George Dafermos

Governance Structures of Free/Open Source Software Development

51

NEXT
GENERATION
INFRASTRUCTURES
FOUNDATION

GOVERNANCE STRUCTURES OF FREE/OPEN SOURCE SOFTWARE DEVELOPMENT

examining the role of modular product design
as a governance mechanism in the FreeBSD Project

George DAFERMOS

GOVERNANCE STRUCTURES OF FREE/OPEN SOURCE SOFTWARE DEVELOPMENT

examining the role of modular product design
as a governance mechanism in the FreeBSD Project

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op 10 december 2012 om 15.00 uur

door George DAFERMOS

Master of Science in Electronic Commerce Applications
at University of Sunderland, England
geboren te Irakleio, Griekenland.

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr. M.J.G. van Eeten

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr. M.J.G. van Eeten	Technische Universiteit Delft, promotor
Prof.mr.dr. J.A. de Bruijn	Technische Universiteit Delft
Prof.dr. J.P.M. Groenewegen	Technische Universiteit Delft
Prof.dr. V.J.J.M. Bekkers	Erasmus Universiteit Rotterdam
Prof.dr. J.M. Bauer	Michigan State University
Dr. M. den Besten	Montpellier Business School

ISBN 978-90-79787-40-1

Published and distributed by:
Next Generation Infrastructures Foundation
P.O. Box 5015, 2600 GA Delft, the Netherlands
info@nginfra.nl, www.nginfra.nl

This research was funded by the Next Generation Infrastructures Foundation programme and TU Delft.

To the memory of my mother

Contents

Acknowledgements....xi
Chapter Synopsis....xiii
CHAPTER 1: INCREASING AND DECREASING RETURNS TO SCALE....1
INTRODUCTION....1
INCREASING RETURNS TO SCALE: THE ADVANTAGES OF BIGNESS …1
DECREASING RETURNS TO SCALE....6
Decreasing returns to scale due to coordination costs 6
Decreasing returns to scale due to reduced individual motivation 10
DOES PRODUCT MODULARITY MITIGATE THE ADVERSE EFFECTS OF INCREASING SCALE?....12
CHAPTER 2: LITERATURE REVIEW....15
THE PRODUCTIVITY PARADOX IN SOFTWARE DEVELOPMENT 15
MODULARITY IN ORGANISATION THEORY....18
Product modularity and coordination costs....22
Product modularity and productivity....27
Product modularity and group size....28
STUDYING MODULARITY IN FREE AND OPEN SOURCE SOFTWARE DEVELOPMENT 30
H1: Product modularity reduces coordination costs in FOSS projects....34
H2: Product modularity increases the potential number of contributors to FOSS projects....37
H3: Product modularity has a positive effect on labour productivity in FOSS projects....40
CONCLUDING REMARKS....41
CHAPTER 3: RESEARCH METHODOLOGY....43
ANALYTICAL FRAMEWORK....43
Research Design....43
Object of investigation....44
Level of analysis....46
WHY THE FREEBSD PROJECT?....47
MEASURING MODULARITY....48
MEASURING COORDINATION COSTS....55
MEASURING DEVELOPERS GROUP SIZE....59

MEASURING LABOUR PRODUCTIVITY....63
STATISTICAL ANALYSIS....65
Sample selection....66
Random-effects GLS regression....67
Operationalisation....69
CHAPTER 4: EMPIRICAL SETTING....75
HISTORICAL BACKGROUND....75
ORGANISATIONAL STRUCTURE....78
Core team....79
Committers....82
Outside contributors....84
Ad hoc teams....84
Hats....86
Maintainers....86
TECHNICAL INFRASTRUCTURE....87
Communication channels....87
Revision control....88
Reporting & managing defects....88
Testing....88
Distribution channels....89
DEVELOPMENT PROCESS....89
SCALE....91
CHAPTER 5: MODULARITY AND COORDINATION COSTS IN FREEBSD....95
INTRODUCTION....95
QUALITATIVE ANALYSIS....99
CONCLUDING REMARKS....104
CHAPTER 6: MODULARITY AND GROUP SIZE IN FREEBSD....107
INTRODUCTION....107
QUALITATIVE ANALYSIS....111
QUANTITATIVE ANALYSIS....113
A SUMMING UP....127
REVERSING THE TERMS OF THE PROPOSITION....130
Scale considerations....134
CONCLUDING REMARKS....137
CHAPTER 7: MODULARITY AND LABOUR PRODUCTIVITY IN FREEBSD

