceptual model of code production based on which empirical testing can be conducted taking practice as rival theory.

This Chapter unifies expectations from three of the design principles of the organizational model of open source (Van Wendel de Joode, 2005). These principles provide a framework for the comparison of competing explanations of code production. The explanations are based on modularity and hierarchy, two structural notions that are believed to be at work in the organization of open source communities. The conceptualization of these notions involves the separation and unification of their component dimensions, which will in turn be used to formulate hypotheses and construct the conceptual model that is the focus of this Chapter.

3.1 Factors of production

For our first model of code production we adopt the concept of factors of production from the field of economics. Factors of production there are the inputs and resources that are employed in the production of goods and services. Notable among many other factors of production are land and labor. These factors are brought together and configured by entrepreneurs in the creation of new goods and services. Similarly in open source communities, new software releases can be modeled as sets of connected files and contributors, that are not necessarily organized in a particular way. In other words, file co-authorship brings contributors together but does not impose any structure or coordination on their work.

As discussed in Section 2.2.2, much literature suggests that increased participation leads to higher code production ((Raymond, 1999); (Crowston et al., 2003); (Van Wendel de Joode, 2005); (Weber, 2004)). For instance, Crowston
et al. (2003) propose that the number of active developers in an open source project can predict success in code production. Active developers are considered to be contributors when they are co-authoring source code as part of a community. Therefore it follows that the number of source code files being co-authored at any particular moment in time would also be indicative of success in code production. This link has been acknowledged in literature, for instance through the increased reputation that comes from increased file ownership ((Raymond, 1999) (pg. 76); (Weber, 2004) (pg. 180-181); (Hecker, 2010)) (As discussed in Section 2.3.3.) It also seems to make intuitive sense to think that more files are associated with more code and that more contributors imply increased labor which results in higher code production. Therefore the first and second hypotheses of this research are formulated as follows:

\[ H_1: \text{The size of an open source community, in terms of the number of files that are actively co-authored, is positively related to the level of code production in that community.} \]

\[ H_2: \text{The size of an open source community, in terms of the number of contributors who actively co-author files, is positively related to the level of code production in that community.} \]

3.2 Factors of organization

The second model of code production introduces the concept of factors of organization. These are the different ways in which files and contributors can be organized in the production of open source software, which are divided into factors of self-organization and factors of institutionalized organization. Notably, some factors are more efficient than others as can be evaluated by their impact on code production. This means that some factors of organization are more strongly linked to higher levels of code production. Another implication is that factors of organization may impose a particular structure and form of coordination on the work of contributors. This is especially believed to be the case for the factors of institutionalized organization.

As discussed in Section 2.3.2, what is not clear in literature is the relative importance of mechanisms and instruments that comprise the factors of organization in comparison to the factors of production. Furthermore, it is unclear exactly which factors of organization are more influential in code production, whether factors of institutionalized organization such as meritocracy or factors of self-organization such as clustering. The difference between these factors of organization is structural and is linked to theory in how each achieves organization: bottom-up or top-down. Furthermore, these mechanisms and instruments are related to the engineering concepts of analysis and synthesis in how they relate to structure. In addition, an underlying question is whether the factors of organization constitute a more comprehensive explanation for code production in comparison to the factors of production.
3.2.1 Self-organization and code production

The self-organization of code production in open source communities is represented by the decentralization of power into loosely-coupled networks of clusters. The loosely-coupled structure in these networks has been argued to be produced by modularity (Weber, 2004). Research suggests that collective choice arrangements, and the mechanisms that emerge to fulfill the role of such arrangements in self-organizing open source communities, are a compromise between convergence and divergence in code production (Van Wendel de Joode, 2005). Divergence promotes innovation, while convergence is needed to prevent incompatibility and fragmentation (Egyedi and Van Wendel de Joode, 2004). Literature reveals that divergence arises from modularity. It also indicates that convergence can be observed in the aggregation of individual choices through tags (Van Wendel de Joode, 2005). Furthermore, research suggests that convergence and divergence lead to the creation of clusters of developers, which have also been referred to as clusters of authorship (Ghosh, 2004). These are groups of contributors that swarm together into clusters through a process of clustering.

There are broad indications in literature which predict that splitting up complex software into different activities and roles leads to increased participation (Van Wendel de Joode, 2005). This has been shown to occur through the emergence of loosely-coupled clusters wherein contributors specialize and focus on quality and elegance of source code in order to increase their reputation ((Raymond, 1999); (Van Wendel de Joode, 2005)). Research suggests that these mechanisms reinforce the process of clustering. The general prediction found in literature, with respect to convergence and divergence in self-organizing open source communities, is that modularity leads to the emergence of networks of clusters where increased participation results in higher code production.

Literature on open source organization suggests that open source communities form clusters of intense participation where contributors collaborate closely with each other. This leads to the expectation that in open source communities there is a high degree of clustering. Secondly, research suggests that mechanisms that emerge from tags lead to increased participation through clustering. Increased participation has in turn been shown in literature to result in higher code production. This suggests that a high degree of clustering results in increased code production. Thus, the third hypothesis of this research is that:

\[ H_3: \text{The degree of clustering in an open source community is positively related to the level of code production in that community.} \]

3.2.2 Institutionalization and code production

The institutionalization of code production in open source communities is represented by the centralization of power into a decision-making hierarchy. This form of hierarchy has been referred to as pyramid (Weber, 2004), to distinguish it in terms of authority from the conventional notion of hierarchy. The pyramid of decision-making is believed emerge in code production. For this reason such an emergent form of hierarchy is believed to amount to an informal institution,
3.2 Factors of organization

from which a formal institution is expected to follow shortly. Furthermore, it is believed that this notion of hierarchy, which can be separated into component dimensions, is more sophisticated and thus better suited for explaining code production than the simplistic form of hierarchy employed in literature.

The simplistic form of hierarchy has been called *archetypical formal hierarchy* (Simon, 1997) (pg. 197). This notion of hierarchy does not seem suitable to describe and understand decentralized ownership and authority structures in open source communities because of the different conception of property in open source licenses and the consequences this conception brings.

**Arguments in favor of the sophisticated notion of hierarchy**

In his argument of the architecture of complexity, Simon (1962) claims that complexity is organized in levels, that it is composed of self-similar structures and that it frequently takes the form of hierarchy ((Simon, 1962); (Simon, 1997)). Such a notion of hierarchy seems also applicable to the self-organization of complexity in open source, as put forth by many authors ((Van Wendel de Joode, 2005); Valverde et al. (2006); (Valverde and Solé, 2007); (Crowston and Howison, 2005); (Crowston et al., 2006)). It seems applicable there because of the expectation that hierarchy emerges from simplicity. In that sense, the architecture of complexity is an argument in favor of the presence of hierarchy in the organization of open source communities.

> “On theoretical grounds we could expect complex systems to be hierarchies in a world in which complexity had to evolve from simplicity.” (Simon, 1997) (pg. 216)

Simon (1997) proposes that in social systems there is more than one kind of hierarchy. He argues that beyond the kind of social hierarchy present in formal organizations, other kinds of hierarchy can be determined by recursive groupings based on social interaction:

> “If we make a chart of social interactions, of who talks to whom, the clusters of dense interaction in the chart will identify a rather well-defined hierarchic structure. The groupings in this structure may be defined operationally by some measure of frequency of interaction in this sociometric matrix.” (Simon, 1997) (pg. 186)

Simon (1997) argues that informal organizations evolve naturally into hierarchically structured organizations. He proposes that this happens by assembly or specialization of component subsystems, which enables rapid evolution:

> “The claim is that the potential for rapid evolution exists in any complex system that consists of a set of stable subsystems, hence influenced mainly by the net inputs and outputs of the other subsystems.” (Simon, 1997) (pg. 193)
Model 3.2 Factors of organization

The dimensions of hierarchy

Krackhardt (1994) argues that the presence of hierarchy in informal organizations is a valid empirical question and an appropriate object of research. He proposes that hierarchical forms have implications for organizations, that may or may not be functional. Furthermore, Krackhardt (1994) specifies a method for measuring the degree of hierarchical structure in informal organizations.

The influence of hierarchy on performance, measured through efficiency or production, seems to also be an appropriate object of research. According to Krackhardt (1994) the investigation of such influence must be based on comparisons to a pure hierarchical structure (pg. 93). He proposes the use of graph theory and measures of structure in a directed graph (digraph), a graph where interactions are specified as ordered pairs. The outtree is an idealized hierarchic structure that can be artificially constructed in digraphs (Krackhardt, 1994). Certain conditions are necessary and sufficient for a digraph to be an outtree:

1. The digraph is connected.
2. The digraph is graph hierarchic.
3. The digraph is graph efficient.
4. Every pair of points in the digraph has a least upper bound.
   (Krackhardt, 1994) (pg. 95)

These conditions are considered dimensions of hierarchy in this research. They are explained in more detail below and used to formulate hypotheses about the influence of hierarchy on code production.

**Graph connectedness**  This condition quantifies how the factors of production are connected to each other.

**Graph hierarchy**  This condition quantifies how asymmetric are the connections between the factors of production.

**Graph efficiency**  This condition quantifies how redundant are the connections between the factors of production.

**Upperboundedness**  This condition quantifies how close is the next level in an informal hierarchy.

The dimensions of hierarchy seem to be in interesting ways connected to the design principles in Van Wendel de Joode (2005). Since we have modeled organization of open source communities using social networks, and will be employing social network analysis, we merely note that connection as a theoretical gesture and avoid having it carry the weight of the argument that meritocracy influences code production. Therefore, for the formulation of hypothesizes we straightforwardly derive our expectations in terms of the well-established empirical tradition of how hierarchy is measured in networks. This tradition
includes the theoretical notion of sophisticated hierarchy ((Simon, 1962); (Simon, 1997)) which has been operationalized (Krackhardt, 1994) into the dimensions discussed previously. Thus, the fourth, fifth and sixth hypotheses of this research state the following:

\[ H_4: \text{The degree of graph connectedness in an open source community is positively related to the level of code production in that community.} \]

\[ H_5: \text{The degree of graph hierarchy in an open source community is positively related to the level of code production in that community.} \]

\[ H_6: \text{The degree of graph efficiency in an open source community is positively related to the level of code production in that community.} \]

### 3.3 Operationalized model of code production

Our model of code production has been operationalized as shown below in Figure 3.1. We have substituted the theoretical design principles for their corresponding dimensions of hierarchy and labeled the hypothesized causal connections. This model is based on opposing research streams of literature. However, it does not require that organization be explained solely by institutionalization or self-organization. Instead, this model allows reflection in terms of both extreme forms of organization. This is important because there is literature that calls for considering both self-organization and institutionalization, and the model that underlies this analysis enables that comparison.

### 3.4 Factors exogenous to the model

In this section we reflect on factors that are not in the model and the extent to which they could be relevant. Most of the factors we reflect upon come from Nagappan et al. (2008), who studied the relationship between organizational structure and software quality through metrics that quantify quality and organizational complexity. The metrics for software quality are shown below in Table 3.1. The metrics for organizational complexity are shown below in Table 3.2.
3.4 Factors exogenous to the model

From Table 3.1 we believe factors 1, 2 and 3 could be relevant to our model, because (for factor 1) the number of changes could very well have an influence on the number of bugs, (for factor 2) the status of a file before a change could reveal more information about prior changes and (for factor 3) various Apache open source projects are written in Object Oriented languages. Factor 4 is already something we measure. And factor 5 is not relevant in open source software, since it refers to bugs detected during quality assurance testing for...
3.4 Factors exogenous to the model

<table>
<thead>
<tr>
<th>#</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of Engineers</td>
<td>“The more people who touch the code the lower the quality.”</td>
</tr>
<tr>
<td>2</td>
<td>Number of Ex-Engineers</td>
<td>“A large loss of team members affects the knowledge retention and thus quality.”</td>
</tr>
<tr>
<td>3</td>
<td>Edit Frequency</td>
<td>“The more edits to components the higher the instability and lower the quality.”</td>
</tr>
<tr>
<td>4</td>
<td>Depth of Master Ownership</td>
<td>“The lower level is the ownership the better is the quality.”</td>
</tr>
<tr>
<td>5</td>
<td>Percentage of Organization contributing to development</td>
<td>“The more cohesive are the contributors (organizationally) the higher is the quality.”</td>
</tr>
<tr>
<td>6</td>
<td>Level of Organizational Code Ownership</td>
<td>“The more cohesive is the contributions (edits) the higher is the quality.”</td>
</tr>
<tr>
<td>7</td>
<td>Overall Organization Ownership</td>
<td>“The more the diffused contribution to a binary the lower is the quality.”</td>
</tr>
<tr>
<td>8</td>
<td>Organization Intersection Factor</td>
<td>“The more diffused the different organizations contributing code, the lower is the quality.”</td>
</tr>
</tbody>
</table>

Table 3.2: Organizational Complexity Metrics proposed in Nagappan et al. (2008)

releases in closed source software.

From Table 3.2 we believe factors 2 and 3 could be relevant to our model, because (for factor 2) we do not distinguish between developers who contributed once and then exited the community (this might very well have an influence) and (for factor 3) we do not count the number of edits to particular files (and this might be relevant in terms of levels of activity). Factors 1, 4, 5, 6, 7 and 8 are already metrics we measure, (for factor 1) we measure the number of contributors, and (for factors 4-8) the remaining factors we measure through dimensions of hierarchy (or at least we measure properties that are very similar to these).

3.4.2 Limitations related to exogenous factors

As explained previously, many of the factors reviewed and proposed in Nagappan et al. (2008), are factors we already measure. These factors represent alternative measures, which in most cases are very similar to the measures we employ. Factors that are not in our model, could be relevant as explained in the previous section. However, we believe our model is attractive because of its simplicity and straightforwardness, and introducing these factors would result in additional and unnecessary complexity. In other words, we believe there is more to lose by introducing too many factors (a problem some refer to as the curse of dimensionality, where more variables does not necessarily improve un-
derstanding of a phenomena). Furthermore, we do not have the data available from the logs we collected in order to measure these factors. On the other hand, some of these factors we already measure in an indirect way. But because of the additional complexity we chose not to pursue the inclusion of such indirect measurements either.
Chapter 4

Method

This Chapter describes the strategy and methodology employed to answer the main question and related sub-questions of this research. Our primary objective in this Chapter is to explain how the research was conducted so that it may be replicated and extended in the future. We provide details about sources and techniques that enable the reader to verify the correctness of our results and to repeat this research in practice on any open source community. As well provided is a specification of the research design and implementation, including that of the contributions to an open source software package for data mining and social network analysis of Subversion repositories.

First we present an introduction to the top-level communities of the ASF. This includes an overview of the data collected in this research. We provide a summary of the factors of production, namely the number of contributors and files that were committed during a given sampling period. We also include sample measurements of code production such as number of lines of code and revision counts for a given set of observations.

Next we describe the broad strategy employed in this research. The strategy is basically survey research on the influence of design principles on code production. Then we provide details on the research methodology. We explain how data mining was performed on Subversion repositories, which data was collected, and how it was subsequently pre-processed and analyzed. We discuss the preliminaries of future research using our method.

The final part of this Chapter is a description of the measurement of conceptual constructs in this research. These are the dependent and independent variables used in statistical analysis. Correspondingly, we will describe the statistical analysis procedures that were used and we explain why they were chosen, also in relation to future research.

4.1 Introduction to ASF communities

The ASF currently hosts 78 top-level open source communities, each of which is generally associated with code production in one open source project. The main categories of these projects are shown in Table 4.1. Each project is concerned with the production of software related to the Apache HTTP Server.
For instance, Apache Tomcat produces software that powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations. Such web applications are usually hosted and deployed on Apache HTTP Servers.

There are also open source communities that produce the software that powers the infrastructure used by other projects and communities in the ASF. For example, Apache Subversion allows contributors to maintain current and historical versions of source code files, web pages, and other documentation. Originally the project was a revision control system founded and sponsored by CollabNet Inc. More recently, Subversion became an Apache project and the software they produce is used by virtually all open source communities in the ASF.

A complete alphabetical listing of the top-level open source communities of the ASF is shown in Table 4.2. The description of each open source project and community can be found on the Apache project listing and a description of the latest releases can be found in the ASF release listing. Although there are many differences among the communities, common to all projects is the presence of a role-based management infrastructure. This basically means that each community and its corresponding project are managed by a Project Management Committee, as described in the ASF website:

“All projects in The Apache Software Foundation are managed by a group termed a Project Management Committee. This group determines the general direction of the project and its releases.”

Similar across all open source projects in the ASF is the formation of communities around code, which the ASF describes as ‘community-over-code.’ This is a form of software development where decisions are made by consensus through peer review. What makes ‘community-over-code’ unique is that it emphasizes community building rather than code production. Nonetheless, the result is a tightly integrated community that co-produces code. We show the results of code co-production, which means code production through file co-authorship (and which we call simply code production), in Table 4.3.

Many aspects of code production can be appreciated in Table 4.3. For instance, there appears to be a wide variation in the factors of production. There are marked differences in the number of contributors and the number of files they co-author. Also appreciable are large differences in code production among ASF communities. These are measured in the number of lines of code and the numbers of revisions produced.
### ASF Project Categories

<table>
<thead>
<tr>
<th>Project Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>build-management</td>
</tr>
<tr>
<td>database</td>
</tr>
<tr>
<td>http</td>
</tr>
<tr>
<td>javaee</td>
</tr>
<tr>
<td>mail</td>
</tr>
<tr>
<td>network-server</td>
</tr>
<tr>
<td>retired</td>
</tr>
<tr>
<td>virtual-machine</td>
</tr>
<tr>
<td>xml</td>
</tr>
<tr>
<td>content</td>
</tr>
<tr>
<td>graphics</td>
</tr>
<tr>
<td>httpd-module</td>
</tr>
<tr>
<td>library</td>
</tr>
<tr>
<td>network-client</td>
</tr>
<tr>
<td>regexp</td>
</tr>
<tr>
<td>testing</td>
</tr>
<tr>
<td>web-framework</td>
</tr>
</tbody>
</table>

Table 4.1: ASF Top-level Project Categories

### Example visualizations of ASF communities

Figure 4.1 shows the Apache HTTP Server community. This representation was extracted from code production in HTTP Server during the year 2009. HTTP Server is a top-level community in the ASF where some contributors have a higher level of reputation than others. In Figure 4.1a this is illustrated by the size of individual nodes, which is determined by the number of contributors that have co-authored files with the contributor represented by a particular node and by the weight of individual contributions measured as lines of code. In Figure 4.1b the community has been grouped to show more clearly the differences in reputation represented by the size of nodes.

Figure 4.2 shows the Apache Hadoop community. This representation was extracted from code production in Hadoop during the year 2008. The Figure 6.2 shows the emergence of a cluster from a parallel line of development (a sub-project) in the Apache Hadoop project. As shown in the Figure, the Zookeeper community cluster is completely parallel from the main Hadoop community.

Figure 4.3 shows the Apache Hadoop community. This representation was extracted from code production in Hadoop during the year 2009. Hadoop is a top-level community in the ASF that is composed of sub-communities, Figure 4.3 shows Hive inside the red oval, Pig inside the yellow oval, and HBase inside the green oval. Some of these communities have been integrated into the main Hadoop codebase, while others have been spun-off as separate projects. Currently, the official sub-communities of Hadoop include Common, HDFS, MapReduce, and ZooKeeper.