.....139
SETTING OF THE PROBLEM.....139
QUALITATIVE ANALYSIS.....142
QUANTITATIVE ANALYSIS.....149
Scale considerations.....150
EFFECT OF MODULARITY ON CORE DEVELOPERS PERFORMANCE 154
Scale considerations.....156
CONCLUDING REMARKS.....160
CHAPTER 8: DOES BROOKS' LAW HOLD IN FREEBSD?.....161
INTRODUCTION.....161
DISAGGREGATING CORE DEVELOPERS' PRODUCTIVITY 164
EFFECT OF GROUP SIZE ON CORE DEVELOPERS PERFORMANCE....167
Scale considerations.....169
Does modularity negate Brooks' Law?.....173
CONCLUDING REMARKS.....174
CHAPTER 9: THE EMERGENCE OF GOVERNANCE.....175
INTRODUCTION.....175
INFORMAL GOVERNANCE PHASE (1993-2000).....176
DEMOCRATIC GOVERNANCE PHASE (2000-TO DATE).....178
THE IMPERATIVE OF AUTONOMY.....185
AUTHORITY AND LEGITIMACY.....190
CONCLUDING REMARKS.....200
CHAPTER 10: CONCLUSIONS.....203
SUMMARY REVIEW OF RESULTS 203
EFFECT OF PRODUCT STRUCTURE ON GROUP DYNAMICS.....205
Decentralisation made scalable.....205
Modularity reinforces the emergent division of labour.....207
Effect of product modularity on labour productivity.....208
EFFECT OF GROUP DYNAMICS ON PRODUCT STRUCTURE.....209
Product structure mirrors organisational structure.....209
Product structure as coordination mechanism.....211
Why not in small-scale development conditions?.....213
EFFECT OF GROUP SIZE ON LABOUR PRODUCTIVITY.....214
Brooks' Law revisited.....214
GENERALISABILITY.....223

Across community of FOSS projects....223
Beyond the realm of FOSS....224
EPILOGUE....227
SUMMARY....239
SAMENVATTING (SUMMARY IN DUTCH)....245
APPENDICES....251
APPENDIX I: THE FREEBSD LICENSE....253
The FreeBSD Copyright....253
APPENDIX II: RELEASE RATE (1993-2003)....255
APPENDIX III: COMMITTERS ADDED AND REMOVED PER MONTH (2000-2003)....257
APPENDIX IV: CORE DEVELOPERS SURVEY....259
Email Questionnaire....259
Analysis....260
Results....260
Collected replies....261
APPENDIX V: BIBLIOGRAPHICAL REFERENCES....267
CURRICULUM VITAE....299

Acknowledgements

Science is by its very nature collaborative. The dissertation you now hold in your hands attests to this fact, as it would not have materialised without the cooperation of a group of individuals. What makes science – in particular, social science – inherently collaborative is not only the realisation, common among scientists, that its development thrives on practices of knowledge sharing, but also the necessity of crossing the boundaries of distinct cognitive fields. Social science is precisely that science which embraces and encompasses all fields of scientific inquiry in order to elucidate the general laws of their development and unify them in an intellectual structure which constitutes in a certain sense society's collective consciousness.

The 'genetic code' of my doctoral research mirrors that interdisciplinary character, combining concepts, theories and methods from fields as diverse as organisation studies, sociology, social psychology, software engineering and econometrics. I am conscious of my inability to grasp, in all its details and positive developments, any very large portion of human knowledge. The greatest intelligence would not be equal to a comprehension of the whole. Thence results, for science as well as for any other field of human activity, the necessity of collaboration between individuals with different backgrounds, skills and knowledges. Without that collaboration, this dissertation would have been impossible to realise. In the course of the doctoral research, I had the extreme luck to collaborate with a number of charismatic individuals, whose contribution is beyond the ability of words to convey. It was a pleasure to work with Ludo Gorzeman on data-mining FreeBSD's software repository and analysing activity logs, as was collaborating with Dr. Xander Koolman and Dr. Fardad Zand, who helped me understand the nitty-gritty of econometrics and provided me with expert consultation in the process of statistical analysis. My gratitude extends to Emiel Kerpershoek for helping me get started with econometrics, for providing feedback on every step of the research process, for illuminating the relevance of social psychology theories and experiments to my research and for being such a great roommate and friend over all these years. I am at a loss for words to express my intellectual debt to my promotor and daily supervisor, Prof. Michel van Eeten: not only was he involved in every stage of the research but, most crucially, he pointed out the way for me to become a better researcher and scientist, shaping my

notion of how empirical research in the social sciences ought to be done. A word of gratitude is also due to my co-promotor Prof. Hans de Bruijn for steering my research project in the direction in which it eventually crystallised and for motivating and challenging me to become a better researcher. Many thanks go to Dr. Alireza Abbasy, Adriana Diaz Arias, Hadi Asghari, Dr. Ruben van Wendel de Joode, Locutus, Dr. Efthimios Poulis, Vincent Verheijen, Webmind and my POLG colleagues at TBM – especially my peer group mates, Dr. Casper Harteveld and Harald Warmelink – for their feedback on various stages of the research. I am grateful to my PhD committee members, Prof. Johannes Bauer, Prof. Victor Bekkers, Dr. Matthijs den Besten and Prof. John Groenewegen, as well as FreeBSD developers Nate Lawson and Gleb Smirnoff for their feedback on the draft dissertation. I would also like to thank Wais Tekedar, my housemate in Den Haag, for his support and friendship.

George Dafermos
Den Haag, October 2012

Chapter Synopsis

Chapter 1 places the emergence of product modularity as a mechanism for combating the organisational problem of decreasing returns to scale in a historical and theoretical context.

In *Chapter 2* we review the literature of modularity as a design principle for complex product development and synthesise its alleged organisational benefits into a conceptual model, from which we draw hypotheses for subsequent empirical testing.

Chapter 3 describes the research methodology.

Chapter 4 introduces the empirical setting of the study: the FreeBSD Project.

Chapter 5 presents the results of testing the effect of modularity on coordination costs in FreeBSD (*hypothesis 1*).

Chapter 6 presents the results of testing the effect of modularity on group size and, reversely, the effect of increasing group size on modularity (*hypotheses 2* and *H2R* respectively).

Chapter 7 presents the results of testing the effect of modularity on labour productivity (*hypothesis 3*).

Chapter 8 presents the results of testing the effect of increasing group size on labour productivity (*hypothesis 4*).

Chapter 9 examines the transformation of FreeBSD's governance structure to which the project resorted in order to more effectively accommodate itself to expanding scale.

Chapter 10 sums up the empirical findings and reflects on the role of modular product design as a governance mechanism in the development of Free/Open

Source Software (FOSS) projects.

The *epilogue* comments on the effect that increasing organisational size exerts upon a group's ability to self-organise without centralised authority.

CHAPTER 1: INCREASING AND DECREASING RETURNS TO SCALE

INTRODUCTION

In recent years, modularity – a design principle implemented by breaking down a product into independent components, which can be developed autonomously without undercutting the functionality of the product as a whole – has emerged as a powerful solution to a classic organisational problem: the adverse effects of increasing scale on productivity known as decreasing returns to scale (Boehm 1981; Brooks 1995). Before we review the literature of modularity, in this chapter we will try to put the emergence of modularity into a historical and theoretical perspective by exploring the notion of increasing and decreasing returns to scale.