### 4.2 Research strategy

In this research we employed a survey strategy to answer our main research question. We investigated the behavior of contributors in file co-authorship interactions. These interactions were observed in more than 70 open source com-
4.2 Research strategy

Table 4.2: ASF Top-level Communities

<table>
<thead>
<tr>
<th>ASF Communities</th>
<th>HTTP Server</th>
<th>APR</th>
<th>Camel</th>
<th>Cocoon</th>
<th>CXF</th>
<th>Felix</th>
<th>Hadoop</th>
<th>Jackrabbit</th>
<th>Logging</th>
<th>Mina</th>
<th>OFBiz</th>
<th>PDFBox</th>
<th>Portals</th>
<th>ServiceMix</th>
<th>SpamAssassin</th>
<th>Synapse</th>
<th>Tiles</th>
<th>Tuscany</th>
<th>Web Services</th>
<th>XMLBeans</th>
<th>XML Graphics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Abdera</td>
<td>ActiveMQ</td>
<td>Ant</td>
<td>Cassandra</td>
<td>Click</td>
<td>Forrest</td>
<td>Harmony</td>
<td>Jakarta</td>
<td>Lucene</td>
<td>MyFaces</td>
<td>OpenEJB</td>
<td>Perl</td>
<td>Qpid</td>
<td>Shindig</td>
<td>STDCCXX</td>
<td>Tapestry</td>
<td>Tomcat</td>
<td>UI/MA</td>
<td>Xalan</td>
<td>Xerces</td>
<td>XML Graphics</td>
</tr>
</tbody>
</table>

munities of the ASF during a period of five years, between 2004 and 2009. The data were collected, pre-processed and analyzed in order to test the hypotheses that were formulated in the previous Chapter. These hypotheses concern the influence of the principal institutions identified in three design principles from the organizational model of open source communities (Van Wendel de Joode, 2005). Namely the design principles are: collective choice arrangements, conflict resolution mechanisms, and multiple layers of nested enterprise. As described in Chapters 2 and 3, in order to improve conceptual modeling we decided to split the design principles into their institutional and self-organizational components. Therefore, we examine the influence of different instruments and mechanisms on the extent of code production.

4.2.1 Testing for the influence of design principles

Different findings are possible through a test of the influence of design principles in the organization of code production in open source communities. A first possibility is finding that open source communities are primarily institutionalized. For instance, should any one of the institutions in open source communities that were identified in the design principles be found to play a significant role in code production, then such institutions would go beyond the
### Code Production in ASF Communities

<table>
<thead>
<tr>
<th>Name</th>
<th>Author Count</th>
<th>File Count</th>
<th>Lines of Code</th>
<th>Revision Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abdera</td>
<td>5</td>
<td>2</td>
<td>529</td>
<td>34</td>
</tr>
<tr>
<td>ActiveMQ</td>
<td>16</td>
<td>1,690</td>
<td>447,273</td>
<td>1,123</td>
</tr>
<tr>
<td>Ant</td>
<td>13</td>
<td>4,034</td>
<td>458,559</td>
<td>1,098</td>
</tr>
<tr>
<td>Archiva</td>
<td>8</td>
<td>127</td>
<td>78,265</td>
<td>622</td>
</tr>
<tr>
<td>Cocoon</td>
<td>9</td>
<td>827</td>
<td>52,886</td>
<td>309</td>
</tr>
<tr>
<td>Comdev</td>
<td>5</td>
<td>35</td>
<td>4,006</td>
<td>35</td>
</tr>
<tr>
<td>Commons</td>
<td>39</td>
<td>806</td>
<td>517,997</td>
<td>2,985</td>
</tr>
<tr>
<td>Cxf</td>
<td>13</td>
<td>306</td>
<td>83,845</td>
<td>915</td>
</tr>
<tr>
<td>DB</td>
<td>20</td>
<td>732</td>
<td>377,968</td>
<td>967</td>
</tr>
<tr>
<td>Directory</td>
<td>10</td>
<td>874</td>
<td>104,723</td>
<td>621</td>
</tr>
<tr>
<td>Felix</td>
<td>15</td>
<td>698</td>
<td>89,276</td>
<td>712</td>
</tr>
<tr>
<td>Forrest</td>
<td>8</td>
<td>76</td>
<td>8,925</td>
<td>315</td>
</tr>
<tr>
<td>Geronimo</td>
<td>20</td>
<td>631</td>
<td>69,606</td>
<td>678</td>
</tr>
<tr>
<td>Hadoop</td>
<td>56</td>
<td>15,889</td>
<td>13,238,520</td>
<td>3,965</td>
</tr>
<tr>
<td>Harmony</td>
<td>17</td>
<td>261</td>
<td>174,681</td>
<td>1,309</td>
</tr>
<tr>
<td>HTTP Server</td>
<td>45</td>
<td>1,560</td>
<td>543,373</td>
<td>2,665</td>
</tr>
<tr>
<td>Jackrabbit</td>
<td>18</td>
<td>4,840</td>
<td>5,838,595</td>
<td>2,028</td>
</tr>
<tr>
<td>Jakarta</td>
<td>12</td>
<td>236</td>
<td>190,526</td>
<td>1,257</td>
</tr>
<tr>
<td>Lucene</td>
<td>25</td>
<td>551</td>
<td>98,815</td>
<td>583</td>
</tr>
<tr>
<td>Maven</td>
<td>16</td>
<td>301</td>
<td>14,229</td>
<td>309</td>
</tr>
<tr>
<td>Mina</td>
<td>17</td>
<td>449</td>
<td>89,547</td>
<td>825</td>
</tr>
<tr>
<td>MyFaces</td>
<td>28</td>
<td>1,859</td>
<td>659,840</td>
<td>2,406</td>
</tr>
<tr>
<td>OpenJPA</td>
<td>15</td>
<td>1,850</td>
<td>795,404</td>
<td>1,600</td>
</tr>
<tr>
<td>PDFBox</td>
<td>3</td>
<td>1</td>
<td>56</td>
<td>51</td>
</tr>
<tr>
<td>Perl</td>
<td>9</td>
<td>13</td>
<td>1,683</td>
<td>89</td>
</tr>
<tr>
<td>Poi</td>
<td>7</td>
<td>202</td>
<td>151,389</td>
<td>494</td>
</tr>
<tr>
<td>Qpid</td>
<td>21</td>
<td>1,833</td>
<td>2,130,681</td>
<td>2,524</td>
</tr>
<tr>
<td>Roller</td>
<td>4</td>
<td>500</td>
<td>229,249</td>
<td>148</td>
</tr>
<tr>
<td>ServiceMix</td>
<td>12</td>
<td>1,206</td>
<td>343,089</td>
<td>2,822</td>
</tr>
<tr>
<td>Sling</td>
<td>9</td>
<td>138</td>
<td>88,509</td>
<td>783</td>
</tr>
<tr>
<td>SpamAssassin</td>
<td>16</td>
<td>226</td>
<td>3,637,979</td>
<td>2,287</td>
</tr>
<tr>
<td>Struts</td>
<td>11</td>
<td>1,850</td>
<td>423,091</td>
<td>583</td>
</tr>
<tr>
<td>Synapse</td>
<td>8</td>
<td>181</td>
<td>36,653</td>
<td>591</td>
</tr>
<tr>
<td>Turbine</td>
<td>3</td>
<td>8</td>
<td>4,987</td>
<td>262</td>
</tr>
<tr>
<td>Tuscany</td>
<td>10</td>
<td>2,871</td>
<td>755,788</td>
<td>295</td>
</tr>
<tr>
<td>Velocity</td>
<td>6</td>
<td>28</td>
<td>12,698</td>
<td>259</td>
</tr>
<tr>
<td>Web Services</td>
<td>53</td>
<td>14,121</td>
<td>4,290,391</td>
<td>3,149</td>
</tr>
<tr>
<td>Xerces</td>
<td>8</td>
<td>117</td>
<td>31,794</td>
<td>477</td>
</tr>
<tr>
<td>XML</td>
<td>8</td>
<td>14</td>
<td>52,205</td>
<td>162</td>
</tr>
<tr>
<td>XMLBeans</td>
<td>3</td>
<td>21</td>
<td>3,857</td>
<td>41</td>
</tr>
</tbody>
</table>

Table 4.3: Code Production in ASF Top-level Communities in 2009
argued external role. This role is limited to protecting the communities from outside pressures. It could then be argued that open source communities are primarily organized around institutions.

A second possibility is finding that open source communities are primarily self-organized. For instance, should clustering be found to play a highly significant role in code production, then such a self-organizational mechanism would favor the argued external role for institutions. This would happen by discarding the the possibility of an influential internal role, namely a lack of influence of institutions on code production. Support would then be found for the claim that open source communities self-organize around artificial properties such as tags, and not around institutions. Furthermore, it could be claimed that mechanisms of self-organization significantly influence code production.

A third possibility is a combined form of organization in which aspects of both institutionalization and self-organization are found to be influential in code production. In such event it could be argued that the observed organization of open source communities is a hybrid form that combines aspects of organic and synthetic communities (O’Mahony and West, 2008).

4.3 Research methodology

In our survey research we employed methods from the field of computational sociology. This is a field that uses computationally-intensive methods to analyze and model social phenomena. We used these methods to collect, model and analyze interaction data in a way that would be useful for empirical testing of the influence of design principles on code production and for continued reflection on theoretical propositions about those design principles. The specific methods we employed from computational sociology were data mining and social network analysis.

We employed data mining to collect logs from the ASF’s Subversion repos-
4.3 Research methodology

Figure 4.2: Apache Hadoop community and sub-communities in 2008

...itory and to extract organizational patterns from those logs. Subsequently, we used social network analysis to model and measure structural properties of organization in open source communities. The rationale for choosing such methods is that they are scalable and can be automated. These are computational properties that we were interested in from the start of this research because they would enable the continuous analysis of increasingly complex and large-scale behavioral data sets. The advantages of this approach will be further explored in Chapter 6 (Section 6.6).

4.3.1 Data mining

Data mining is the process of extracting knowledge from large amounts of data that are generally found in repositories and stored for subsequent use in databases and data warehouses (Han and Kamber, 2000). Because the process can vary widely data mining is known by several names, one of which is knowledge discovery. Knowledge discovery is a process where data is transformed for the extraction and measurement of patterns that can be used to represent knowledge (Han and Kamber, 2000). As a technology, data mining for knowledge discovery and representation combines techniques from statistics, pattern recognition and machine learning (Russell and Norvig, 2010) (pg. 24).

In this research we employed data mining techniques to extract knowledge from the Subversion repository of the ASF. Our main aim was to represent...
organizational instruments like institutions and self-organizational mechanisms like clustering. As will be discussed in Sections 4.3.2 and 4.4, both phenomena can be measured using social network analysis. Another motivation for the use of data mining was to employ artificial intelligence techniques on collected data, once the process for data collection and transformation had been determined and centralized in a data warehouse. This is discussed in more length as future research in Chapter 6.6.

Our data mining method was based on the following two steps:

1. Collect Subversion repository logs.
2. Pre-process collected logs to obtain a file co-authorship representation of open source communities.

In order to obtain a file co-authorship representation we employed a base organizational pattern that will be explained shortly. The result was an interaction data set that could be used for social network analysis. This data set contained file co-authorship interactions in source code files. Such files were committed by more than 1,000 contributors in over 70 open source projects of the ASF during a period of five years, between 2004 and 2009. We decided to focus on source code files rather than email messages because source code files include metadata that we believe can be used to gain insights about the

Figure 4.3: Apache Hadoop community and sub-communities in 2009
organization of open source communities and to represent the institutions that we hypothesize are influential in the production of open source software.

Data collection

The data collection phase of this research focused on the collection of Subversion repository logs from top-level projects of the ASF. All open source projects in the ASF are hosted on a single Subversion repository. This repository is mirrored in the US and Europe. The infrastructure setup of the ASF simplified data collection since all projects were found to be accessible through a central server. However, we faced some difficulties in data collection.

The ASF establishes quotas on the amount of data that can be collected. There is also a permissible period of time between successive collection events. This slowed down our data collection substantially. To circumvent these limitations, the ASF infrastructure team provided us with the URL (Internet location) where archived copies of the ASF Subversion repository are made publicly available.

We downloaded an archived version of the Subversion repository, measuring 65 gigabytes in total, that contains all the source code contributions made to ASF projects between 2004 and 2009. The archived repository was mounted on a local server from which log files were collected. For the data collection we used SVNPlot, an open source software package that collects Subversion logs and stores them in a database. Each collection event produced a database with one year’s worth of log data for a single open source project.

The collected Subversion logs gave us access to metadata about the entire change history of the source code of an open source project. The change history includes data about the creation of new source code files and also about changes that were made to existing source code files. Such events are called commits in Subversion. Commits happen every time a contributor sends the changes he or she made to the source code of an open source project. These changes are sent to the central Subversion repository when a contributor invokes the ‘svn commit’ command (Pilato, 2004). We used the available metadata for pre-processing in order to come up with a representation of open source communities that could be subjected to further analysis.

Data pre-processing

The collected logs from the Subversion repository were pre-processed with a modified version of SVNPlot. The output from data pre-processing consisted of two lists and a matrix of social relationships, which in literature is referred to as a sociomatrix ((Simon, 1997); (Krackhardt, 1994)). The first list in the output is an index of contributors who authored source code files. The second list is a record of all the source code files that were modified or newly submitted to the Subversion repository during a specified period of time. For our analyses this period was one year. The resulting sociomatrix combined the two lists
mentioned previously. This matrix had a register of all the relationships between contributors that were established through file co-authorship.\textsuperscript{67}

From the list of contributing authors (who are also called committers), we created a sociomatrix with all file co-authorship interactions. These interactions were either the result of committing a new file to the Subversion repository or the result of committing changes to an existing file in the repository. This meant that the resulting sociomatrix would be a square matrix, since it would register all interactions between contributors in an open source community. In our view, file co-authorship determines the boundaries of an open source community by showing which contributors are active social members in the community. Consistent with that view, each \textbf{row x column} in the sociomatrix generated by \texttt{SVNPlot} represents a social relationship between two contributors that are part of an open source community. The strength of that social relationship is determined by the number of lines of code that these contributors co-authored. Should a contributor cease to co-author source code with others, he or she will not show up in the community representation, and hence will not be considered part of that community.

The pre-processing phase of our data mining approach was inspired by Apache Agora, an open source application to mine and visualize mailing lists as virtual communities and visually understand their shape and behavior.\textsuperscript{68} The main idea behind Agora is described as follows:

\begin{quote}
\textit{“The basic idea is that every time you reply to somebody’s message, you are creating a social relationship between you and that person.”}\textsuperscript{69}
\end{quote}

As described in section 3.2.2, the concept of a hierarchy that emerges from frequency of interaction was originally proposed by (Simon, 1997) (pg. 186). In section 3.2.2 we discussed how the concept of hierarchy was operationalized and measured through the degree of hierarchical structure in informal organizations (Krackhardt, 1994). Accordingly, we based the approach for data pre-processing on the idea that frequency of interaction could give us insights into the organizational structure of code production in open source communities. Therefore, in data pre-processing we focused on establishing which contributors co-authored source code files with other contributors. In this way we could find the contributors that had created a directed social relationship with others in the community.

In order to obtain a file co-authorship representation we employed a base organizational pattern. This pattern consists of the following steps:

1. Determine if a file is the first of its kind in the repository.
2. Alternatively, determine if a file is the first of its kind in the data set.
3. Register the contributor who committed a ‘first of a kind’ file as the primary author of that particular file.
4. Register all other contributors to the same file as co-authors with a directed social relationship to the primary author.
The rationale for this base organizational pattern is that it would apply to any data set, regardless of the period of collection. In other words, it would still be possible to apply this pattern in data sets where the period of collection differed from the period of one year used in this research. Of course, the probability of having a primary author that matches the original author of a file would be higher in data sets with a longer period of collection. But despite possible mismatches, we reasoned that a period of one year would suffice in view of the modeling advantages this trade-off implied.

Another reason to use this organizational pattern is that it allows us to determine the primary author of a file, regardless of whether there is one in the data set or not. Furthermore, by always assigning a primary author to a file we avoid problems that could arise when operating under the closed-world assumption. In this assumption anything that is not known to be true is presumed as a fact to be false (Russell and Norvig, 2010) (pg. 299). Together with the unique-names assumption, where every constant symbol refers to a distinct object, we would comply with database semantics that are used in database systems and logic programming (Russell and Norvig, 2010) (pg. 299-300).

4.3.2 Social network analysis

Social network analysis is the set of methods and analytic concepts which focus on patterns of relations that are indicative of the presence of structure (Wasserman and Faust, 1994) (pg. 3). These patterns are measured through structural variables that quantify relations between interacting nodes in a network. The network perspective has been proposed as a research method that allows new leverage for answering behavioral questions by giving precise formal definitions to aspects of social structural environments where regularities may be observed (Wasserman and Faust, 1994) (pg. 4). Furthermore, network analysis has been advanced as a strategy to test theories about social relational processes and structures that can be evaluated against network data (Wasserman and Faust, 1994) (pg. 5).

In this research we employed social network analysis to measure the clustering coefficient (Watts and Strogatz, 1998) and dimensions of hierarchy (Krackhardt, 1994) in social networks of file co-authorship. These networks were constructed through data transformation (ie. collection and pre-processing as discussed in Section 4.3.1). Our main aim was to measure the degree of organizational instruments and self-organizational mechanisms. One advantage is that these phenomena are measured on a scale between 0 and 1, which can easily be interpreted as a percentage.

Another motivation for the use of social network analysis was to automate the construction of temporal, sequential or time-series-based databases of measurements of organizational phenomena. These are knowledge representation repositories that are updated based on sequential or time-based attributes or on the repeated application of measurements over time (Han and Kamber, 2000) (pg. 17). The availability of such measurements would enable the study of evolution of an open source community. A further extension that was envi-
sioned was the conversion of any event related to committership in a Subversion repository into a data stream that could be used in real-time to monitor organizational phenomena in open source communities. These ideas are discussed in more length as future research in Chapter 6.6.

4.4 Measurement of conceptual constructs

The conceptual constructs employed in this research were measured using *ORA*, a network analysis tool that measures properties of an organization’s design structure (Carley and Reminga, 2010). Namely the conceptual constructs are: graph connectedness, graph hierarchy, graph efficiency and clustering coefficient. Other conceptual constructs were also measured in this research. These were measured using SVNPlot, an open source software tool that creates various types of statistics and graphs from Subversion repository log data. Namely those conceptual constructs include: lines of code, revision count, author count and file count.

It is important to note that all the dependent and independent variables were captured from the same source, namely the Subversion repository of the ASF. However, this does not imply that the measurements reflect in similar ways on the phenomena of code production. As was explained in Section 1.5.1, we make a clear separation between factors of production and factors of organization. We believe that the influence of factors of production on code production does not reflect the same reasoning as that applied when we consider the influence of factors of organization on code production. Therefore, we clearly state our belief that there is no risk of circular reasoning in our approach.

Furthermore, as was explained previously in Section 4.3.1, the collected data were pre-processed in order to measure the factors of organization. This additional step was not performed for the factors of production. Therefore, we believe it is valid to consider these data as coming from different sources, even though we are measuring properties from the same unit of analysis. Moreover, we believe this approach increases the robustness of our data collection and of our findings.

4.4.1 Reliability and Validity

In conducting our measurements we carefully considered the reliability and validity of the measurement constructs. We first focused on ensuring overall precision and accuracy with which conceptual constructs were measured. And then we concentrated on assuring that our measurement instruments achieved their aim.

Reliability

The focus was on repeatability, time independence and testing. These aspects were established and verified by our use of a Python script that was integrated into SVNPlot, which made data collection repeatable and independent of the
time of collection. Furthermore, SVNPlot scripts were systematically tested and debugged. Part of the debugging included matching the data that was collected with statistics available on project size in terms of author count. We obtained this information from various Apache project websites (where developers who are committers are listed and credited) and used that information to verify the results of data collection.

Systematic debugging and testing also included employing different software tools and matching their outputs. We conducted this exercise using \*ORA and SVNPlot, and found their outputs to match with respect to the variables author count, file count and revision count, albeit with different names that each software package gives to data that is collected. We would like to underscore that we consider our use of separate and unrelated software tools for the measurement of dependent and independent variables as a strength of this research and its approach. We believe that the use of separate tools increases the reliability and correctness of the results of this research.

Time independence was the next aspect we considered. In order to test this aspect, we collected data remotely over the Internet and locally from our test environment. In both cases the results were found to be identical, verifying our expectations of repeatability and time independence and confirming that the Python scripts integrated into SVNPlot were free of undesired behavior that could distort our measurements. Furthermore, we used these results as a form of parallel testing where we ran the Python scripts on different computers and instructed them to collect data remotely in one case, and locally in the other.

Validity

Once we had confidence of the precision and accuracy with which conceptual constructs were measured, we focused on the degree to which our measurement instruments achieved their aim. Namely the aim of our instruments was to measure the factors of production and the factors of organization as well as to measure the outputs from code production. We used several approaches to verify that our instruments actually measure what we would like them to measure, namely we cross-checked results, conducted tests with simple networks and conducted tests with extreme forms of organization found in literature (Krackhardt, 1994).

As described previously, we cross-checked the results produced by different software tools. Furthermore, we double checked these results with statistics available on Apache project websites (such as for example the number of developers in an open source project). In addition, we looked at the variance in code production for individual open source projects, and found that for most projects code production does not change significantly from one year to the next. In fact, very few projects show significant variation in code production and we used this knowledge in debugging the Python scripts employed in this research.

In order to test the validity of measurements from the factors of organization we employed simple networks of file co-authorship for which we knew the results
4.4 Measurement of conceptual constructs

Method

beforehand, and for which results were obtained through simple calculations. For example, when testing asymmetry one can take a reduced number of nodes (e.g. 10 nodes) and point all links to a single node, and then change these values gradually to get an understanding of how measurements change. This was a useful exercise in developing the Python scripts employed in this research.

Another strategy to test the validity of measurements from the factors of organization was to measure extreme forms of organization found in literature. Krackhardt (1994), among others, provide examples of extreme forms of trees, such as the outtree, which can be used to test for validity. In addition, the exercise of removing outliers, which will be described in Chapter 5 (Section 5.3) was especially useful to confirm the validity of the measurement constructs. Because of its extensiveness, we leave the explanation of that procedure for Section 5.3.

Limitations

The limitations of this research will be explained in more detail in Chapter 6. However, we would like to state a few limitations which are relevant to the reliability and validity of our method and the results obtained. The primary limitation of this research is that it does not consider the evolution of measurement constructs. Instead we pool all the data together and conduct statistical analyses without taking the time dimension into account. The effect on our findings, as will be explained in Chapter 6, is that we take the time dimension for granted. As will be expanded in Section 6.6, we propose this as a direction of future research, namely to repeat measurements over time and include the time dimension in continued analyses.

4.4.2 Dependent variables

The dependent variables in this research (lines of code and revision count) were chosen because they are intimately related to code production. First, lines of code measures contributions granularly at the file level, which means it can be used to distinguish between files based on the amount of work that has been invested in them. For example, parts of a code base that are under more active development will have higher lines of code. This allows us to distinguish which files are more important to the community from others, and in turn which developers are more active contributors to those files (which is what is professed by meritocracy).

Another reason to employ lines of code as a dependent variable was because adding lines of software code is something that is common to all software projects, regardless of whether they are open or closed source. Furthermore, keeping counts of lines of code makes it easy to distinguish between code production and code re-production. Code reproduction happens when a code base is copied, for instance to create a new branch. Such event might imply that code is just copied but that no work is yet performed. We consider it important to be able to distinguish between these two events. Lastly, lines of code is a metric that is widely used in industry and in academic research.
We decided to employ revision count as a dependent variable in order to compensate for weaknesses in the dependent variable lines of code. One such weakness is that lines of code measures and heavily taxes deletions of source code. As explained in Section 4.3.1, we decided to ignore file deletions and deletions within source code files since such events do not constitute social acts of production but are instead associated with the absence of a social relationship. However, deletions might be coupled with additions to source code (or source code files) and therefore it is important to know when a revision has been made to the repository.

In short, our dependent variables were chosen because they measure relevant aspects of code production. In certain situations measuring this fundamental phenomena requires a focus on slightly different aspects. The dependent variables we chose meet this criteria and increase the reliability and validity of our measurements because either measure compensates for weaknesses in the other. As will be explained in Chapter 5, we believe this choice increases the robustness of our results.

- **lines of code**
  This is a form of code production that is measured through the number of lines that were added to a new source code file or an updated source code file that was committed to the Subversion repository.

- **revision count**
  This is a form of code production that is measured through the number revisions that were made to register new source code files or to update existing source code files in the Subversion repository. Each revision event that is counted in this measurement, is one commit to the Subversion repository and represents a change in the decision-making state and therefore a change in the social network.

### 4.4.3 Independent variables

**Factors of production**

As discussed in Section 3.1, much literature suggests that increased participation leads to higher code production ((Raymond, 1999); (Crowston et al., 2003); (Van Wendel de Joode, 2005); (Weber, 2004)). For instance, Crowston et al. (2003) propose that the number of active developers in an open source project can predict success in code production. Other factors such as the number of source code files being co-authored by active developers at any particular moment in time would therefore also be indicative of success in code production. We explain each factor of production in more detail below.

- **author count**
  This factor measures the number of contributors that were active in co-authoring source code files during the sampling period. Any contributor who did not co-author source code files is not counted, and therefore not
considered to be part of the community. As mentioned in section 4.3.1, file co-authorship determines the boundaries of an open source community by showing which contributors were active social members in the community.
• **file count**

This factor measures the number of files where lines of code were contributed. As discussed in Section 2.3.3, software projects are intrinsically ownable (Rideau, 2010). Through ownership of source code files, determined by file count per author, it is possible to get an idea of the ownership structure in an open source project.

**Factors of organization**

As discussed in Section 3.2, in order to isolate the effects of clustering and meritocracy, and evaluate if there is an effect of factors of self-organization and factors of institutionalized organization, we have grouped together the dimensions of hierarchy and clustering. Hierarchy consists of three dimensions and modularity (clustering) consists mainly of one dimension. Clustering represents a competing argument to that of institutionalization, from the research stream of self-organization as discussed in Chapter 2.

**Factors of institutionalized organization: Hierarchy**

• **graph connectedness**

This condition quantifies how the factors of production are connected to each other.

• **graph hierarchy**

This condition quantifies how asymmetric are the connections between the factors of production.

• **graph efficiency**

This condition quantifies how redundant are the connections between the factors of production.

**Factors of self-organization: Clustering**

• **clustering coefficient**

This condition quantifies the degree to which open source communities form clusters of intense participation where contributors collaborate closely with each other.
Chapter 5

Data Analysis

In this Chapter we describe the results that were obtained in this research. We follow an order in which we first describe how the data was examined per each of the hypotheses that were formulated in Chapter 3. Then we discuss how and why in some cases the data was transformed. Finally, we report the results of multiple-regression analysis and hypothesis testing. We report the results of such analyses and tests with tables and figures throughout the present Chapter. These results will be interpreted in Chapter 6.

5.1 Procedure

Social reality in open source communities of the ASF is complex, the wide range of behaviors that can be observed across different communities raises several methodological challenges. In order to simplify the description of complex social reality in the Apache communities, we employ statistical models and tools. Statistical analysis in this research is limited to describing the outcomes of social processes in the ASF and is not meant as a way to describe the processes themselves. Therefore, in order to gain a deeper understanding of the outcomes of code production we employed two statistical approaches that gave us insights at different levels. Each approach has its specific advantages and disadvantages, but together they are believed to provide meaningful insights.

The approaches we employed are bivariate and multivariate methods using a pooled data design. In all cases we set out to explain differences in code production measured as lines of code and revision count. This means that empirical analyses were conducted for these two dependent variables, and therefore the results that are presented in this Chapter refer to both metrics.

The procedure we followed is the one suggested in Fox (2008). The procedure suggests to first gain an understanding of the data through examination. Univariate displays such as histograms and scatterplots were used to understand how data is distributed, whether there are any visible outliers or data that seemed unusually influential. This initial approach was complemented with bivariate methods that were used to test for simple correlation between variables. Now we could envision the possibility of certain relationships between the constructs that had been measured.
The second step suggested by Fox (2008) is to transform the data in order to comply with the assumptions of multiple regression analysis. Once the data were transformed we removed outliers. Then we were able to apply multiple regression analysis. Following multiple regression analysis we conducted linear-model diagnostics, to address unusual and influential data, and diagnoses for non-normality, nonconstant error variance, nonlinearity and collinearity. No issues were found and so the output of those diagnoses is not included in this Chapter.

It is worth noting that the procedure followed for data analysis resulted in the progressive construction of a model. First we examined the data and this allowed us to consider whether or not to keep the variables that were being examined. Data transformations allowed us to further assess the choice to keep certain variables in the model. In this way, we were building towards a parsimonious model that only included the variables that expressed the relationships that are of interest in this research. After following this process we decided to keep two models, one per dependent variable (lines of code and revision count), and both models using the same independent variables. The intermediate models, which were discarded, are included in Appendix A, and are discussed in Section 5.4.

5.2 Data Examination

To examine the data we will mainly use histograms and scatterplots. These plots are complemented with the outcome of statistical tests performed on the data. These tests will be used to confirm our observations when needed. One such observation is whether data is normally distributed or not. This observation will determine the statistical test we use to find the relatedness between independent and dependent variables. In all cases we will see that the data is not normally distributed. Thus, we will use Spearman’s rank correlation coefficient test.

Our guiding purpose in this section will be to understand the main features of the data that has been collected and pre-processed. We are particularly interested in how the data is distributed and whether it is skewed. The main outcome will be to determine the relatedness between independent and dependent variables.

Outcomes of Production

The dependent variables lines of code and revision count are the outcomes of production that we are interested in explaining. The data for these outcomes is not normally distributed, which means we will need to make use of non-parametric tests to determine the relatedness of these variables to the independent variables. As shown in Figure 5.1, the data are highly skewed, which means that most of the observations are confined to a small part of the range of the data (Fox, 2008). In section 5.3 we will consider the use of power transformations to make these skewed distributions more symmetric.
5.2 Data Examination

(a) loc: 2004-2009  
(b) rev: 2004-2009

Figure 5.1: Histograms of (a) Lines of Code and (b) Revision Count, both from Pooled data (2004-2009)

5.2.1 Factors of production: hypotheses 1 and 2

To test hypotheses one and two, first we need to find out how the factors of production (file count and author count) are distributed. As can be seen in Figure 5.2 and Figure 5.4, the variables that correspond to file count and author count are not normally distributed. Furthermore, the data are highly skewed. This observation is confirmed by the skewness/kurtosis test for normality, which for these variables is found to be at a significant level (p = 0.00).

Hypothesis 1

\[ H_1: \text{The size of an open source community, in terms of the number of files that are actively co-authored, is positively related to the level of code production in that community.} \]

Figure 5.2: Histogram of File Count, from Pooled data (2004-2009)
Spearman’s rank correlation coefficient test

Since file count is not normally distributed we will use Spearman’s correlation coefficient test to evaluate the relatedness of this variable to lines of code and revision count. The outcome of this test indicates that file count is positively correlated with both lines of code and revision count ($\rho = 0.74, p = 0.00$) and ($\rho = 0.68, p = 0.00$). More specifically, the results of the statistical test used indicate that there is a strong correlation between the independent variable file count and the dependent variables lines of code and revision count.

![Scatterplots for (a) File Count vs. Lines of Code and (b) File Count vs. Revision Count, both for Pooled data (2004-2009).](image1)

Figure 5.3: Scatterplots for (a) File Count vs. Lines of Code and (b) File Count vs. Revision Count, both for Pooled data (2004-2009).

Hypothesis 2

$H_2$: The size of an open source community, in terms of the number of contributors who actively co-author files, is positively related to the level of code production in that community.

![Histogram of Author Count, from Pooled data (2004-2009).](image2)

Figure 5.4: Histogram of Author Count, from Pooled data (2004-2009)
Spearman’s rank correlation coefficient test

Since author count is not normally distributed we will use Spearman’s correlation coefficient test to evaluate the relatedness of this variable to lines of code and revision count. The outcome of this test indicates that author count is positively correlated with both lines of code and revision count ($\rho = 0.51$, $p = 0.00$) and ($\rho = 0.82$, $p = 0.00$). More specifically, the results of the statistical test used indicate that there is a moderate correlation between the independent variable author count and the dependent variable lines of code. Furthermore, there is a strong correlation between the independent variable author count and the dependent variable revision count.

![Scatterplots](image)

(a) authorcount vs. loc: 2004-2009    (b) authorcount vs. rev: 2004-2009

Figure 5.5: Scatterplots for (a) Author Count vs. Revision Count and (b) Author Count vs. Revision Count, both for Pooled data (2004-2009).

Preliminary Findings

On Figure 5.3a and Figure 5.5a, we can observe several orders of magnitude difference in code production measured as lines of code and revision count. These quantities correspond to different numbers of contributors (author count). Similarly on Figure 5.3b and Figure 5.5b we observe large differences in terms of numbers of files (file count). In both cases a high variation can be observed. However, despite the significant variation in code production we find that larger open source communities, in terms of both number of files (file count) and number of contributors (author count), are associated with more code production. This is expected because such behavior meets the common intuition of labor: more work done by human beings produces more output.

The preliminary finding that a higher number of files (file count) equates to more lines of code and a higher revision count, is analogous to stating that more land in a farm means more production. This is not necessarily true unless more work by human beings is performed, which can be crudely stated in numbers of human beings performing work. However, it is also possible that such work be performed more efficiently. Other factors then come into play, such as the
factors of organization, which are divided into the factors of institutionalized organization and the factors of self-organization.

In the remaining hypotheses we shall examine different factors of organization that can be measured in a social network, namely those related to self-organization (clustering coefficient) and those related to institutionalization (graph hierarchy, graph connectedness and graph efficiency). We will test which of these accounts for the most significant contribution to production, also in comparison to the non-organized sheer contribution of author count and file count. For now, we ignore organization and preliminarily accept both hypotheses concerning the factors of production.

5.2.2 Factors of self-organization: hypothesis 3

In this hypothesis we consider the factor of self-organization referred to as clustering. Clustering is measured by the clustering coefficient among contributors. Networks with a high clustering coefficient and a small average distance between contributors are associated in the Watts and Strogatz model with small world phenomena. Such phenomena occur in systems that lie somewhere in the spectrum between fully-structured and self-organized systems (Watts and Strogatz, 1998). As was explained in Section 2.2.2, clustering is a consequence of modularity that is associated with bottom-up organization.

To test the present hypothesis we first need to find out how the factor of self-organization (clustering) is distributed. As can be seen in Figure 5.6 the variable clustering coefficient is not normally distributed. This observation is confirmed by the skewness/kurtosis test for normality, which is found to be at a significant level (p = 0.00).

Hypothesis 3

\[ H_3: \text{The degree of clustering in an open source community is positively related to the level of code production in that community.} \]

Figure 5.6: Histogram of Clustering Coefficient (Watts and Strogatz, 1998), for Pooled data (2004-2009)
Spearman’s rank correlation coefficient test

Since clustering coefficient is not normally distributed we will use Spearman’s correlation coefficient test to evaluate the relatedness of this variable to lines of code and revision count. The outcome of this test indicates that clustering coefficient is positively correlated with both lines of code and revision count ($\rho = 0.33, p = 0.00$) and ($\rho = 0.38, p = 0.00$). More specifically, the results of the statistical test used indicate that there is a moderate to low correlation between the independent variable clustering coefficient and the dependent variables lines of code and revision count. This is contrary to our expectation that clustering coefficient would be strongly correlated with code production outputs but since it is only a preliminary finding we will confirm our expectations later in Section 5.4.

Scatter Plots

Figure 5.7: Scatterplots for (a) Clustering Coefficient (Watts and Strogatz, 1998) vs. Lines of Code, and (b) Clustering Coefficient (Watts and Strogatz, 1998) vs. Revision Count, both for Pooled data (2004-2009)
5.2 Data Examination

Preliminary Findings

On Figure 5.7a we can observe several orders of magnitude difference in code production measured as lines of code. These differences seem to be somewhat explainable through simple correlation, thereby from changes in clustering coefficient. Therefore, the expectation (when we employ multiple regression analysis in Section 5.4) is that predictions of the response variable (lines of code) will moderately improve when we base such predictions on the linear relationship between the response variable and the explanatory variable (Fox, 2008) (the explanatory variable being clustering coefficient). Similarly on Figure 5.7b we observe large differences in terms of code production measured as revision count. For revision count, it will also become apparent whether differences in the response variable will be better explained by changes in the explanatory variables.

The observations discussed are reinforced with a fitted least-squares regression line (in red) and a line for lowess smooth with a span of 1/2 (in blue), as shown in Figure 5.7. In Figure 5.7b the fitted least-squares regression line (in red) seems to be more steep than on Figure 5.7a. This indicates a stronger correlation and thereby leads to the expectation of an improved prediction of the response variable in terms of the explanatory variable clustering coefficient.

Despite the significant variation in code production we find that open source communities with a higher clustering coefficient between contributors are associated with higher code production. This is compatible with expectations of self-organized behavior, such as those set forth in theory as explained in Section 2.2.2. More specifically the expectation that comes to mind here is that developers work on activities that buildup their reputation and as a result the level of the activity in the (sub-)community changes and more developers come together to form clusters of authorship (Ghosh, 2004).

In the remaining hypotheses we shall examine different aspects of the factors of institutionalized organization, such as connectedness, asymmetry and redundancy between the factors of production. These are properties of hierarchy that can be measured in a social network (graph connectedness, graph hierarchy and graph efficiency). We will test how each of these factors of institutionalized organization accounts for significant contributions to code production outputs, also in comparison to the non-organized sheer contribution of author count and file count. For now, we ignore the aspects of institutionalized organization and preliminarily accept the hypothesis concerning factors of self-organization.

5.2.3 Factors of institutionalized organization: hypotheses 4, 5, 6

In these hypotheses we consider the factors of institutionalized organization jointly referred to as hierarchy. Hierarchy consists of four components graph connectedness, graph hierarchy and graph efficiency (and upperboundedness which as discussed in Section 3.2.2 will not be considered). As discussed in Section 3.2.2, these are dimensions of hierarchical structure in informal organizations. Hierarchical structure can occur anywhere in the spectrum between
fully-structured and self-organized systems. To measure the extent of full-
hierarchical structure, Krackhardt (1994) proposes that comparisons be made
against a pure hierarchical structure, a so-called outtree (pg. 93). We consider
dimensions of hierarchy as top-down instruments of institutionalized organiza-
tion as explained in Section 2.3.2.

To test the present hypotheses we first need to find out how the factors
of institutionalized organization are distributed. As can be respectively seen
in Figure 5.8, Figure 5.10 and Figure 5.12, the variables graph connectedness,
graph hierarchy, and graph efficiency are not normally distributed. These ob-
servations are confirmed by the skewness/kurtosis test for normality, which for
these variables is found to be at a significant level (p = 0.00).