INCREASING RETURNS TO SCALE: THE ADVANTAGES OF BIGNESS

Arguably, no variable in organisation theory has garnered more attention than size (Daft & Lewin 1993, p. iii). The fascination that the size factor has exerted – and still exerts – for social scientists becomes easily understood once one considers the significance for economic growth that has been historically attributed to increasing returns to size. No illustration of the importance of increasing returns to the division of labour is better known than the oft-quoted passage from the first chapter of *The Wealth of Nations* (1776) where Adam Smith, writing at the threshold of the industrial age, points out that while a single worker, when working alone, can produce no more than twenty pins in a day, individual productivity rises up to four thousand eight hundred pins when the process is split up between ten workers, provided that each one of them specialises in a single task. The first systematic treatment of increasing returns to large-scale production, however, comes about sixty years later by which time the process of industrialisation was in full swing.[1] Charles Babbage, a computer pioneer and

1 In tracing the genealogy of ideas that fed the thrust toward bigness, our treatment overlaps with that of Rosenberg (1992, 1994).

inventor driven by the vision of 'the calculating engine', which was to occupy his lifelong labours, became thoroughly acquainted with contemporaneous developments in the industrial application of machinery. His studies culminated in a book entitled *On the Economy of Machinery and Manufactures* (1832), which, besides its illuminating descriptions of scores of industrial processes, offers a pioneering economic analysis of the factory. In the chapter on 'the division of labour', Babbage reminded his readers that to the three circumstances to which Adam Smith attributed the increased productivity springing from the division of labour – the increased dexterity of the individual worker, the saving of time that would be otherwise lost by switching from one task to another, and mechanical inventions – there must be added a fourth one:

> That the master manufacturer, by dividing the work to be executed into different processes, each requiring different degrees of skill and force, can purchase exactly that precise quantity of both which is necessary for each process; whereas, if the whole work were executed by one workman, that person must possess sufficient skill to perform the most difficult, and sufficient strength to execute the most laborious, of the operations into which the art is divided (Babbage 2009, pp.137-138).

According to Babbage, the chief advantage of the extension of the division of labour is that it permits an 'unbundling' of labour skills: by decomposing the production process into distinct tasks, and decoupling the tasks requiring skilled labour from those that do not, the former can be assigned to skilled workers and the latter to unskilled ones. Consequently, as the employer no longer needs to pay for labour corresponding to higher skill levels than those absolutely necessary for each stage of the process, production costs can be dramatically reduced. Equally important, the unbundling of skills can be carried very far: tasks into which the production process has been decomposed can be further decomposed into sub-tasks until there is no task in the production process that is complex enough for unskilled workers to perform. Following this line of reasoning, Babbage concluded that the drive to reduce production costs through such an unbundling of skills leads necessarily to the establishment of large factories. Babbage's treatment of the subject had a profound influence on two of the most prominent, perhaps *the* two

most prominent, economists of the 19th century, John Stuart Mill and Karl Marx. Drawing on the economic advantages of bigness that Babbage identified, Mill opens chapter 9, 'Of Production on a Large, and Production on a Small Scale', of his highly influential *Principles of Political Economy* (1848) by asserting 'that there are many cases in which production is made much more effective by being conducted on a large scale' (Mill 1965, p. 131). In particular, the benefit of expanding the scale of production is obvious

> when the nature of the employment allows, and the extent of the possible market encourages, a considerable division of labour. The larger the enterprise, the 'farther' the division of labour may be carried. This is one of the principal causes of large manufactories (Mill 1965, p. 131).

Following Babbage, Mill enumerates economies in the use of machinery, in operating costs like lighting, in management, and the 'economy occasioned by limiting the employment of skilled agency to cases where skill is required'. But if Mill was the first economist to call attention to the tendency for firms to expand in size due to economies associated with large-scale production, it was Marx who first stressed that the thrust toward large scale is irreversible and unstoppable. Because of the economies attendant upon increasing the scale of production, Marx was led to the conclusion that small firms cannot compete against larger ones and so, on a long enough timeline, gigantic firms are bound to dominate the market. As large-scale firms can produce the same products at lower cost, they can sell them at a lower price, thereby outselling their smaller-scale competitors in the market. Cut-throat price competition results in the absorption of the smaller firms by the bigger ones. In Marx's words:

> The battle of competition is fought by the cheapening of commodities. The cheapness of commodities depends, all other circumstances remaining the same, on the productivity of labour, and this depends in turn on the scale of production. Therefore, the large capitals beat the smaller (Marx 1990, p. 777).