Hypothesis 4

\[ H_4: \text{The degree of graph connectedness in an open source community is positively related to the level of code production in that community.} \]

Figure 5.8: Histogram of Graph Connectedness (Krackhardt, 1994), for Pooled data (2004-2009)

Spearman’s rank correlation coefficient test

Since graph connectedness is not normally distributed we will use Spearman’s
correlation coefficient test to evaluate the relatedness of this variable to lines
of code and revision count. The outcome of this test indicates that graph con-
nectedness is positively correlated with both lines of code and revision count
(\( \rho = 0.26, p = 0.00 \)) and (\( \rho = 0.29, p = 0.00 \)). More specifically, the results
of the statistical test used indicate that there is a moderate to low correla-
tion between the independent variable graph connectedness and the dependent
variables lines of code and revision count.
5.2 Data Examination

Data Analysis

(a) graph connectedness vs. loc
(b) graph connectedness vs. rev

Figure 5.9: Scatterplots for (a) Graph Connectedness (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Connectedness (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2009)

Hypothesis 5

\[ H_5: \text{The degree of graph hierarchy in an open source community is positively related to the level of code production in that community.} \]

Figure 5.10: Histogram of Graph Hierarchy (Krackhardt, 1994), for Pooled data (2004-2009)

Spearman’s rank correlation coefficient test

Since graph hierarchy is not normally distributed we will use Spearman’s correlation coefficient test to evaluate the relatedness of this variable to lines of code and revision count. The outcome of this test indicates that graph hierarchy is negatively correlated with both lines of code and revision count ($\rho = -0.38$, $p = 0.00$) and ($\rho = -0.34$, $p = 0.00$). More specifically, the results of the statistical test used indicate that there is a moderate and negative correlation between the independent variable graph hierarchy and the dependent variables lines of code and revision count.
Hypothesis 6

$H_6$: The degree of graph efficiency in an open source community is positively related to the level of code production in that community.

Spearman’s rank correlation coefficient test

Since graph efficiency is not normally distributed we will use Spearman’s correlation coefficient test to evaluate the relatedness of this variable to lines of code and revision count. The outcome of this test indicates that graph efficiency is negatively correlated with both lines of code and revision count ($\rho = -0.14$, $p = 0.00$) and ($\rho = -0.13$, $p = 0.00$). More specifically, the results of the statistical test used indicate that there is a low and negative correlation between the independent variable graph efficiency and the dependent variables lines of code and revision count.
5.2 Data Examination

Data Analysis

Preliminary Findings

On Figure 5.9a, Figure 5.11a and Figure 5.13a, we can observe several orders of magnitude difference in code production measured as lines of code. These differences seem to be explainable at different levels through simple correlation, thereby from changes in graph connectedness, graph hierarchy and graph efficiency. Therefore, our predictions of the response variable (lines of code) are expected to moderately improve when we base such predictions on the linear relationship between the response variable and the explanatory variables (Fox, 2008) (graph connectedness, graph hierarchy and graph efficiency). Similarly on Figure 5.9b, Figure 5.11b and Figure 5.13b we observe large differences in terms of code production measured as revision count. For revision count, it will also become apparent whether differences in the response variable will be better explained by changes in the explanatory variables.

The observations discussed are reinforced with a fitted least-squares regression line (in red) and a line for lowess smooth with a span of 1/2 (in blue) as shown in Figure 5.9, Figure 5.11 and Figure 5.13. In Figure 5.9b the fitted least-squares regression line (in red) slopes positively and seems to be more steep than in Figure 5.9a. For Figure 5.11b the same relationship appears to hold; the fitted least-squares regression line (in red) slopes negatively and seems to be more steep than in Figure 5.11a. For Figure 5.13b it is difficult to tell which relationship applies. It is not clear from the figure whether the fitted least-squares regression line slopes positively or negatively and therefore the magnitude of steepness can not be easily determined.

The common pattern in the data is different to that reported in Section 5.2.2. For the factors of institutionalized organization, the correlation between the independent and the dependent variables (lines of code and revision count) seem to be more or less at the same level.

Our preliminary finding is that open source communities with higher graph connectedness are associated with higher code production. The inverse relationship appears to hold for graph hierarchy, for which increases in that variable are associated with decreased code production. This seems to indicate that...
the more connected the factors of production the higher the code production, while the more asymmetric those connections the lower the code production. Predictions in terms of the redundancy of connections between the factors of production remain somewhat impartial.

5.3 Data Transformation

Fox (2008) proposes data transformation as a way to conform to restrictive assumptions of classical statistical models like least-squares regression (pg. 50). Furthermore, he indicates that transforming data can facilitate data examination and statistical modeling. Data transformation can be used to make distributions more symmetric, linearize the relationship between two variables, and equalize variation across groups (Fox, 2008) (pg. 50). One family of transformations is power transformations. Fox (2008) explains that power transformations are particularly useful to make skewed distributions more symmetric (pg. 54).

In the previous section (5.2), the skewness of certain variables was noted. Fox (2008) specifies a method to analytically select transformations, based on examination of the ratio between the median value and the hinges (extreme values) of a distribution. He explains that one should select a transformation for which the following ratio is closest to the value of ‘1’ (pg. 55).

\[
\frac{H_U - \text{Median}}{\text{Median} - H_L}
\]  

(5.1)

Here, however, we will make use of the \texttt{gladder} function in \texttt{Stata}, the statistical software package used for data analysis in this research. The \texttt{gladder} command suggests different transformations for a specific variable and shows the corresponding output distribution for each variable per transformation. The transformation that should be chosen is that for which the output distribution appears to be more symmetric, this is what the ‘1’ value suggested by Fox (2008) indicates. The variables selected for data transformation are shown below in Table 5.1. And the output of the \texttt{gladder} command in \texttt{Stata} is presented in Appendix A (Section A.2).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>lines of code</td>
<td>( \log_{10}(X) )</td>
</tr>
<tr>
<td>revision count</td>
<td>( \sqrt{X} )</td>
</tr>
<tr>
<td>filecount</td>
<td>( \sqrt{X} )</td>
</tr>
<tr>
<td>authorcount</td>
<td>( \log_{10}(X) )</td>
</tr>
<tr>
<td>hierarchy</td>
<td>( \sqrt{X} )</td>
</tr>
<tr>
<td>connectedness</td>
<td>( X^2 )</td>
</tr>
</tbody>
</table>

Table 5.1: Variables selected for data transformation
Removing outliers

The presence of outliers was detected in the data, specifically in the values of graph hierarchy. A large number of observations (62 out of 257) displayed a graph hierarchy value of ‘0’. Before removing the outlying values we reasoned why these outliers were present in the first place. Measurement errors were unlikely since all the data were collected and processed in an automated fashion, hence human error was avoided and could be reasonably discarded. The explanation that was found relates to the definition of graph hierarchy and the time period of collection for each unit of analysis. First we will start with the definition.

As explained in section 3.2.2, graph hierarchy is a measure of structure in a directed graph (digraph) where interactions are specified as ordered pairs and compared against an idealized structure called an outtree (Krackhardt, 1994). The measure of graph hierarchy reflects on the level of asymmetry in connections between pairs:

\[
\text{"[I]n a digraph } D, \text{ for each pair of points where one } (P_i) \text{ can reach another } (P_j), \text{ the second } (P_j) \text{ cannot reach the first } (P_i)." \text{ (Krackhardt, 1994) (pg. 97)}
\]

The graph hierarchy condition quantifies how asymmetric are the connections between the factors of production. The degree of graph hierarchy is defined as:

\[
\text{Graph hierarchy } = 1 - \left[ \frac{V}{\text{MaxV}} \right] \tag{5.2}
\]

where \( V \) is the number of symmetrically linked pairs and \( \text{MaxV} \) is the total number of pairs (ie. all pairs, including both symmetrically and asymmetrically linked pairs). A value of ‘0’ for graph hierarchy is obtained whenever \( V = \text{MaxV} \), that is, when the number of symmetrically linked pairs is equal to the total number of pairs. This makes \( \frac{V}{\text{MaxV}} = 1 \). Therefore, should enough connections be present such that all pairs become symmetrically linked, then graph hierarchy will have a value of ‘0’. This is misleading in that it does not really represent the condition of asymmetry in the network.

When enough data are collected from an open source community, it is possible to have cases in which all members of the community interacted with each other at least once. In such cases, graph hierarchy would have a value of ‘0’. Therefore, we reason that the outlying values that are present in the data are due to the period of collection of one year for file co-authorship interactions in open source communities. One possible solution is to reduce the period of collection, but that would involve starting over. Instead, we chose to remove the outlying values. After removing outliers, the histogram for graph hierarchy is distributed as shown below in Figure 5.14.

It is important to note that it is not necessary to re-run the correlations after data transformation. The output of spearman correlation, for instance, is preserved after transformation of the variables. In other words, the same
value will be obtained regardless of the transformation. Furthermore, it is not necessary because in the preliminary findings we were only interested in getting an idea of how the dependent variables are related to the independent variables. What does change, however, is the output of multiple regression analysis. The reason it changes is that after transformation the data better conforms to strict assumptions of multiple regression analysis. This will be explained in more detail in Section 5.4.

(a) graph hierarchy with outliers  (b) graph hierarchy without outliers

Figure 5.14: Histogram of Graph Hierarchy (Krackhardt, 1994), with and without outliers, for Pooled data (2004-2009)

Boxplots before and after data transformation

Table 5.2 and Table 5.3 show boxplots of the independent variables that have a distribution between 0 and 1. These values can be easily compared in a boxplot. The main transformations that can be noted are for graph hierarchy and graph connectedness, which in Table 5.3 have more symmetric distributions around a center which is more or less consistent for all variables.

Correlation matrix

Table A.1 in Appendix A (Section A.3) shows the correlation matrix between all the independent variables that are part of the regression model. For ease of visual reference and comparison, we include in Figure A.13 (also in Section A.3) a graph matrix showing all independent variables and their relationship to each other.
5.3 Data Transformation

Data Analysis

Table 5.2: Box plot of graph hierarchy, graph connectedness, graph efficiency and clustering coefficient

Table 5.3: Box plot of graph hierarchy, graph connectedness, graph efficiency and clustering coefficient after data transformation
<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Independent variable</th>
<th>Statistical test</th>
<th>Result</th>
<th>Decision</th>
<th>Preliminary Finding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>authorcount</td>
<td>Spearman's rank correlation coefficient test</td>
<td>[\text{loc}] ((\rho = 0.74, \ p = 0.00)) [\text{rev}] ((\rho = 0.68, \ p = 0.00))</td>
<td>Accepted</td>
<td>More authors more production</td>
</tr>
<tr>
<td>2</td>
<td>filecount</td>
<td>Spearman's rank correlation coefficient test</td>
<td>[\text{loc}] ((\rho = 0.51, \ p = 0.00)) [\text{rev}] ((\rho = 0.82, \ p = 0.00))</td>
<td>Accepted</td>
<td>More files more production</td>
</tr>
<tr>
<td>3</td>
<td>clustering coefficient</td>
<td>Spearman's rank correlation coefficient test</td>
<td>[\text{loc}] ((\rho = 0.33, \ p = 0.00)) [\text{rev}] ((\rho = 0.38, \ p = 0.00))</td>
<td>Accepted</td>
<td>More clustered more production</td>
</tr>
<tr>
<td>4</td>
<td>graph connectedness</td>
<td>Spearman's rank correlation coefficient test</td>
<td>[\text{loc}] ((\rho = 0.26, \ p = 0.00)) [\text{rev}] ((\rho = 0.29, \ p = 0.00))</td>
<td>Accepted</td>
<td>More connected more production</td>
</tr>
<tr>
<td>5</td>
<td>graph hierarchy</td>
<td>Spearman's rank correlation coefficient test</td>
<td>[\text{loc}] ((\rho = -0.38, \ p = 0.00)) [\text{rev}] ((\rho = -0.34, \ p = 0.00))</td>
<td>Rejected (opposite holds)</td>
<td>More asymmetric less production</td>
</tr>
<tr>
<td>6</td>
<td>graph efficiency</td>
<td>Spearman's rank correlation coefficient test</td>
<td>[\text{loc}] ((\rho = -0.14, \ p = 0.00)) [\text{rev}] ((\rho = -0.13, \ p = 0.00))</td>
<td>Rejected (opposite holds)</td>
<td>More redundant less production</td>
</tr>
</tbody>
</table>

Table 5.4: Summary of Preliminary Findings and Statistical Tests Used
5.4 Multiple-Regression Analysis

In Section 5.2 we tested our hypotheses individually. These tests revealed relationships between the independent and the dependent variables, in some cases in accordance to our hypotheses about their association and in other cases contrary to them. These findings have been summarized in Table 5.4. This table shows for each of the independent variables, the results of applying spearman’s rank correlation coefficient test to find the degree of association to the dependent variables lines of code and revision count. Furthermore, it also shows the preliminary findings and our decision of whether or not to accept our hypotheses based on these findings.

Following from the results obtained, Table 5.7 shows each of the independent variables, whether or not they are associated with the dependent variables. All variables will be included in the multiple regression models. For all independent variables the number of observations was reduced to 195 (from 257) after removing outliers (62 values out of 257).

Regression analysis

In this section we seek to understand the relationship between a response variable, two in our case namely lines of code and revision count, and a set of explanatory variables (filecount, authorcount, graph connectedness, graph hierarchy, graph efficiency, and clustering coefficient). More specifically, we want to know in what sense the explanatory variables influence the response variable(s). For this we will use multiple regression analysis.

Multiple regression analysis tries to find what the distribution is of a response variable $Y$, conditional on the values of a set of explanatory variables referred to as the X’s (Fox, 2008) (pg. 14). When there is more than one X value, regression analysis is referred to as multiple-regression analysis. Linear regression analysis restricts the relationships sought between the X variables to be linear relationships. The focus of linear regression analysis is on the expectation, or the mean of $Y$ as a function of the X’s (5.5). This is the probability density of $Y$ conditional on a set of specific values of the X’s and is written as shown below in Equation (3).

$$p(y \mid x_1, x_2, ..., x_k)$$ (5.3)

Typical assumptions in regression analysis are that $p(y \mid x_1, x_2, ..., x_k)$ will be normally distributed, there is constant variance regardless of the values of the X’s, as shown below in Equation (4), and the expected value of $Y$ is a linear combination of the X’s, as shown below in Equation (5).

$$V(Y \mid x_1, x_2, ..., x_k) = \sigma^2$$ (5.4)

$$E(Y \mid x_1, x_2, ..., x_k) = \alpha + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k$$ (5.5)

Possible violations of the assumptions for regression analysis include: skewness, multiple modes, heavy tails, unequal variances and nonlinearity (Fox,
For this reason we first examined and then transformed the data in sections 5.2 and 5.3 (including the removal of outliers). The data is now suitable for linear regression analysis.

**Explaining the models**

For our regression models we used forward/backward, or stepwise, selection methods. These methods of variable selection combine two approaches, the bottom-up approach and the top-down approach which dictate the order in which variables are introduced into the model during regression (Fox, 2008).

The models that will be presented below are models for the dependent variables *lines of code* and *revision count*. The dependent variables have been transformed, as explained in Section 5.3. The two models make use of the same independent variables, which were also transformed. The reason to include models for both dependent variables is that they enable reflection on different aspects of code production. This will be explained in more detail in Chapter 6.

### 5.4.1 First model: explaining lines of code

**Stepwise Regression using log(Lines of Code) (loc_log) as the dependent variable**

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>$\beta$</th>
<th>(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Independent variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File count</td>
<td>.040</td>
<td>9.01***</td>
</tr>
<tr>
<td>Author count</td>
<td>.528</td>
<td>3.04**</td>
</tr>
<tr>
<td>Graph connectedness</td>
<td>-.320</td>
<td>-.82</td>
</tr>
<tr>
<td>Graph hierarchy</td>
<td>-1.40</td>
<td>-1.63</td>
</tr>
<tr>
<td>Graph efficiency</td>
<td>.406</td>
<td>0.82</td>
</tr>
<tr>
<td>Clustering coefficient</td>
<td>2.582</td>
<td>2.14**</td>
</tr>
</tbody>
</table>

| R$^2$               | .561     |
| Adjusted R$^2$      | .547     |
| F-value             | 40.05    |

†, p < .10; *, p < .05; **, p < .01; ***, p < .001; N = 195

**Table 5.5: Regression Model with log(Lines of Code) (loc_log) as the dependent variable.**

This is our transformed regression model. As was explained, this model includes all of the transformed variables. The model is significant, it has a highly significant F, all the $\beta$’s are significant as are their corresponding t-values. The model explains 54% of the variation and the signs are in the expected direction, meaning that clustering coefficient, file count and author count positively influence code production measured as lines of code. Importantly, file count has the strongest influence, followed by author count and then by clustering coefficient. However, for these last two variables the difference in contribution is not
so substantial. The order of influence means that code production measured as *lines of code* is mostly dependent on *file count*, and then almost equally on *author count* and *clustering coefficient*. What this exactly means will be explained in Chapter 6.

All the variables for hierarchy, namely *graph connectedness*, *graph hierarchy* and *graph efficiency*, were dropped from the model, which means they do not significantly explain any of the variation present in the data. This will also be interpreted in Chapter 6.

### 5.4.2 Second model: explaining revision count

**Stepwise Regression using** $\sqrt{\text{Revision Count}}$ $(\text{rev}_{\text{sqrt}})$ **as the dependent variable**

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>$\beta$</th>
<th>(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision count</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Independent variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File count</td>
<td>.168</td>
<td>6.34***</td>
</tr>
<tr>
<td>Author count</td>
<td>12.57</td>
<td>12.21***</td>
</tr>
<tr>
<td>Graph connectedness</td>
<td>-.470</td>
<td>-.20</td>
</tr>
<tr>
<td>Graph hierarchy</td>
<td>-.274</td>
<td>-.54</td>
</tr>
<tr>
<td>Graph efficiency</td>
<td>-.154</td>
<td>-.05</td>
</tr>
<tr>
<td>Clustering coefficient</td>
<td>23.16</td>
<td>3.23***</td>
</tr>
<tr>
<td><strong>$R^2$</strong></td>
<td>.736</td>
<td></td>
</tr>
<tr>
<td><strong>Adjusted $R^2$</strong></td>
<td>.728</td>
<td></td>
</tr>
<tr>
<td><strong>F-value</strong></td>
<td>87.55</td>
<td></td>
</tr>
</tbody>
</table>

$\dagger$, $p < .10$; $*$, $p < .05$; $**$, $p < .01$; $***$, $p < .001$; $N = 195$

Table 5.6: Regression Model with $\sqrt{\text{Revision Count}}$ $(\text{rev}_{\text{sqrt}})$ as the dependent variable.

This is our second transformed regression model. As was explained, this model also includes all of the transformed variables. The model is significant, it has a highly significant F, all the $\beta$’s are significant as are their corresponding $t$–values. The model explains more of the variation in the data, namely 73% of the variation and the signs are in the expected direction, meaning that *clustering coefficient*, *file count* and *author count* positively influence code production measured as *revision count*. Importantly, there is a change of roles in the order of influence in the variables. In this model *author count* has the strongest influence, followed by *file count* and then by *clustering coefficient*. However, again for this model, the last two variables do not display a substantial difference in contribution. The order of influence means that code production measured as *revision count* is mostly dependent on *author count*, and then almost equally on *file count* and *clustering coefficient*. What this exactly means will be explained in Chapter 6.

Again, all the variables for hierarchy, namely *graph connectedness*, *graph hierarchy* and *graph efficiency*, were dropped from the model, which means
Summary of the outcomes of multiple-regression analysis

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Log(Lines of Code) as dependent variable</th>
<th>Sqrt(Revision count) as dependent variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pooled 2004-2009</td>
<td>$F = 0.00$ and $R^2 = 0.547$</td>
<td>$F = 0.00$ and $R^2 = 0.728$</td>
</tr>
</tbody>
</table>

Table 5.7: Summary table of the outcome of multiple-regression analysis

they do not significantly explain any of the variation present in the data. This will also be interpreted in Chapter 6.

5.4.3 Alternative model specifications

In this Section we address alternative model specifications. These alternative specifications are meant to show the reader the composition of additional models different than those described previously. These are models that we tested and for which we did not find different results. For this reason, we will keep the description short and refer the reader to results shown in Appendices.

Multiple regression on untransformed variables

For completeness we include, in Appendix A (Section A.3.1 and Section A.3.2), the outputs of multiple regression analysis on untransformed variables. As can be observed in Table A.3.1 and Table A.3.2, the results of multiple regression analysis are not consistent. In Table A.3.1 the variable graph hierarchy shows up as significant while clustering coefficient does not. Furthermore, author count does not show up as significant either. The results shown in Table A.3.2 are more consistent with those presented in Section 5.4. However, after data transformation the relationships are strengthened, results are consistent across different models and the explanatory power of the models increases. Therefore, we rely on the multiple regression analyses that were presented in Section 5.4.1 and Section 5.4.2.

Multiple regression on transformed but discarded variables

For completeness we include, in Appendix A (Section A.3.3 and Section A.3.4), the outputs of multiple regression analysis on transformed independent variables and unified and transformed dependent variables $\log(\text{lines of code per author})$ and $\sqrt{\text{revision count per author}}$. As can be observed in Table A.3.3 and Table A.3.4, the results of multiple regression analysis are consistent but $\beta$’s and their corresponding $t$ – values values are less significant. Furthermore, both models explain less of the variation, respectively with an adjusted $R^2$ of .380 and .322, in comparison to the models that were presented for which we respectively obtained an adjusted $R^2$ of .547 and .728.
In addition, the inclusion of *author count* into the dependent variable, although enabling a reflection about productivity (i.e., capital resources per unit labor), interferes with the elegance and simplicity of our models. For that reason our models were not extended to include analyses of productivity. Furthermore, the inclusion of *author count* into the dependent variable, obscures the comparison of *factors of production* and *factors of self-organization*. This comparison is deemed useful because it opens up the possibility for reflection about the relative importance of *factors of self-organization* in comparison to *factors of production*. As will be explained in Chapter 6, this comparison differs for the dependent variables *lines of code* and *revision count*. This difference provides further insights into code production.
Chapter 6

Conclusions

In this chapter we draw conclusions from our research on code production in top-level open source communities of The Apache Software Foundation (ASF). We extensively analyzed Subversion repository logs from 70 top-level Apache open source projects in the ASF from 2004 to 2009. Based on interactions in code production during one-year periods we constructed networks of file co-authorship that gave us access to the organization of Apache open source communities. This allowed us to measure graph level properties and their influence on the outputs of code production.

Apache communities are groups of individuals that organize their code production efforts in order to develop enterprise grade open source software. The ASF explains the success of its communities and the software they produce by claiming to have instituted a meritocracy that brings contributors together in a way that significantly influences code production, namely by building communities instead of only focusing on technical properties of the source code like modularity. Apache communities share an institutional infrastructure that is embodied in the principle of meritocracy, a principle which states that:

‘The more you do, the more you are allowed to do.’

Apache open source communities are scientifically and managerially interesting because they seem to be at the center of a debate about how open source communities are organized. In literature there are two opposing sides to this debate from the research streams of self-organization and institutionalization. These research streams contribute bottom-up and top-down explanations of organization that focus on how coordination is achieved in the communities. A representative institution in this debate is hierarchy, which self-organization argues is irrelevant while institutionalization claims to be relevant.

The debate is further complicated by the nature of meritocracy. Such principle seems to result in an emergent hierarchy that is continuously re-conceptualized and interpreted into a reputation system which grants allowances in terms of power and control. This means that emergent hierarchy would be constantly formalized into a role-based organizational structure, similar to what has been called a ‘voluntary hierarchy’ (Weber, 2004). One way to untangle this complex debate is to understand how open source communities are organized.
through an organizational model of open source. Such a model was developed by Van Wendel de Joode (2005).

Van Wendel de Joode (2005) found that the role of institutions is minor when compared to individual behavior. In line with the debate centered on institutions, like hierarchy, he found that the ASF appears to be an exception to this finding (of a minor institutional role) primarily because the ASF foundation is an institution that increases the external recognition of the communities while protecting them from outside pressures. This exception is apparently related to a proposition about institutions and their limited external influence and was put forth as a hypothesis that could lay the basis for future research (Van Wendel de Joode, 2005) (pg. 214).

In this research we adopted the proposition of Van Wendel de Joode (2005) to test if meritocracy has internal influence on the organization of open source communities of the ASF. We phrased our main research question as follows:

**Does meritocracy influence code production?**

According to the limited external role of institutions, meritocracy would not be expected to have an influence on the output of code production in Apache communities. In this research code production is believed to provide a measurable understanding of how open source communities are organized. We theorized that such understanding could be obtained from the influence of measurements of graph properties of organization on the outputs of code production. More specifically, we measured hierarchy and clustering in networks of file co-authorship in order to determine the extent of influence each has on the outputs of code production measured as lines of code and revision count.

According to the ASF, ‘the more you do, the more you are allowed to do’, roles matter and are assigned by merit, and institutions have internal influence. We decided to test these claims along with the theory of self-organization of Van Wendel de Joode (2005) by focusing on factors of production, factors of organization and the outputs of code production in Apache communities. In other words, explanations from theory and practice appear to be opposites and through its empirical approach this research aims at making a meaningful inference taking practice as rival theory. This implies that in this research we are testing the theory of self-organization of Van Wendel de Joode (2005) and at the same time we are testing claims that come from practice and which are rival to such theory.

### 6.1 Research model

Our research model is based on the organizational model of open source developed by (Van Wendel de Joode, 2005). Van Wendel de Joode (2005) adopted a framework from research on community-managed common pool resources, which explains how more traditional communities, such as huertas and communities sharing water resources, have managed to self-organize around institutions. Based on this framework he created an organizational model of open source “(...) to serve as input to discussion and reflection and as a guide for
6.1 Research model

Further research on the subject” (Van Wendel de Joode, 2005) (pg. 37). The model consists of eight design principles that reconcile the two main strands of research on the organization of open source communities. Such strands correspond to two extreme views on organization, namely self-organized anarchies and institutionalized communities (Van Wendel de Joode, 2005). We briefly discuss the most relevant points from these opposing streams of research below.

6.1.1 Opposing streams of research

Much recent research argues that open source communities are self-organizing systems (Raymond (1999); Kuwabara (2000); Axelrod and Cohen (2001); Madey (2002); Van Wendel de Joode (2005); Robles et al. (2005); Heylighen (2007)). A common thread in self-organization literature is focused on explaining how coordination is achieved in spite of the lacking influence of collective institutions. There is much agreement among proponents of self-organization regarding the irrelevance of hierarchy as a centralized coordination instrument in open source communities ((Bonaccorsi and Rossi, 2003); (Garzarelli et al., 2002); (Raymond, 1999)). However, there are various organizational aspects of coordination that must be fulfilled somehow. Mostly these aspects are fulfilled by clustering, a bottom-up process of organization that is brought about by modularity where clusters of developers arise as a result of splitting up complex software into different activities (Van Wendel de Joode, 2005) (pg. 134).

An opposing stream of organizational research in open source is spearheaded by researchers who explain organization through the institutionalization of collective mechanisms that have been found to be instrumental in coordination ((Weber, 2004); (Kogut and Metiu, 2001b); (Lerner and Tirole, 2002); (O’Mahony and Ferraro, 2007); (O’Mahony and West, 2008)). Many researchers in this stream often criticize explanations based on self-organization. They attribute a more significant role to hierarchy and favor explanations that show how open source communities organize around institutions.

6.1.2 A model of code production

Based on the research streams of institutionalization and self-organization we constructed a research model of code production. This model is shown below in Figure 6.1. The model shows three of the design principles from the organizational model of open source (Van Wendel de Joode, 2005), as top-down instruments that influence organization. It also shows the bottom-up influence of clustering, a mechanism which as discussed is the result of modularity. Both instruments and mechanism are believed to comprise the factors of organization. Additionally, the model shows the influence of labor and capital through factors of production. Namely these are the number of files and the number of authors found to be involved in code production.
6.2 Main findings

6.2.1 Meritocracy does not influence code production

The main question of our research, as discussed previously, reads as follows.

Does meritocracy influence code production?

This main research question was divided into two research sub-questions that refer to the main theories of organization in open source, namely self-organization and institutionalization. The sub-questions read as follows.

1. Does modularity influence code production?
2. Does hierarchy influence code production?

Based on theory, there was an expectation that both modularity and hierarchy would have an influence on code production. The hypotheses that were formulated for the concepts of modularity and hierarchy were based on operationalizations of these concepts found in literature, namely clustering coefficient, graph connectedness, graph hierarchy and graph efficiency. The last three constructs are related to institutions conceptualized in literature and reported to have an influence in organization. Thus, our expectation was that these constructs would also have an influence in code production. Next we answer the sub-questions of this research.
Conclusions

6.2 Main findings

Hierarchy does not influence code production

Based on empirical analysis and the findings therein, we can answer the main research question and sub-questions. We will start with the main research question and the sub-question that refers to hierarchy. In the next section we will discuss exactly what these findings mean by examining their relationship to the design principles from the organizational model of open source (Van Wendel de Joode, 2005).

Based on the theory of institutionalization in open source, it was expected that meritocracy would have an influence on code production. As explained, different aspects of meritocracy were identified in institutions found in literature. The expectation was that at least one of these institutions would have an influence on code production, measured as lines of code or revision count. The expectations were not supported in this research. Broadly, meritocracy was not found to have any influence on code production. More specifically, the dimensions of meritocracy were not found to have an influence on the outcomes of code production, measured as lines of code, nor on the outcomes measured as revision count.

Modularity does influence code production

Next we provide an answer to the research question that refers to modularity.

Based on the theory of self-organization in open source, it was expected that modularity would have an influence on code production. As explained, one of the consequences of modularity is clustering. The expectation was that clustering would have an influence on code production, measured as lines of code or revision count. These expectations were supported in this research for both outcomes of code production. With similar and reinforcing results, clustering coefficient was found to have an influence on the outcomes of code production, measured as lines of code and as revision count.

Answer to the research question

Having answered the research sub-questions we are now able to provide an answer to the main research question.

Based on the theories of self-organization and institutionalization in open source, it was expected that modularity and hierarchy would have an influence on code production. As explained, one of the consequences of modularity is clustering. The expectation was that clustering would have an influence on code production, measured as lines of code or revision count. And that hierarchy would have an influence on code production, measured as lines of code or revision count. Only the expectations with regards to clustering were supported in this research for both outcomes of code production. With
similar and reinforcing results, clustering coefficient was found to have an influence on the outcomes of code production, measured as lines of code and as revision count.

Open source communities are self-organizing

Our finding that clustering has predominant influence on code production, over that of hierarchy, confirms the expectations found in the theory of self-organization. The predominant influence of clustering over hierarchy gives insights into the non-hierarchical organization of open source communities in relation to their sustained code production. In other words, since organization can be understood through code production and because clustering was found to play a predominant role over hierarchy we conclude that open source communities are predominantly self-organizing.

In this research we confirm the theory of self-organization of Van Wendel de Joode (2005) through the test of a case that was considered critical and which was found to be an exception.

6.3 Examination of design principles

In this section we will examine the answers to the research question and sub-questions in relation to the design principles from the organizational model of open source (Van Wendel de Joode, 2005). Our intention here is to reflect on the answers by linking them to the design principles of the organizational model of open source that were chosen for this research. We find that this approach allows us to reflect on the meaning of the results of this research.

6.3.1 Conflict resolution mechanisms

Our examination of the design principle of conflict resolution is based on the finding in Van Wendel de Joode (2005) that conflicts are managed by creating modular software. Based on our results we believe that modularization includes the creation of parallel lines of development. As explained in Van Wendel de Joode (2005), modular software allows each module to perform a limited set of tasks which individuals can undertake autonomously. Our results indicate that clustering, author count and file count predominantly influence the outputs of code production. The data we collected in order to conduct this test of influence, includes all modules and parallel lines of development for a given open source project for a period of one year.

Based on our results we believe that modular code production includes the creation of parallel lines of development. Modularization through parallelization of code is beneficial because it allows contributors to tackle activities independently of other individuals. This important for some parts of a code base, for instance those where conflict is located, but for other parts of the codebase the opposite, and even a combined, effect is vital. In other words, for parts of
the codebase where (sub-)communities are in the process of emerging, or have emerged, it is important to encourage contributors to tackle activities in concert (i.e. to co-author more intensively) but independently of other developers.

Either in the presence or absence of conflict, or somewhere in between, coding activities may result in increased quality, for example through the elegance of code being produced or through the inclusion of code in a release version of the software. This leads to positive feedback and results in the formation of clusters of authorship (Ghosh, 2004), which are also referred to as swarms (Van Wendel de Joode, 2005) (and which we simply call clusters). Therefore, we believe that clusters emerge out of parallel lines of development similar to how it has been reported to happen for modular code (Van Wendel de Joode, 2005). In fact, we consider parallel lines of development to be an extreme form of modularization.

The emergence of a cluster from a parallel line of development (a sub-project) in the Apache Hadoop project is illustrated in Figure 6.2 below. As shown in the Figure, the Zookeeper community cluster is completely parallel from the main Hadoop community.

![ZooKeeper](hadoop2008.xml)

Figure 6.2: Apache Hadoop community and sub-communities in 2008

**Relevance and connection of our findings**

*In terms of the design principle of conflict resolution mechanisms, what our findings mean is that clustering is as much a phenomenon*
that emerges from modular code, as it is one which emerges from parallel lines of development. Our belief is based on the fact that the data we collected includes both modular and parallel lines of development in a codebase, and therefore the results we obtained reflect both as one (clustering). We consider parallel lines of development to be an extreme form of modularization.

6.3.2 Collective choice arrangements

Our examination of the design principle of collective choice is based on the finding in Van Wendel de Joode (2005) that voting systems have weak influence in the communities. Based on the results of this research we find that tags such as reputation result in positive feedback which in turn leads to the formation of clusters of authorship (Ghosh, 2004). As described by Van Wendel de Joode (2005), the process starts with increasing levels of activity and reputation which lead to the swarming of developers, or the formation of clusters of authorship that emerge around code. Code which is elegant is often produced by individuals with a high level of reputation and surrounded with high levels of activity, and in most cases included as part of a release version. Therefore, we believe that clusters emerge primarily around the tag of level of reputation of a contributor (Van Wendel de Joode, 2005).

Van Wendel de Joode (2005) describes five different kinds of tags that lead to clustering, and proposes that a combination of tags might be necessary for new projects to become popular (pg. 202). The empirical data in this research provide evidence regarding the role of reputation and level of activity as tags. As illustrated in Figure 6.3, in the Apache HTTP Server community (during 2009) there are some contributors with a higher level of reputation than others. In Figure 6.3a this is illustrated by the size of individual nodes, which is determined by the number of contributors that have co-authored files with the contributor represented by a particular node and by the weight of individual contributions measured as lines of code. In Figure 6.3b the community has been grouped to show more clearly the differences in reputation represented by the size of nodes.

Relevance and connection of our findings

In terms of the design principle of collective choice arrangements, what our findings mean is that clustering emerges from the level of reputation of developers. Our belief is based on the fact that the data we collected includes the number of lines of code and the revision count per author, for all contributors in an open source project, allowing us to determine the reputation of individuals in a cluster based on the number of co-authors (incoming). And when observed in a graph, clusters seem to form around contributors with high indegree centrality, which in social network analysis literature is equated with reputation (Wasserman and Faust, 1994) (pg. 179)
6.3 Examination of design principles

(a) Apache HTTP Server community

(b) Apache HTTP Server community

Figure 6.3: Apache HTTP Server community in 2009

6.3.3 Multiple layers of nested enterprise

Our examination of this design principle is based on the presence of diversity and interdependencies in code production. Van Wendel de Joode (2005) observes that open source communities, including Apache communities, produce modular software whose modularization leads to interdependencies between modules and therefore between contributors, and that these interdependencies must be managed. A series of observations are put forth regarding this design principle, including

- i) a highly developed division of labor, that is
- ii) emergent and
- iii) results in high degrees of task specialization, which
- iv) raises the level of efficiency in the communities and
- v) constitutes a learning environment for participants (Van Wendel de Joode, 2005) (pgs. 204-205).

Based on the results of this research, we confirm that modularization results in a highly developed division of labor. In this research we measure clustering and its influence on the outputs of code production. And since we consider clustering as a bottom-up phenomenon that is brought about by modularity (modularization really), we agree that the division of labor is emergent because we found that clustering influences code production. Furthermore, we concur that there are high degrees of task specialization. This is illustrated in Figure 6.4 below, where tasks are assigned different colors, and nodes different sizes depending on co-authorship and the number of lines of code that were co-authored. In addition, the emergent division of labor is apparent in Figure 6.4, and its shape suggests the increased level of efficiency which results in the creation of sub-communities (Hive inside the red oval, Pig inside the yellow oval, and HBase inside the green oval).

Relevance and connection of our findings

In terms of the design principle of multiple layers of nested enterprise, what our findings mean is that modularization results in
a highly developed division of labor. Our belief is based on the fact that the data we collected, particularly for larger Apache open source projects, shows the presence of sub-communities that have emerged as highly specialized sub-projects (in some cases even resulting in spin-off projects). This suggests an increased level of efficiency in the communities.

Figure 6.4: Apache Hadoop community and sub-communities in 2009

6.3.4 Comparison to results from previous research

This research finds support for the theory of self-organization in open source communities of Van Wendel de Joode (2005). One of the main findings in the formulation of such theory, is that “(...) the role of institutions is minor compared to individual behavior (...)” (Van Wendel de Joode, 2005) (pg. 207). In this research we tested whether institutions play a role in code production, in comparison to individual behavior observed in factors of production and factors of self-organization.

The theory of self-organization in open source communities of Van Wendel de Joode (2005) puts forth a series of propositions. One of those propositions was adopted in this research and used to formulate the main research question. The proposition reads as follows.
6.4 Discussion of results

The institutions behind meritocracy, found in the theory of institutionalization in open source, do not seem to play a significant internal role. Based on empirical findings, such institutions, which were operationalized as measures of hierarchy in a network, do not have an influence on the outcomes of code production. Code is the main product of open source communities, and measuring influence in code production is believed to give insights into how open source communities are organized. Furthermore, code production shows how open source communities sustain their efforts. Therefore, code production appears to be the primary internal activity in open source communities. However, no influence of institutions was found on such internal activity.

6.4.1 Alternative explanations

The results of this research confirm that the role of institutions is minor when compared to individual behavior. But, could this result be found while it still being the case that institutions are important? First of all we wish to clarify that we agree with Van Wendel de Joode (2005) that the role of the ASF is not marginal, that it performs the important function of protecting contributors from future legal claims. Furthermore, as described in Section 6.3.4, our results confirm the proposition put forth in Van Wendel de Joode (2005).