In hindsight, it seems fair to say that Marx's predictions have not materialised. Small firms have not been eclipsed by larger – and because larger, more productive – ones. Marx failed to anticipate the disruptive effect of technological innovation, namely the changes in the organisation of the production process that the diffusion of the telephone and the electric motor were to catalyse from the end of the 19th century onwards. While the defective system of communication that antedated the telephone confined efficient administration to a single manufacturing site, and steam power – by reason of being more efficiently utilised in large units than small ones – fostered the tendency toward large industrial plants, the introduction of the telephone and the electric motor worked a transformation within the factory, imparting a great measure of flexibility to its design. As the engineers were no longer forced by the requirements of large steam engines to crowd as many productive units as possible on the same shaft, there was no point in centralising manufacturing. The displacement of steam power by electricity gave small-scale industry – as well as domestic production – a new lease of life, making it possible for small units to compete on even terms with larger ones (Mumford 1963, pp. 224-227).

Doubtlessly, the use of the telephone and electric motors gave small firms the requisite instruments to reach their full potential, enabling them to build up the *flexibility* on which their real strength actually rests. Although large firms might be well-suited to a stable and routine environment, their mode of operating renders them unsuitable for environments undergoing rapid changes. Operating through layers of management with rigid rules, they cannot match the flexibility offered by small firms, which is highly advantageous to experimentation in industries galvanised by disruptive change. As Rosenberg (1992) puts it:

> Many experiments are easier and less costly to conduct on a small scale. It is inherently difficult to experiment and to introduce numerous small changes, and to do so frequently, in a large hierarchical organizational structure where permissions and approval are required from a remote central authority.

The history of industries that have been recently undergoing radical technological change – such as the electronics and computer industry – attests to the fact that small firms have a comparative advantage in developing and launching

new technology products. Large firms are not receptive to the kind of risk-taking that is characteristic of smaller and leaner firms. In fact, risk aversion with respect to new technology is endemic to the structure of incentives in large organisations. By contrast, small firms, by cutting out the inevitable red-tape of even efficient large organisations, are well-positioned to experiment with respect both to technology and to form of organisation (Rosenberg 1992).

However, this flexibility would have been extremely limited in scope had not been for the possibility to draw upon a decentralised network of external capabilities – a practice nowadays known as outsourcing. A familiar path followed by small firms is that of specialisation. To increase their competitiveness, they opt to specialise in those activities at which they excel while outsourcing the rest to other firms. A good example of such *external economies* – a concept Alfred Marshall (1891, p. 325) coined to describe those economies that 'do not depend on the size of individual factories' but are 'dependent on the general development of the industry' – is the microcomputer industry. As established firms of the likes of IBM initially failed to appreciate the market potential for small computers for individual end-users, the early stages in the history of the microcomputer industry (better known today as the personal computer industry) are largely the story of enterprising hobbyists who fed on the capabilities of a large network of external sources to develop their own computers (Anderson 1984; Gray 1984; Hauben 1991; Stern 1981). Lacking the technical capabilities for producing in-house all the components they needed to build a personal computer, hobbyists banded together in user-groups (such as the legendary *Homebrew Computer Club* out of which emerged the distinctive culture of high-tech entrepreneurship that Silicon Valley is acclaimed for) and resorted to specialising in some components while outsourcing the rest. Had these hobbyists – and the start-ups they founded – not drawn upon a globally distributed network of capabilities, it would have been impossible to give flesh to their vision of 'computers for the masses'.[2] As Langlois (1992, p. 38) says, 'the rapid growth and development of the microcomputer industry is largely a story of *external economies*. It is a story of the development of capabilities within the context of a decentralized market rather than within large vertically integrated firms'. By allowing small firms to benefit from the economies in specialised skills

2 The Apple II (1982) illustrates this well: its stuffed boards were developed by GTC; its floppy-drives from Shugart and Alps; its hard-drives from Seagate; its RAM and ROM chips from Mostek, Synertek and NEC; its monitor from Sanyo. The only components that Apple developed in-house were floppy and hard-drive controllers, the power-supply and the case. See Langlois (1992, pp. 14-15, footnote 44).

and machinery that other firms have developed, external economies remove the necessity of increasing in size.