We do not believe the same results would be obtainable should institutions also play a significant internal role. The reason is that in such case, when compared to individual behavior, institutions would result in predictable rule-following behavior that could be readily observed and found to influence code production. However, we did not find this despite testing various alternate models and transforming the variables in the models in necessary ways. This leads us to believe, and confirm, the minor role of institutions compared to individual behavior.

Lastly, we would like to note that if the reader takes away from this research that institutions are not important, it could still be the case that institutions are relevant in an external way, as previous research has demonstrated (Van Wendel de Joode, 2005).
6.4.2 Limitations of this research

As mentioned previously, data pre-processing could potentially be a limitation of this research in that it could lead to distorted results. Pre-processing is an important step in this research because it determines the construction of the networks where measurements are conducted. Therefore, should a different construction method be used, the results could potentially vary because they would relate to aspects that were not measured (which would be found in the alternative networks). For this reason it would be worthwhile to confirm the results of this research by replicated studies that focus on other open source communities.

Another limitation of this research is the focus on a period of one year for data collection. In communities with higher code production there are many more interactions in a shorter period of time. Such communities become highly connected before other communities. By the time one year of code production has transpired, such communities have become fully connected, and this potentially gives rise to outliers. Furthermore, some communities are more complex than others and deserve to be dissected more closely. One way of inspecting such communities in more detail is to only consider certain code branches, or to focus on individual sub-communities. Therefore, the consequence of focusing on a period of one year for data collection is that we are not able to measure the adequate networks in some cases.

Additionally, there are a variety of measures that can be used to gain insights on code production. Two specific types of measures that are relevant are measures that refer to software quality and organizational complexity (Nagappan et al., 2008). Of the measures of software quality, we find code churn, file status, code complexity to be relevant (measures reviewed in (Nagappan et al., 2008)). And of the measures of organizational complexity, we find number of ex-engineers and edit frequency to be relevant (measures proposed in Nagappan et al. (2008)).

6.4.3 Generalizability of results

The results of this research are very particular, they refer to open source communities in the ASF. In that sense, the results are not generalizable. However, the ASF is a critical case for the theory of self-organization of Van Wendel de Joode (2005). We believe that if institutions do not influence code production for ASF communities, then it probably is not the case in the rest of open source. In that sense, the results of this research are highly generalizable. In order to sort out this dilemma, we propose in Section 6.6 to replicate the analyses for all ASF communities over time. This will enable the study of evolution of open source communities in the ASF. In addition, in Section 6.6 we also propose to replicate the analyzes on project hosting sites, such as those in Google Code, SourceForge, and other open source communities, such as those in the Python Software Foundation and the Eclipse Software Foundation.
Generalizability of methods

Despite limitations, the results of this research are also highly suitable for generalization. First, the method of data collection and pre-processing used in this research applies to any open source community that makes use of a Subversion repository. Other than CVS, BitKeeper, and GitHub, Subversion is the primary source code repository used in open source communities, including those in the Python Software Foundation\textsuperscript{77} and the Eclipse Software Foundation.\textsuperscript{78} In addition, communities that rely on CVS, like the FreeBSD community, are rapidly migrating their source to Subversion repositories, which are already available online.\textsuperscript{79} Other major communities like Debian also use Subversion.\textsuperscript{80} In addition, project sites like Google Code\textsuperscript{81} and SourceForge\textsuperscript{82} host all of their projects on Subversion.

Second, another reason why the results of this research are highly generalizable is that the method of data collection and pre-processing can be applied on any time period. This means that code production for any period of time can be studied. Therefore, the time dimension is at the disposal of the researcher. Furthermore, the code dimension is also configurable. The tools for data collection and pre-processing can be easily configured to only collect data for a given branch of an open source codebase, and the granularity stretches down to the file level.

6.5 Implications and Reflection

This section focuses on two related aspects. First, we discuss the implications of this research in relation to its results and how they can be understood and used. Second, we include reflections useful for organizations interested in the community type of (software) development. These reflections focus on what our research means for organizations that want to organize work more in the open source way.

6.5.1 Implications

The results of this research are potentially controversial. Major companies sponsor the ASF, in part because of the belief that the foundation can purposefully design open source communities by making them abide to organizational principles like meritocracy. The ASF incubator is a clear example of this role of the ASF. The Apache incubator is a top-level project that is charged with adopting external projects, indoctrinating them into the ASF and developing a community around them. In this research, meritocracy was not found to influence code production which would directly reflect a lack of organizational influence of such institution. Another reason why this research is potentially controversial is that it can be used to investigate the extent to which there is corporate control of open source projects. In other words, this research can be used to investigate what share of meritocracy sponsoring companies have in the ASF.
On the other hand, the results of this research are potentially the basis for an improved understanding of *The Apache Way*. This would require extending this research and finding application areas where it might be useful. One example where this might be the case is converting this research project into an Apache Lab. Should the analyses proposed in this research be adopted into an Apache Lab, a space for innovation and experimentation within the ASF, this research could serve as the foundation for active monitoring of Apache projects and communities. An application that can be envisioned is a management information system useful for decision-making on the evolution of incubator projects, including when such projects should be allowed to graduate from the Apache incubator.

Another application is as a recommendation system employing technologies developed in Apache projects like Apache Mahout, a scalable machine learning library that supports large data sets. Such a system would recommend files to developers based on their file co-authorship behavior. The adoption of one or both of the proposed applications would imply that this research can be used to create vibrant, better-organized and highly sustainable open source communities. These ideas go beyond current organizational practices employed in open source and are believed to have the potential to revolutionize community-building efforts.

### 6.5.2 Reflection

Organizations who are interested in community-based production can consider implementing the findings of this research. We consider it beneficial for organizations to consider our findings in light of the examination of design principles from the organizational model of open source (Van Wendel de Joode, 2005). In other words, we believe that an examination of the design principles brings out the relevance of our findings for organizations who want to organize work in the open source way.

The first point for reflection is on the modularization a software codebase. We believe this requires careful contemplation, since the intention should be to leave room for bottom-up modularization. In other words, we do not advocate taking a planning approach towards modularization. Instead, we support the mindful adaptation of a software system’s architecture (also called architectural adaptation (Taylor et al., 2009)) to the bottom-up changes that emerge. The adapted (organizational) architecture would be of similar character to a multiagent plan used for high level coordination (Russell and Norvig, 2010).

Second, based on our results we believe organizations should encourage the creation of parallel lines of development for which a sub-community emerges. In other words, we consider a parallel line of development as a useful effort so long as it leads to increased co-authorship. This may be because a parallel line of development is used to deflect or alleviate conflict (Van Wendel de Joode, 2005) or because it represents a sub-project that needs to be embedded in the codebase of a larger project, yet also needs some level of independence. In any case, a parallel line of development should be an effort geared towards the formation of a community cluster.
Third, our results indicate that reputation leads to the formation of clusters. Reputation can be observed in a file co-authorship network based on the number of incoming links to an individual contributor. Having observed clusters around contributors of high repute, we believe organizations should encourage contributors to build a reputation in the community regardless of whether they are employees or volunteers. In other words, the focus should be on community-based production not on employees of a particular organization.

Fourth, based on our results we believe that modularization leads to a highly developed division of labor. We believe that organizations that encourage modularization will reap the various benefits, such as: emergence, innovation, efficiency, and specialization.

Fifth, based on our results we believe that it is much more important to focus on modularization of the code base instead of synthetically building a community by instituting rules or practices. In other words, we believe less time should be spent on artificially building a community or institutions (rules and practices), instead the focus should be on making sure the project is modular and then giving it room to grow.

Lastly, our results indicate that the use of software tools, like the those developed as part of this research, can be used to understand corporate influence on an open source project, monitor community development, and monitor the role and level of reputation of individual contributors. We believe these are useful tools for recruitment and for procuring an increased understanding of organization in open source.

6.6 Future research

Various directions for future research have been envisioned. These directions are focused on academic research, industrial research, (open source) technology applications, and industrial applications. The most relevant ones are described below.

6.6.1 Future research

Evolution of open source communities

In this research we collected data for code production in open source communities for (fixed) one-year periods. However, some communities quickly become densely connected through file co-authorship interactions. For this reason, it would be beneficial to focus on shorter periods of time in code production and to unite the collected data in order to enable analyses over time. One advantage would be a richer understanding of organization, provided by the evolution of open source communities. In other words, we believe the analyses of this research can be extended over time by employing incremental data collection. Such type of data would allow research on the patterns of evolution in open source communities which are in many ways connected to the notion of hierarchy, such as preferential attachment and power-law growth. Many researchers
have studied evolution in open source using email to represent organization ((Weiss et al., 2006); (Crowston et al., 2006); (Valverde et al., 2006); (Valverde and Solé, 2007)). We are critical of this approach since ‘sending emails’ is not what open source developers do. What they do is ‘write code.’ Therefore it seems more appropriate to study code production rather than other aspects. We believe the tools developed as part of this research could be extended to cater for future research in this direction.

**How do open source communities differ from online social networks?**

Van Wendel de Joode (2005) proposes that an interesting line of research would be one that compares open source communities with other types of organizations. He asks, ‘What organizations are comparable?’ In this research we used social networks as a representation of open source communities and employed graph level measures to study the influence of organizational phenomena on the outputs of code production. This leads us to believe that other social networks can be studied in a similar way. A direction that seems interesting in this respect is to study the influence of organizational phenomena in online social networks like Twitter, Facebook and Hyves, on the outputs of social media production in these platforms. In other words, if we think of pictures (or other media that lead to connections) among friends in a social network as the source code files that connect developers in open source communities, and if we count the number of clicks on advertisements (the so-called click-through-rate), then we could measure the influence of organizational phenomena on the success of an online advertising campaign. It might also be possible to extend this idea into organizations represented as links between webpages.

**Comparison to open source communities in the Python Software Foundation and Eclipse Software Foundation**

In Section 6.4.3 we noted our belief that if institutions do not influence code production for ASF communities, then it probably is not the case in the rest of open source. An interesting strand of future research would be to replicate our analyses on open source communities of the Python Software Foundation (PSF) and the Eclipse Software Foundation (ESF). We have started exploring this possibility, and have collected Subversion repository logs for all open source communities in the PSF and ESF. The data can be found online at: [http://people.apache.org/~ocastaneda/eclipse.org](http://people.apache.org/~ocastaneda/eclipse.org) and [http://people.apache.org/~ocastaneda/python.org](http://people.apache.org/~ocastaneda/python.org).

**Comparison to open source communities in project hosting sites like Google Code and SourceForge**

In Section 6.4.3 we noted our belief that if institutions do not influence code production for ASF communities, then it probably is not the case in the rest
of open source. An interesting strand of future research would be to replicate our analyses on open source projects in hosting sites like Google Code or SourceForge. We have started exploring this possibility, and written a script that collects Subversion repository logs for the top open source projects in SourceForge.net. However, there is a limiting mechanism for the amount of data we can collect and therefore face that limitation at the moment. But that would be part of the challenge in this strand of research. The data we have collected thus far can be found online at: http://people.apache.org/~ocastaneda/sourceforge.net.

Open source corporate ventures

In combination with organizational research on open source communities, we believe an interesting direction for future research would be to explore open source incubators and evaluate their potential as tools for strategic renewal and innovation within companies.\(^{85}\) The aim of such research would be to provide practical insights on how to effectively benefit from continued participation in open source. In other words, exploring the possibilities for corporate venturing programs to leverage open source technologies in a framework of open innovation, and how that might result in strategic renewal and possible improvements to the efforts of managing innovation.

6.6.2 Future research on methods and techniques

Semantic Web Sociology

Semantic Web Sociology (SWS) is an application of Web Science aimed at developing an understanding of social and technological concerns that are prevalent in virtual organizations embedded on the Web.\(^{86}\) The idea behind SWS applied to open source communities is that Subversion repository logs can be converted into metadata formatted in accordance to the Resource Description Framework (RDF).\(^{87}\) Once data is formatted in RDF, it is possible to make statements about resources that represent relationships in the form of subject-predicate-object expressions, such as “developer A co-authored file B with developer C.” Furthermore, RDF formatted metadata can be used by computers to artificially reason about the distribution of resources on the Web. Inferences could be useful to connect developers by suggesting files to co-author based on prior observed behavior (as described in the previous Section (6.5.1)).

Extensions to SVNPlot

Currently, SVNPlot is able to collect and pre-process Subversion repositories. However, it is not yet able to perform social network analysis of the data it collects, despite the availability of software libraries for social network analysis.\(^{88}\) SVNPlot is written in Python and there are libraries available for Python to access the functionality of the statistical package R, that would enable SVNPlot
to perform social network analysis on the data it collects from Subversion. The idea would be to build a monitoring system for Subversion repositories that would enable open source developers to get an updated view into the organization of an open source community, visually through SNA network graphs and through measures like those used in this research.

Apache Labs

Proposing an Apache Lab that employs Apache technologies, such as Apache Hadoop and Apache Mahout for highly scalable data mining and machine learning of Subversion repository data. The idea would be to build a machine learning application that can recognize patterns and evolve behaviors based on empirical data about file co-authorship. An application would be to recognize complex patterns in code production and make intelligent decisions based on behavioral data, such as producing recommendations, automating specific functions, or involving contributors when considered useful based on past experience.
Notes
1 Subversion - Enterprise-class centralized version control for the masses
http://subversion.apache.org/ (October 2010)
2 Netcraft web server survey - August 2010
3 From an article on the Internet, “31 software defects in 58,944 lines of source code, yielding a defect density of .53 per 1,000 lines of source code.” http://www.infoworld.com/d/developer-world/open-source-code-quality-endorsed-291 (August 2010)
4 For instance, Mockus et al. (2000) showed how quality is ensured by contributors in the Apache HTTP server.
5 Thanks to Sally Khudairi for pointing this out during the Media & Analyst Training at ApacheCon Europe 2009 in Amsterdam.
6 For example, Matt Mullenweg, who is a platinum sponsor of the ASF and founding developer of Wordpress (the blogging software that runs on millions of blogsites around the world).
7 Microsoft, Yahoo!, Google, Facebook, HP, and most recently IBM, are the main sponsors of the ASF.
8 The IBM HTTP Server is based on the Apache HTTP server.
9 For example, the netcraft web server survey shows a decrease, starting in May 2007, in the market share for the Apache HTTP server that corresponds to a rise of a Google web server.
10 In 2008 Microsoft became a platinum sponsor of the ASF. http://news.cnet.com/8301-10805.3-9999824-75.html (August 2010)
11 http://sourceforge.net/
12 http://code.google.com/
13 The Apache Software Foundation - How the ASF works
14 The Apache Software Foundation - How the ASF works - Meritocracy
15 The Apache Software Foundation - The Apache Way -
http://incubator.apache.org/learn/theapacheaway.html
16 The Apache Software Foundation - How it works - Roles enabled by meritocracy
http://www.apache.org/foundation/how-it-works.html#roles (August 2010)
17 This clarifies why splitting up software into modules, as described in Chapter 2, does not only influence the dependent variables lines of code and revision count, but more importantly also results in an increased level of observed organization measured as the degree of clustering or hierarchy in an open source community.
18 These reasons explain why we chose the three design principles of the organizational model of open source of (Van Wendel de Joode, 2005).
19 Thanks to sponsorship from the Google Open Source Programs Office through its Google Summer of Code programs in 2009 and 2010.
20 Thanks to sponsorship from the European Union’s AlBan programme
21 Thanks to sponsorship from ApacheCon’s Travel Assistance Committee.
22 Conway’s Law:
23 Weber (2004) refers to this as the ‘dysfunctional flip side of Conway’s Law.’
24 For instance Resnick (1994) (pg. 19-25) describes how people he described his self-organization ideas to always seemed to look for centralized explanations.
25 The Foundation for P2P Alternatives - Stigmergy
http://p2pfoundation.net/Stigmergy
26 NB: The argued direction of causality is: modularity results in loose coupling. This seems vague in Weber (2004) (pg. 172) because of the words “to what” but upon check reveals the same causal direction.
27 Key signing is usually conducted at ApacheCon conferences. It is an opportunity for
committees and general attendees to sign each other’s PGP or GPG keys and grow the ASF’s Web Of Trust.

http://wiki.apache.org/apachecon/PgpKeySigning

28 The Apache Software Foundation - Apache Web of Trust
http://people.apache.org/ erikabele/tools/wot/wot.html

29 The Apache Software Foundation - Consensus Gauging through Voting - Binding Votes
http://www.apache.org/foundation/voting.html (September 2010)

30 The Apache Software Foundation - Consensus Gauging through Voting - Implications of Voting
http://www.apache.org/foundation/voting.html (September 2010)

31 We use ‘acquiring ownership in’ (ie. within) an open source project to clarify that ownership can be distributed.

32 The Apache Software Foundation - How the ASF works - Roles
http://www.apache.org/foundation/how-it-works.html#roles (September 2010)

33 Apache HTTP Server Project - How Apache Came to Be
http://httpd.apache.org/ABOUT_APACHE.html

34 Apache HTTP Server Project - How Apache Came to Be
http://httpd.apache.org/ABOUT_APACHE.html

35 The Apache Software Foundation - How the ASF works - Roles
http://www.apache.org/foundation/how-it-works.html#roles (October 2010)

36 The Apache Software Foundation - How the ASF works - Roles
http://www.apache.org/foundation/how-it-works.html#roles (September 2010)

37 The Apache Software Foundation - How the ASF works - Roles
http://www.apache.org/foundation/how-it-works.html#roles (September 2010)

38 The Apache Software Foundation - How the ASF works - Meritocracy
http://www.apache.org/foundation/how-it-works.html#meritocracy (September 2010)

39 The most famous example is that of the Linux kernel.

40 Because Apache communities are highly connected, as will be explained in Chapter 5, the condition of upperboundedness turns out to have the same value across all communities. Therefore, we have chosen to drop this variable and not include it for further analysis.

41 Although there are individual contributors that participate in more than one project, this is the exception rather than the rule.

42 Apache Tomcat
http://tomcat.apache.org/

43 Apache Subversion - Subversion is now an official ASF project!
http://svn.haxx.se/dev/archive-2010-02/0418.shtml

44 The Apache Software Foundation - Project Listing
http://projects.apache.org/indexes/quick.html (October 2010)

45 The Apache Software Foundation - Apache Projects - Project Releases
http://projects.apache.org/indexes/releases.html (October 2010)

46 The Apache Software Foundation - Project Listing
http://projects.apache.org/indexes/pmc.html (October 2010)

47 The Apache Software Foundation - Apache Projects - Project Categories
http://projects.apache.org/indexes/category.html (October 2010)

48 The Apache Software Foundation - Apache Projects - Alphabetical Index
http://projects.apache.org/indexes/alpha.html (October 2010)

49 The Apache Software Foundation - Apache ZooKeeper: “Apache ZooKeeper is an open source [...] subproject of Hadoop.”
http://hadoop.apache.org/zookeeper/

50 The Apache Software Foundation - Hive credits
http://hive.apache.org/credits.html

51 The Apache Software Foundation - Pig - Who are We?
http://pig.apache.org/whoweare.html

52 The Apache Software Foundation - HBase credits
http://hbase.apache.org/credits.html

53 The Apache Software Foundation - Common - Credits
http://hadoop.apache.org/common/credits.html

54 The Apache Software Foundation - HDFS - Credits
The design principles are included in the model because we are using the organizational model of open source of Van Wendel de Joode (2005) as a guide for research into the organization of open source communities, and because the design principles provide “(...) a way to decide where to look for mechanisms to understand how the communities are organized” (Van Wendel de Joode, 2005) (pg. 38). However, we did not measure particular aspects of the design principles, even though we did note they are connected in interesting ways.

Many thanks to Daniel Shahaf and Tony Stevenson of the Apache Infrastructure Team.

The repository was mounted on the high performance computing cluster (HPC) of the Faculty of Technology Policy and Management at Delft University of Technology.

We decided to ignore file deletions and deletions within source code files since such events do not constitute social acts of production but are instead associated with the absence of a social relationship.

The design principles are included in the model because we are using the organizational model of open source of Van Wendel de Joode (2005) as a guide for research into the organization of open source communities, and because the design principles provide “(...) a way to decide where to look for mechanisms to understand how the communities are organized” (Van Wendel de Joode, 2005) (pg. 38). However, we did not measure particular aspects of the design principles, even though we did note they are connected in interesting ways.
Eclipse Software Foundation - Eclipse Source Repositories
http://dev.eclipse.org/viewsvn/index.cgi/

FreeBSD.org subversion server
http://svn.freebsd.org/

Debian.org
http://svn.debian.org/

Google Code - Google Code projects
https://PROJECTNAME.googlecode.com/svn/

SourceForge - SourceForge projects
https://PROJECTNAME.svn.sourceforge.net/svnroot/PROJECTNAME

A similar direction for future research is suggested by (Crowston et al., 2006).

Wikipedia - Clickthrough rate
http://en.wikipedia.org/wiki/Clickthrough_rate

This idea was proposed in a research paper for the course MOT9556: Corporate Entrepreneurship at Delft University of Technology
http://blackboard.tudelft.nl/bbcswebdav/users/ocastaneda/TUDelft/MOT9556.pdf

This idea was proposed in a research paper for the course IN4324: Web and Semantic Web Engineering at Delft University of Technology.
http://blackboard.tudelft.nl/bbcswebdav/users/ocastaneda/TUDelft/IN4324.pdf

W3C - Resource Description Framework (RDF)
http://www.w3.org/RDF/

Carter’s Archive of S Routines for the R Statistical Computing Environment
http://erzuli.ss.uci.edu/R.stuff/
Bibliography


Kevin Crowston and James Howison. The social structure of free and open source software development. First Monday, 10(2), 2005.


Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1st edition, August 2000.


Faré Rideau. About eric s. raymond’s articles - projects’ nature and ownership, October 2010. URL http://fare.tunes.org/articles/about_esr.html#pnao.


BIBLIOGRAPHY


BIBLIOGRAPHY


Appendix A

Additional Figures and Tables

A.1 Tables for Section 5.2
Figure A.1: Scatterplots for (a) File Count vs. Lines of Code and (b) File Count vs. Revision Count, both for Pooled data (2004-2009).
Figure A.2: Scatterplots for (a) Author Count vs. Lines of Code and (b) Author Count vs. Revision Count, both for Pooled data (2004-2009).

(a) filecount vs. loc: 2004-2009
(b) filecount vs. rev: 2004-2009

123
Figure A.3: Scatterplots for (a) Clustering Coefficient (Watts and Strogatz, 1998) vs. Lines of Code, and (b) Clustering Coefficient (Watts and Strogatz, 1998) vs. Revision Count, both for pooled data (2004-2009).
Figure A.4: Scatterplots for (a) Graph Connectedness (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Connectedness (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2009)
Figure A.5: Scatterplots for (a) Graph Hierarchy (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Hierarchy (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2008).
Figure A.6: Scatterplots for (a) Graph Efficiency (Krackhardt, 1994) vs. Lines of Code, and (b) Graph Efficiency (Krackhardt, 1994) vs. Revision Count, both for Pooled data (2004-2008).
A.2 Tables for Section 5.3

lines of code

Figure A.7: Stata output of `gladder loc` command

revision count

Figure A.8: Stata output of `gladder rev` command
**filecount**

Figure A.9: Stata output of `gladder filecount` command

**authorcount**

Figure A.10: Stata output of `gladder authorcount` command
hierarchy

Figure A.11: Stata output of gladder hierarchy command

connectedness

Figure A.12: Stata output of gladder connectedness command
A.3 Tables for Section 5.4
Table A.1: Correlation Table showing all independent variables

<table>
<thead>
<tr>
<th>Variables</th>
<th>File Count</th>
<th>Author Count</th>
<th>Graph Connectedness</th>
<th>Graph Hierarchy</th>
<th>Graph Efficiency</th>
<th>Clustering Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. File Count</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2. Author Count</td>
<td>0.3399</td>
<td>0.9357</td>
<td>0.6877</td>
<td>0.1877</td>
<td>0.3757</td>
<td>0.2438</td>
</tr>
<tr>
<td>3. Graph Connectedness</td>
<td>-0.4880</td>
<td>-0.3536</td>
<td>-0.1369</td>
<td>-0.6527</td>
<td>-0.3399</td>
<td>-0.2438</td>
</tr>
<tr>
<td>4. Graph Hierarchy</td>
<td>0.9357</td>
<td>0.6877</td>
<td>0.1877</td>
<td>0.3757</td>
<td>0.2438</td>
<td>0.1867</td>
</tr>
<tr>
<td>5. Graph Efficiency</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

*p < 0.05 (two-tailed); **p < 0.01 (two-tailed); N = 48
Figure A.13: Graph matrix showing all independent variables.
A.3 Tables for Section 5.4 Additional Figures and Tables

A.3.1 First (untransformed) model: explaining lines of code

Stepwise Regression using *Lines of Code* (loc) as the dependent variable

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>β</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Code</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Independent variables</th>
<th>β</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>File count</td>
<td>.060</td>
<td>10.42***</td>
</tr>
<tr>
<td>Author count</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph connectedness</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph hierarchy</td>
<td>-840693.3</td>
<td>-2.82***</td>
</tr>
<tr>
<td>Graph efficiency</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Clustering coefficient</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

R² = .406
Adjusted R² = .400
F-value = 65.81

†, p < .10; *, p < .05; **, p < .01; ***, p < .001; N = 195

Table A.2: Regression Model with *Lines of Code* (loc) as the dependent variable.

A.3.2 Second (untransformed) model: explaining revision count

Stepwise Regression using *Revision Count* (rev) as the dependent variable

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>β</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision Count</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Independent variables</th>
<th>β</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>File count</td>
<td>.060</td>
<td>4.50***</td>
</tr>
<tr>
<td>Author count</td>
<td>47.066</td>
<td>13.35***</td>
</tr>
<tr>
<td>Graph connectedness</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph hierarchy</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph efficiency</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Clustering coefficient</td>
<td>1628.138</td>
<td>4.51***</td>
</tr>
</tbody>
</table>

R² = .640
Adjusted R² = .634
F-value = 113.41

†, p < .10; *, p < .05; **, p < .01; ***, p < .001; N = 195

Table A.3: Regression Model with *Revision Count* (rev) as the dependent variable.
A.3.3 First (discarded) extended model: explaining lines of code per author count (productivity)

Stepwise Regression using $\log(\text{Lines of Code per author})$ ($\text{loc\_per\_author\_log}$) as the dependent variable

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>$\beta$</th>
<th>$(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log(\text{Lines of Code per author})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Independent variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File count</td>
<td>0.036</td>
<td>9.15***</td>
</tr>
<tr>
<td>Author count</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph connectedness</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph hierarchy</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph efficiency</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Clustering coefficient</td>
<td>2.789</td>
<td>3.69***</td>
</tr>
</tbody>
</table>

$R^2$ | .386 |
Adjusted $R^2$ | .380 |
F-value | 60.51 |

†, $p < .10$; *, $p < .05$; **, $p < .01$; ***, $p < .001$; N = 195

Table A.4: Regression Model with $\log(\text{Lines of Code per author})$ ($\text{loc\_per\_author\_log}$) as the dependent variable.

A.3.4 Second (discarded) extended model: explaining revision count per author count (productivity)

Stepwise Regression using $\sqrt{\text{Revision Count per author}}$ ($\text{rev\_per\_author\_sqrt}$) as the dependent variable

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>$\beta$</th>
<th>$(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sqrt{\text{Revision count per author}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Independent variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File count</td>
<td>0.035</td>
<td>5.38***</td>
</tr>
<tr>
<td>Author count</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph connectedness</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph hierarchy</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Graph efficiency</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Clustering coefficient</td>
<td>8.118</td>
<td>6.54***</td>
</tr>
</tbody>
</table>

$R^2$ | .329 |
Adjusted $R^2$ | .322 |
F-value | 47.27 |

†, $p < .10$; *, $p < .05$; **, $p < .01$; ***, $p < .001$; N = 195

Table A.5: Regression Model with $\sqrt{\text{Revision count per author}}$ ($\text{rev\_per\_author\_sqrt}$) as the dependent variable.
Appendix B

Contributions to SVNPlot
Contributions to SVNPlot

---

python script to process a Subversion log collected by svnlog2sqlite.py, which is stored in a sqlite database. The idea is to use the SQLite database generated by SVNPlot to create the XML input file for CMU’s Organizational Risk Analyzer (ORA). Using ORA several SNA graphs and analyses may be conducted.

Note: This version was inspired by Apache Agora. It considers commits as part of conversations (like email conversations in Apache Agora). Upon committing code, a committer creates a revision in SVN which in turn creates a link to all committers who have co-authored the corresponding files from that revision. The idea is the same as in Agora, namely to create links based on reply actions, but differs in that there is no one originator but instead links are created to all co-authors who are active in the sqlite db contents.

This version of svnsqlite2ora.py has been tested with SVNPlot version 0.6.1.

---

import sqlite3
import datetime
from datetime import date
import calendar
import string
from datetime import datetime
from optparse import OptionParser
from numpy import *
from numpy import matrix

class SVNSqlite2Ora:
    def __init__(self, sqlitedbpath, outputfilepath):
        self.dbpath = sqlitedbpath
        self.dbcon = None
        self.outputfile = outputfilepath
        self.Process()

    def initdb(self):
        self.dbcon = sqlite3.connect(self.dbpath, detect_types=sqlite3.PARSE_DECLTYPES|sqlite3.PARSE_COLNAMES)

    def closedb(self):
        self.dbcon.commit()
        self.dbcon.close()

    def Process(self):
        output = open(self.outputfile, 'w')
        self.initdb()
        print "Processing..."

Figure B.1: Code snippet (1) svnsqlite2ora.py

---
revisions = []
revisions_count = 0
r = {}

committers = []
committer_count = 0
c = {}

# Write XML prelude to CMU node specification
output.write("<?xml version="1.0" standalone="yes" ?>\n")
output.write("<DynamicMetaNetwork id="Meta Network"/>\n")

# We create a cursor for SVNLog and do a SELECT on all records (*), so cur = SVNLog
cur = self.dbcon.cursor()

# Write XML specification for MetaNetwork, then start writing <nodes> section of the XML file consisting of Agents
output.write("<MetaNetwork id="Meta Network" date="20000101T00:00:00"/>\n")
output.write("<nodes/>\n")

# We go through all the committers and their revisions, then we create lists of both.
cur.execute('SELECT * FROM SVNLog')
for row in cur:
    committer = row[2]
    revno = row[0]

    # If committer has not been counted then add him/her to the list, and increment committer_count
    # then write <node id> in XML file.
    if (committers.count(committer) == 0):
        committers.append(committer)
        committer_count = committer_count + 1
        c[committer] = committer_count
        output.write("<node id=" + "%s" %committer + "/\n")

    # If a revision has not been counted then add it to the list, increment revision_count and
    # associate revision to committer.
    if (revisions.count(revno) == 0):
        revisions.append(revno)
        revisions_count = revisions_count + 1
        r[revno] = committer

committer_count = committer_count + 1
cur.close

# Finish the <nodeclass> and <nodes> section, and start the <networks> section
# of the XML file.
output.write("</nodeclass>\n")
output.write("</nodes>\n")
output.write("</networks>\n")

Figure B.2: Code snippet (2) svnsqlite2ora.py
# Write sociomatrix from
# Agent x Resource(changedpathid) and Resource(changedpathid) x Agent
# Create a matrix of committers with the dimensions we found out previously.
mat = array([[0]*committer_count]*committer_count)
for row in cur:
    committer = row[2]
    revno = row[0]
    cur2 = self.dbcon.cursor()
    cur2.execute('SELECT * FROM SVNLogDetail where revno=' + "'" + str(revno) + "'")
    for row2 in cur2:
        changedpathid = row2[1]
        # Iterate over the individual files (changedpathid's) to get the work contents
        # from them, namely lines-of-code (loc).
        # Note: We only take into account lines added (row3[6]) and not lines deleted
        # because we are interested in what committers 'do' and that is more evident from
        # the loc they add, and not so from the loc they delete. Furthermore, negative links
        # between developers are meaningless.
        cur3 = self.dbcon.cursor()
        cur3.execute('SELECT * FROM SVNLogDetail where changedpathid=' + "'" + str(changedpathid) + "'")
        for row3 in cur3:
            # We only consider the lines of code that have been added by a committer.
            loc = row3[6]
            # And create links to all previous committers who have revised this same
            # file, ie. file co-authorship.
            if (row3[0] <= row2[0]):
                mat[c[committer]][c[r[row3[0]]]] = mat[c[committer]][c[r[row3[0]]]] + loc
            else:
                continue
cur.close
cur2.close
cur3.close

Figure B.3: Code snippet (3) svnsqlite2ora.py
# Then we prepare for writing the file co-authorship networks.
output.write("<network sourceType="Agent" source="" targetType="Agent" target="Agent" id="Agent-Agent">\n")

# We iterate over the resulting matrix to write it out to the XML file.
i = 0
j = 0
for i in c:
    for j in c:
        output.write("<link source=""%s"" target=""%s"" value=""%s"">\n")

self.closedb()

def RunMain():
    usage = "(File co-authorship version) usage: %prog <sqlitedbpath> <outputfile>" 
    parser = OptionParser(usage)
    (options, args) = parser.parse_args()

    if( len(args) < 2):
        print "Invalid number of arguments. Use svnsqlite2ora_filecoauthorship.py --help to see the details." 
    else:
        sqlitedbpath = args[0]
        outputfilepath = args[1]

    try:
        print "Processing the sqlite subversion log"

        SVNSQLite2Ora(sqlitedbpath, outputfilepath)
    except:
        raise

if( __name__ == "__main__"):
    RunMain()
Appendix C

Research Proposals

C.1  Google Summer of Code 2009
C.2  Google Summer of Code 2010
How are new open source communities in the ASF organized and how do they sustain themselves? This question requires an understanding of the collective institution of voting in the purposeful creation of new open source communities and the role of institutions in self-organization. It also requires an understanding of the complex interactions between community structure and voting processes.

The role of institutions in self-organization was found to be minor compared to individual behavior (Van Wendel de Joode, 2005). Nevertheless, the role of institutions in the organization of open source communities, one that focuses on the use of metaphors to explain self-organization (think cathedral) and the other which is its opposite (think bazaar) was identified. This exception leads to the following proposition suggested as a hypothesis for future research:

Proposition: How are new open source communities developed in the Apache Incubator?

Furthermore, Van Wendel de Joode (2005) puts forth the following interesting exception:

“Studying in more detail the emergence and role of institutions in self-organization of open source communities, one that focuses on the use of metaphors to explain self-organization (think cathedral) and the other which is its opposite (think bazaar) was identified. This exception leads to the following proposition suggested as a hypothesis for future research:

Proposition: How are new open source communities developed in the Apache Incubator?"
The project has several goals at different levels, most of which are shared goals with my masters thesis. The main goals are described below.

**Stage 0: Introductory research and testing**

- Ability to mine complete SVN repositories and mailing lists.
- Acquired familiarity with data collection and analysis tools.
- Testing of data collection and analysis tools.

**Stage 1: Data collection: mining repositories and archives**

- Duration: 1 to 2 weeks
- Milestone: Ability to mine complete SVN repositories and mailing lists.
- Deliverable: Community structure of individual communities.

**Stage 2: Individual Community Structure**

- Duration: 1 to 2 weeks
- Milestone: Ability to obtain community structure from mined data bases.

**Stage 3: Social Network Analysis**

- Mini scenario / demo: Graphs to be used for social network analysis.
- Mini scenario / demo: Queries from database showing useful data.

**Stage 4: Data Mining**

- Set the stage for agent-based simulation of new open source communities.
- Determine the possibility of using statistical analysis to obtain the relevance and importance of voting in new open source communities in the Apache Incubator.
- Establish a research direction in which other factors relevant in the context of a project that has both a practical and a theoretical foundation, can be investigated.

The project has the potential of providing a basis for further research into the role of open source communities and their relation to the collective institution of voting in new open source communities. The project will enable the evaluation of Apache Incubator projects and their use in Apache Labs. Additionally, because the stages are considered success factors for this project, an estimated duration has been already started working on some activities. Even though iteration and early start iteration between stages, which means that there will be overlaps, but the general orientation come from the research of Van Wendel de Joode (2005) and my own thought to myself '...it would be nice to have a tool like Apache Agora for this...' I see the idea as being useful to the ASF for continued analysis of projects in the Apache Incubator. Metrics recently suggested for evaluation of incubator projects as a vital interfacing role between communities and corporations (O'Mahony, 2008), and the analytical framework and organizational development practices.

The idea for this project came about from a need for data collection for my thesis. The analysis will be based on the evolution of community structure over time and its relation to the collective institution of voting in new open source communities. The analysis will be performed in my thesis, so that, if found to be useful, it can continue into the 'Community structure of modules in the Apache project' (Gonzales-Barahona and Robles, 2005), and the analytical framework and organizational evolution of community structure over time and its relation to voting processes in new open source communities. Based on the data gathered from those sources, the social network structure of the communities can be constructed and analyzed. I propose that patterns in the evolution of organization of new open source communities can be investigated in the context of a project that has both a practical and a theoretical foundation, and that is rooted in rigorous academic research and innovative open source development practices.
I wish that my efforts provide a useful starting point for future GSoC students. The University of Syracuse who have been studying Apache for years.

Bergman, Chair of the Apache Incubator PMC, and asked him for help on my thesis. He gladly accepted and promised to get me in touch with researchers from last, but most definitely not least, one of the long-term goals of this project is to propose an alternate project seems to be the most appropriate submission. And usually lack a community, but are geared towards research and innovation, and reflect on the relationship of this role to "The Apache Way".

There are several reasons for an alternate project. First, I have already found a previous academic research. Third, the project I propose is multi-disciplinary in nature, and as such benefits from the supervision of a faculty member that has been extensively involved in multi-disciplinary research, and allows me to gain experience in such research. Fourth, the main goal of this project is to implement in the evolution of community structure in new open source communities which can be investigated in the context of a project that has both a practical and a theoretical foundation, and that is rooted in rigorous academic studies.

Why proposal? In the context of a project that has both a practical and a theoretical foundation, and that is rooted in rigorous academic studies. Why propose an alternate project? Why Apache Labs?

I am an Apache committer so I have access to create a new Apache Lab. Apache Labs is a place for innovation where committers of the foundation can experiment with new ideas. Labs provides a place for the Apache committer community to collaborate on such efforts - without discrimination of purpose, medium or implementation technology.