Profound changes in the structure of the global economy have also tended to favour the persistence of small firms. To a large extent, the persistence of the small firm is owed to the rapid expansion of the service industry since 1970, that is, to the shift of the labour force 'from manufacturing, with its relatively large establishments, to the service industry, with its small ones' (Granovetter 1984, p. 327). Indicative of the growth of services is that the proportion of U.S. private sector workers in services in 1982 rose up to 25.7%, overtaking that in manufacturing (25.5%). Considering that 'economies of scale in production show up for relatively small plants and that profit maximization does not generally dictate very large ones' (Granovetter 1984, p. 331), the declining share of employment in manufacturing – that is to say, the rising predominance of the service industry in the economy – implies that workers do not find themselves in increasingly larger organisational structures.

DECREASING RETURNS TO SCALE

Decreasing returns to scale due to coordination costs

Although Victorian economists commonly believed there is no limit to the division of labour within the firm, attempts to enlarge the scale of production were often checked by the tendency for coordination costs to rise. Not all writers of the 19th century were oblivious to this phenomenon, as shown, for example, by Amasa Walker's writings, who argued that the efficiency of supervision cannot be maintained beyond a definite scale of operations, setting thus a limit to firm size:

> When the concentration of capital has become so great that interested personal supervision cannot be brought to bear upon each department, and upon the whole enterprise, with sufficient intensity to insure efficiency and fidelity on the part of those employed, and harmony in the general conduct of the business. Beyond this point, the advantages derived from the power of concentration are neutralized (Walker 1866, chapter 5).

It was though not until the 1930s that economic theory turned to this question,

drawing attention to the limits to firm growth posed by diminishing returns to management. The contributions of Nicholas Kaldor (1934), Austin Robinson (1934) and Ronald Coase (1937) may be considered emblematic of this literature stream. According to Kaldor, the management function consists of two tasks: supervision and coordination. 'The essential difference between' them 'is that in the case of the former, the division of labour works smoothly' (Kaldor 1934, p. 69, footnote 1): while there are no limits as regards the number of individuals among whom the task of supervision can be split up, the nature of the coordinators' job on the contrary implies that they grasp the totality of productive processes. Coordinating ability, for the simple reason that humans are limited in their ability to process information, does not scale-up:

> You cannot increase the supply of co-ordinating ability available to an enterprise alongside an increase in the supply of other factors, as it is the essence of co-ordination that every single decision should be made on a comparison with all the other decisions already made or likely to be made (Kaldor 1934, p. 68).

A production system cannot be enlarged indefinitely without incurring increased costs of coordination and control required for the management of larger units. Consequently, these costs determine optimum firm size – that is, the limit to firm size. As Robinson (1934, p. 248) puts it:

> For every type of product there is in a given state of technique some size at which the technical and other economies of larger scale production are outweighed by the increasing costs of the co-ordination of the larger unit, or by a reduced efficiency of control due to the growth of the unit to be co-ordinated.

Enlarging the scale of production brings about 'diseconomies of co-ordination' (Robinson 1934, p. 252), which, in the final analysis, arise 'from the limitations of human abilities, from the fact that they can only think and organize at a certain pace, that they can only crowd so much work into twenty-four hours' (Robinson 1934, p. 247, footnote 1). In consideration of the limitations to the scale that can be

managed, 'an industrial body will be profitably enlarged only up to that point where marginal productivity is equal to average productivity' (Robinson 1934, p. 253).