There are several reasons to choose Apache Labs, including:

- Labs is a place for innovation where committers of the foundation can experiment with new ideas.
- Labs provides a place for the Apache committer community to collaborate on such efforts - without discrimination of purpose, medium or implementation technology.
- Labs allows for the implementation of projects that are not yet ready for deployment.
- Labs is a place for research and development.
- Labs is a place for innovation where committers of the foundation can experiment with new ideas.
- Labs allows for the implementation of projects that are not yet ready for deployment.
- Labs is a place for research and development.

Why Apache Labs?

Apache Labs is a place for innovation where committers of the foundation can experiment with new ideas. Labs provides a place for the Apache committer community to collaborate on such efforts - without discrimination of purpose, medium or implementation technology.

Apache Labs is a place for innovation where committers of the foundation can experiment with new ideas. Labs allows for the implementation of projects that are not yet ready for deployment.

Why propose an alternate project? Why Apache Labs?

I am an Apache committer so I have access to create a new Apache Lab.

I am an Apache committer so I have access to create a new Apache Lab.

Copyright © 2009, M.J.G. van Eeten. All rights reserved.

Delft University of Technology
Faculty of Technology, Policy and Management

Dr. M.J.G. van Eeten
Proposed mentor

Figure C.3: Google Summer of Code 2009 research proposal (pgs. 5,6/9)
Benefits to the Community

As mentioned previously, there seems to be a lack of metrics to evaluate new open source projects and communities in the Apache Incubator. Some metrics are available, but they are mainly about website statistics for the different projects in the ASF [5]. Moreover, recent metrics suggested for evaluation of incubator projects [3] seem overly simplistic, and I think a more in-depth analysis with tools would be useful. Furthermore, such analysis could serve as a starting point for more complex analyses. The main benefit to the community is a tool that can be used to continuously monitor the incubation process of projects and communities in the Apache Incubator. I believe this project will encourage innovation within the Apache Incubator and serve as a starting point for an improved understanding of the organization of new open source communities, and thus will have a positive and lasting effect in the sustainability of these communities.

About me

I'm a highly motivated student from Guatemala. I studied Computer Science and have more than 5 years of work experience between a Checkpoint reseller and IBM. As a student and professional I've worked extensively with Free and Open Source software, and over time acquired a knack for making things work. In 2007, I came to Delft to study a masters in Management of Technology. More recently, I applied and was accepted to the Software Engineering Masters program at Carnegie Mellon Silicon Valley and currently await the admissions decision for the Information Architecture Masters programme at Delft University of Technology. My plan is to continue my studies in Software Engineering. In this regard, the GSoC'09 stipend would be very useful financial support to continue my education.

Project Quote

"Some people dream of great accomplishments; while others stay awake and do them."

-- Constance Newman

Tools

All of the tools proposed for this project are open source software.

[d] https://nwb.slis.indiana.edu/community/?n=Main.NWBTool
[e] http://tools.libresoft.es/mailing_list_stats

References


Acknowledgements

Supported by the Programme AlBan, the European Union Programme of High Level Scholarships for Latin America, scholarship No. E07M401381GT.
Figure C.5: Google Summer of Code 2009 research proposal (pg. 9/9)
My alternate GSoC'09 project marked the start of this research. In that project I focused on stands in my thesis committee and on whose research I initially based mine.

Does Meritocracy influence code production in open source communities?

What is Meritocracy?

I found two mentors for my project, they come from Academia and Open Source.

What is an open source community?

Prof.dr. van Eeten supervised the PhD research of Dr. van Wendel de Joode, who also supervised the PhD research of Dr. Victor Scholten.

Prof.dr. Michel van Eeten is my academic project mentor and Msc. thesis supervisor.

1. Prof.dr. Michel van Eeten is my academic project mentor and Msc. thesis supervisor.

I have been supported by the Programme AlBan, the European Union - Union Programme of High Level Scholarships for Latin America, scholarship No. E07M401381GT.

The European Union - Union Programme of High Level Scholarships for Latin America, scholarship No. E07M401381GT.

The Apache Software Foundation (ASF) explains the success of its communities and the software they produce by claiming that meritocratic principles and organizational orientation to software engineering through evaluation my submission for presentation at ApacheCon NA 2010 (for more information please see release 0-7-20100426p1).

Four organizations come together in supporting my Google Summer of Code work. The ASF explains the success of its communities and the software they produce by claiming that meritocratic principles and organizational orientation to software engineering through evaluation my submission for presentation at ApacheCon NA 2010 (for more information please see release 0-7-20100426p1).

Nitin Bhide

The SVNPlot open source project - [A]

Figure C.6: [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

Because my proposed alternate project brings together four different mentoring organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

The theoretical aim behind this research project is to implement the official SVNPlot release for active measuring of community Meritocracy within the ASF (especially useful for incubating projects). And present research findings at ApacheCon NA 2010.

Propose an Apache Lab to implement the official SVNPlot release for active measuring of community Meritocracy within the ASF (especially useful for incubating projects). And present research findings at ApacheCon NA 2010.

There are eight design principles in the [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

Because my proposed alternate project brings together four different mentoring organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

The model is intended to serve as input to discussion and reflection and as a framework for analysis of the institutional success of the communities supported by each of these foundations. It is characterized by the modular design principles proposed in [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

The model is intended to serve as input to discussion and reflection and as a framework for analysis of the institutional success of the communities supported by each of these foundations. It is characterized by the modular design principles proposed in [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

The model is intended to serve as input to discussion and reflection and as a framework for analysis of the institutional success of the communities supported by each of these foundations. It is characterized by the modular design principles proposed in [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

The model is intended to serve as input to discussion and reflection and as a framework for analysis of the institutional success of the communities supported by each of these foundations. It is characterized by the modular design principles proposed in [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

The model is intended to serve as input to discussion and reflection and as a framework for analysis of the institutional success of the communities supported by each of these foundations. It is characterized by the modular design principles proposed in [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

The model is intended to serve as input to discussion and reflection and as a framework for analysis of the institutional success of the communities supported by each of these foundations. It is characterized by the modular design principles proposed in [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.

The model is intended to serve as input to discussion and reflection and as a framework for analysis of the institutional success of the communities supported by each of these foundations. It is characterized by the modular design principles proposed in [C.6.1] Add social network analysis capabilities, through libsna, to the code that was developed as part of GSoC'09. Produce output in GraphML format to enable follow-up visualization and analysis with Gephi (thereby eliminating dependency to CMU's (non-open source) social network analysis software). Also, because SVNPlot is not a mentoring organization for this project, I am developing my own visualization/analysis tools. However, as SVNPlot does not have similar capabilities, I am not able to use its analysis tools. This may result in different interpretations of the data.

Additionally, this project aims to develop into an Apache Lab for which there currently are no organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason why doesn't your project fit into any of the work being done by the other mentor organizations: the ASF, Gephi, SVNPlot and Delft University of Technology. Another reason is because this project has an academic focus and is based on ongoing research.
Stage 0: Introductory research and testing

Mini scenario / demo: Results from complex data mining exercises.

C.2 Google Summer of Code 2010 Research Proposals

Duration: 1 to 2 weeks

Milestone: Ability to perform fully automated data mining on complete SVN repositories.

The project has the potential to contribute to a proposed incubator project and community.

More broadly, the proposed alternate project would be useful for incubator projects and communities. For instance, it could be used for active mentoring of students.

In general, the proposed alternate project would be useful in the evaluation of building and code production, this alternate project could be useful in the evaluation of incubator projects and communities. For instance, it could be used for active mentoring of students.

The project aims to evaluate the role of Meritocracy in community building and code production in the ASF, EF, and PSF. Furthermore, ongoing research in the ASF, EF, and PSF might be considered success factors for this project.

The structure and goals of the project will remain focused on institutions in open source communities, in general, and incubating communities in particular.

The organizational analysis this alternate project will be a part of focuses on open source software. Moreover, the proposed alternate project will enable me to wrap-up projects that I am using for my master's thesis. The main goals are to present my research findings in the context of a project that has both a practical and scientific publication.

Key Techniques

- Social network analysis
- Data mining
- Pattern recognition
- Machine learning
- Visualization

This alternate project is about automating the analysis of voting and community structure in the ASF, EF, and PSF. Furthermore, ongoing research in the ASF, EF, and PSF might be considered success factors for this project.

The organizational analysis this alternate project will be a part of focuses on open source software.

Key Tools

- SVNPlot
- Gephi
- CMU's *ORA
- Tools for pattern recognition and machine learning
- Tools for data mining
- Tools for visualization

This project has various goals at different levels, most of which are shared goals with my ongoing master's thesis work. So even though iteration and an in-depth understanding of organizational open source is already providing compelling insights for the thesis, it will be possible to contribute to the SVNPlot project that enables: (1) collection of data, (2) visualization and analysis with Gephi. Furthermore, the resulting code of this alternate project will be integrated into 'trunk' to make it part of the official SVNPlot release.

Goals

- Automate data collection and analysis for the thesis
- Implement the tools used for analysis in Apache Labs
- Set the stage for continued research that applies Pattern Recognition and Machine Learning
- Explore interesting directions for future research proposed by Van Wendel de Joode
- Scratch my own itches -- fully automate the data collection and analysis that are part of this project.

This coming November, I will propose an Apache Lab to implement the official SVNPlot release for active mentoring of students.

The project will be integrated into 'trunk' to make it part of the official SVNPlot release.
for traveling to Atlanta to present my research findings at ApacheCon NA this coming summer. In this regard, I have discussed with several researchers in the field of Software Engineering and Management, as my research interests show. A second alternate GSoC project would enable me to explore directions of future research, for which I am particularly interested in seeing the SVNPlot community grow. As such, Nitin Bhide is my open source project mentor, he is also my faculty advisor and master's thesis supervisor, he also supervised the PhD research of Dr. van Wendel de Joode, who also stands in my thesis committee. Nitin Bhide is my open source project mentor and master's thesis supervisor, he also supervised the PhD research of Dr. van Wendel de Joode, who also stands in my thesis committee. My aim is to combine software engineering and software management, as my research interests show. A second alternate GSoC project would enable me to explore directions of future research.

I have found two mentors to supervise my project: my thesis supervisor Prof.dr. Michel van Eeten and SVNPlot project founder Nitin Bhide. Prof.dr. Michel van Eeten is my academic project mentor and master's thesis supervisor, he also supervised the PhD research of Dr. van Wendel de Joode, who also stands in my thesis committee. Figures C.8 - C.18 show the projects I have undertaken, all of which were based on academic research. Nitin Bhide is my open source project mentor, he has been extensively involved in multi-disciplinary research, which in turn allows me to gain experience in such research.

There are other important reasons.

Figure C.8: Google Summer of Code 2009 research proposal (pgs. 5,6/10)
C.2 Google Summer of Code 2010

Research Proposals

Figure C.9: Google Summer of Code 2009 research proposal (pgs. 7,8/10)
On 10th April 2010 11:39
Oscar Castaneda wrote:
Hi,
I forgot to mention in my proposal that I will be attending the 24th
"European Conference on Operational Research" this coming July in
Lisbon. I am attending to explore the possibility of presenting my research
at the next (25th) EURO conference. More information
here:
best,
-oscar

On 14th April 2010 11:26
Carol Smith wrote:
Oscar -
We like your proposal, but have been in contact with both of the mentors
you list on your proposal and neither has responded.
Can you contact them please and confirm that they are able to mentor?

On 14th April 2010 11:35
Oscar Castaneda wrote:
Hi Carol,
I contacted both mentors and sent my proposal to them. I also asked them
to please confirm their availability for mentoring my project.
Thanks.
best,
-oscar

On 15th April 2010 15:10
Oscar Castaneda wrote:
Hi Carol,
As of today, Nitin Bhide has confirmed. I contacted Prof. van Eeten and
his secretary, yesterday and today, to kindly request confirmation.
Otherwise you can also reach him by telephone on the number listed in
my proposal.
Thanks.
best,
-oscar

On 16th April 2010 11:11
Oscar Castaneda wrote:
Hi Carol,
Prof. van Eeten informed me he has sent his confirmation.

Thanks.
best,
-oscar

On 24th April 2010 18:05
Oscar Castaneda wrote:
Hi,
I loaded the entire ASF repository on my server! Now I can experiment
more freely :-) Just thought i’d share the good news. I owe thanks to the
ASF infrastructure team.
More information
here:
http://blackboard.tudelft.nl/bbcswebdav/users/ocastaneda/gsoc/ASFInfraEmail_24Apr2010.pdf
best,
-oscar
Appendix D

Posters and Presentations

D.1 Poster presented at AlBan Conference

D.2 Presentation at ApacheCon NA 2010

D.2.1 Abstract

D.2.2 Presentation Slides

These slides and a recording of the talk I gave at ApacheCon NA 2010 can be found online: http://people.apache.org/ocastaneda/apacheconna.html.

D.2.3 Blog Post
Evolution of New Open Source Communities

Oscar Castañeda

Introduction

This paper aims to understand the evolution of open source communities, a group of stakeholders contributing to the code of a software project. These communities are typically compared to virtual commons like those found on the Internet. Importantly for this research, the study will look at open source communities that have been part of The Apache Software Foundation (ASF). The Apache Software Foundation is a non-profit organization aiming to develop open-source software. Its main goal is to provide a neutral environment where individuals can contribute to the development of software.

Burning questions...

1. How do new open source communities evolve in the Apache Incubator?
2. How are the processes of 'The Apache Way' related to the organization of new open source communities?
3. How does voting influence community structure over time?

How are open source communities organized and how do they sustain themselves?

Recent organizational research into open source communities by Van Wendel de Joode (2005), showed that open source communities are self-organizing, which limits their malleability and critically questions whether it is possible to purposefully design and invent new open source communities. To come to this conclusion, Van Wendel de Joode (2005) adopted a framework from research on community managed common pool resources. The framework consists of eight design principles that reconcile the two main strands of state of the art research into the organization of open source communities: self-organization and for purpose.

How does voting influence community structure?

To study other communities managing common pool resources; from farming communities sharing irrigation systems for more than 1,000 years, to global commons like telecommunications providers that transcend national boundaries. The framework used by this project is "...and the corresponding file co-authorship social networks over time.

Observations in a new open source community currently under incubation

Social network analysis of email message interaction as a proxy for voting...

...and the corresponding file co-authorship social networks over time.

Next steps

The next steps for this project are to analyze social network interaction over time at different levels of analysis and to different open source communities. These communities are either currently under incubation or have recently graduated from the Apache Incubator. The focus is on identifying patterns that may explain the influence of institutions like voting on community structure.

Acknowledgements:

Supported by the Programme Al|BaN, the European Union Programme of High Level Scholarships for Latin America, scholarship No. E07M401381GT.

Faculty of Technology, Policy and Management
TUDelft University of Technology

Figure D.1: Poster presented at AlBan Conference
This talk started with a project proposal ... 

Overview
• Institutions in open source.
• Modeling behavior.
• Measuring behavior.

What are institutions?
• Rules that underlie the behavior of individuals.
• Allow for reflection at a collective level.
• Institutions can be engineered.
• But also have a natural dimension (Selznick, 1984).

Why are institutions important?
• They can be used to distinguish between open source communities.
• ASF vs. Google Code or SourceForge.
• ASF vs. Python SF, Eclipse SF.

Why are institutions important?
• Useful in decision-making.
• Delimiting the boundaries of an open source community.

Why are institutions important?
• Delimiting the boundaries of an open source community.
• Individuals co-author source code files.
• The resulting network delimits the community.
• Literally: community through code.

Figure D.2: Presentation abstract ApacheCon NA 2010

Figure D.3: Slides Presented at ApacheCon NA 2010 (pg. 1/5)
Modeling behavior

- Useful to gain a deeper understanding
  - How are communities organized?
    - e.g. Are there sub-communities?
  - How does behavior influence code production?

Modeling behavior

- How is the network constructed?
  - Original author always gets incoming links
  - Subsequent authors only get incoming links from later co-authors

Modeling behavior

- What other aspects were modeled?
  - Clustering
  - Average distance

Small-world effect

- In 1967, Stanley Milgram:
  - Gave letters to 160 random people, each
  - Addressed to a stockbroker in Boston,
  - To be delivered by first-name connections.
  - 42 letters delivered
  - 5.5 intermediaries

Measuring behavior

- In 1967, Stanley Milgram:
  - “If we make a chart of social interactions, of who talks to whom, the clusters of dense interaction in the chart will identify a rather well-defined hierarchic structure. The groupings in this structure may be operationalized by some measure of frequency of interaction in this sociometric matrix.”
  - Simon (1997), pg. 186

Small-world effect

- Social networks tend to have short average distances between nodes
  - Many highly connected nodes
  - Some nodes also have global connections

Small-world effect

- High clustering coefficient, short paths
  - Self-organized

Small-world effect

- Regular graphs
  - High clustering coefficient, long paths
    - Fully structured
  - Random graphs
  - Grids (low clustering, shortest paths)
  - Small-world graphs
  - "Small-world in between" (Watts and Strogatz, 1998)

Measuring behavior

- Institutionalized behavior
  - Follows rules or norms.

Self-organized behavior

- Emergent
  - "To measure is to know." - Lord Kelvin

Measuring behavior

- Sample: ~260 observations
  - Each observation = 1 project / 1 year
  - Dump of ASF Subversion repository

Tools

- Data mining: SVNPlot (version 0.7.0)
- SNA: *ORA, Gephi

What is SVNPlot?

- A tool that creates various types of graphs and statistics from SVN logs
  - In 2 steps:
    - 1. Convert Subversion logs to sqlite3 db
    - 2. Query database to produce graphs

Figure D.4: Slides Presented at ApacheCon NA 2010 (pg. 2/5)

Figure D.5: Slides Presented at ApacheCon NA 2010 (pg. 3/5)
Posters and Presentations D.2 Presentation at ApacheCon NA 2010

Why is SVNPlot better than others?
– Does not require ‘checked out’ repository
– Separates data collection and report generation (2 steps).
– Easy to write your own tools
– Generate networks of file co-authorship from Subversion logs

In fact, that was the coding part of my GSoC project

Generate networks of file co-authorship from Subversion logs

Measuring behavior
• Measures of hierarchy
  – graph hierarchy (asymmetry)
  – graph connectedness (connectedness)
  – graph efficiency (redundancy)
  » (Krackhardt, 1994)

Measuring behavior
• Measures of self-organization
  – clustering coefficient
  – average distance

Modeling and measuring behavior gives insights on code production

Modeled aspects: varied impact on code production

Self-organization also plays a role
– Apache communities:
  • Highly clustered
  • 1-2 degrees of separation (low average distance)
  • Appear to be small-world networks

Conclusions
• Compare with PSF, ESF, SourceForge, Google Code
• Develop an Apache Agora script extension for SVNPlot
• Recommend files to developers based on behavior

All data up on my Apache page:
– http://people.apache.org/~ocastaneda/
– Collected SVN db’s data available offline.

Future Directions
• Charel Morris, Stone Circle Productions
• The ASF, Apache TAC
• Karl Fogel
• Tony and Daniel ASF Infrastructure team
• Nino Biddle, Founder of SVNPlot and Google’s Open Source Programs Office
• Google’s Open Source Programs Office

Acknowledgements
• Charel Morris, Stone Circle Productions
• The ASF, Apache TAC
• Karl Fogel
• Tony and Daniel ASF Infrastructure team
• Nino Biddle, Founder of SVNPlot and Google’s Open Source Programs Office

Motivation: extend GSoC, not focus on metrics.

Q. Does sustained code production indicate health?
– Is a healthy community one that produces lots of code?

Thanks.
Figure D.8: Blog Post in Google’s Open Source Blog