It was in the context of this discussion that Coase advanced his theory of the firm, according to which firms exist because they provide an institutional environment within which transactions corresponding to certain tasks or stages of production can be coordinated more efficiently than in the open market. By implication, a firm shall grow up to the point that the cost of organising internally an additional activity exceeds the cost of carrying out this activity in the market or in another firm. To the question 'why is not all production carried on by one big firm?', Coase (1937, pp. 394-395) replied:

> As a firm gets larger, there may be decreasing returns to the entrepreneur function, that is, the costs of organising additional transactions within the firm may rise. Naturally, a point must be reached where the costs of organising an extra transaction within the firm are equal to the costs involved in carrying out the transaction in the open market, or, to the costs of organising by another entrepreneur. Secondly, it may be that as the transactions which are organised increase, the entrepreneur fails to place the factors of production in the uses where their value is greatest, that is, fails to make the best use of the factors of production.

Coase's transaction-cost theory explicitly acknowledges the primacy of costs commonly subsumed under the heading of management or coordination in determining the boundaries of the firm. However, a full consideration of the implications of this analysis had to await thirty years until Oliver Williamson, a student of Coase, expanded on the 'organisational failures' caused by increasing firm size. The first building block of his theory of institutional economics, which synthesises insights drawn from organisation theory and social psychology, was laid in 1967 when Williamson dissected the organisational implications of 'bounded rationality', that is, of 'human behaviour that is intendedly rational but only limitedly so' (Simon 1957). Given that bounded rationality results in finite spans of control, expanding the scale of operations implies that more hierarchical layers

have to be added: that is, the larger the scale of operations of a firm, the taller its hierarchy (Williamson 1985, p. 134).[3] The taller a hierarchy, in turn, the more prone it is to serious communication distortions, impairing thus the quality of the data transmitted upward as well as the quality of the instructions supplied downward, a phenomenon Williamson (1967, p. 135) calls 'control loss':

> For any given span of control...an irreducible minimum degree of control loss results from the simple serial reproduction distortion that occurs in communicating across successive hierarchical levels.

In the next years Williamson further elaborated on the factors responsible for limits to firm size. Besides communication distortions exacerbated by extensions of the hierarchical chain, he stressed the demotivating effects of working inside large firms. Bigness has negative 'atmospheric consequences': workers' feeling of alienation tends to grow in proportion with firm size. In parallel, as increasing firm size leads to taller hierarchies, 'leaders are less subject to control by lower-level participants' (Williamson 1975, p. 127). The larger a firm grows the more insulated and therefore the more opportunistic its managers tend to become: 'Efforts to tilt the organization, often through greater hands-on management, commonly result', as managers, perceiving themselves to be a separate group with its own goals, usurp the resources of the firm to further their personal agendas (Williamson 1985, p. 149). In sum, increasing firm size sets a limit to the incentives that the wage relation (i.e. the contractual employment relation) permits to be effectuated: as relative to small firms, the cost of tying rewards closely to individual performance is prohibitive for large firms. By emphasising the effect of increasing firm size on the behaviour of individuals, Williamson's work highlights the importance of considering reduced individual motivation, in addition to coordination problems, as a cause of decreasing returns to scale.[4]

3 'If any one manager can deal directly with only a limited number of subordinates, then increasing firm size necessarily entails adding hierarchical levels' (Williamson 1985, p. 134).

4 For an extensive review of Williamson's work as well as for an empirical test of his conclusions in a sample of 784 large US manufacturing firms, see Canback et al. (2006).

Decreasing returns to scale due to reduced individual motivation

Social psychologists have long been interested in the effect of working in a group on individual motivation and performance. The first experiment that found a negative effect of increasing group size on group performance was conducted by Ringelmann in the 1880s. Ringelmann observed that when groups of men pulled on a rope, tug-of-war fashion, their collective performance was inferior to the sum of their individual performances.[5] However, it was not until 1974 that his findings were replicated by Ingham et al. (1974), who ascertained that group performance indeed declined when more than one person pulled on the rope. More importantly, Ingham et al. (1974) attempted to separate the effect of coordination from motivation loss by asking subjects to pull in pseudo-groups, where they believed there were from one to five other members. Although they actually pulled alone, their (individual) performance was lower than when they believed they pulled unassisted by others, showing thus that the negative effect on group performance is due to reduced individual motivation, as distinct from coordination loss. Latané et al. (1979) arrived at the same conclusion in their highly influential 1979 experiment for which they asked college students to shout and clap as loudly as they could individually and in groups. Blindfolded and wearing headphones to mask the noise, students shouted and clapped in both real groups and pseudo-groups, where they believed they were part of a group but were on their own: individual performance dropped in both cases, demonstrating that reduced individual motivation was responsible for the decrease of group performance. For this demotivating effect, Latané et al. (1979) coined the term *social loafing*, which, as later studies have shown, generalises across tasks and *S* populations.[6]

However, this is not to say that social loafing is an inevitable side-effect of collective work. The tendency for people to expend less effort when working collectively is reduced or eliminated when individual outputs can be evaluated collectively; when working on tasks perceived as meaningful and engaging; when a group-level comparison standard exists; when working with friends or in groups one highly values; and when inputs to collective outcome are (or are perceived as being) indispensable (Karau & Williams 1993). In large groups, in particular, social loafing depends first and foremost on whether or not individual efforts are

5 The experiment was first reported in 1927 by Ringelmann's teacher, Walther Moede (1927). For a more extensive discussion of Ringelmann's experiment, see Kravitz and Martin (1986).

6 For a review of the relevant literature, see Karau and Williams (1993).

dispensable (or are perceived as such) (Kerr & Brunn 1983). This is the central thesis of Mancur Olsen's (2002) hugely influential treatment of collective action by voluntary associations. Drawing on public goods theory, Olsen's study of the conditions under which groups of individuals act in their collective interest led to conclusions diametrically opposed to group theorists who claimed that groups are mobilised by the consciousness of the collective goal to be attained. According to Olsen, while an individual is likely to contribute to a small group as he receives a large fraction of the total benefit or because 'his contribution or lack of contribution to the group objective [has] a noticeable effect on the costs or benefits of others in the group',

> in a large group in which no single individual's contribution makes a perceptible difference to the group as a whole...it is certain that a collective good will *not* be provided unless there is coercion or some outside inducements that will lead the members of the large group to act in their common interest (Olsen 2002, p. 44).

Since in a large group individual contributions do not have a discernible effect on the provision of the good and if the good is provided, being a collective good, nobody can be excluded from consuming it, Olsen concluded that when the latent group is composed of a large number of individuals, it would be rational for each of them to withhold their contribution:

> unless the number of individuals in a group is quite small, or unless there is coercion or some other special device to make individuals act in their common interest, *rational, self-interested individuals will not act to achieve their common or group interests* (Olsen 2002, pp. 1-2).

Simply put, individuals tend not to act in large groups because, on the one hand, they perceive their individual contribution to have no significant effect on whether or not the good shall be provided, while, on the other, they know that if it is provided, they cannot be excluded from using it. This non-act has come to be known as *free-riding*. Although Olsen's 'size principle' has been heavily criticised

on several grounds,[7] a substantial corpus of research supports the view that free-riding is caused by reduced identifiability and evaluation and hence is endemic in (large) groups where collective output is the only observable indicator of inputs.[8]

DOES PRODUCT MODULARITY MITIGATE THE ADVERSE EFFECTS OF INCREASING SCALE?

Despite the growing realisation that expanding the scale of operations beyond a certain point may decrease productivity through the overhead costs it entails, the fixation on expanding the scale of production has not waned. Characteristically, in his study of the rising industrial concentration in the household laundry equipment industry between 1947 and 1972, Scherer (1979, p. 195) remarked that 'there are unusually compelling economies of scale connected with both large annual volumes and large cumulative outputs of any given washing machine model', even though his own review of empirical studies of optimum plant size concluded that scale economies are exhausted beyond a relatively small size (Scherer 1970, pp. 72-103). On the same wavelength, in his study of 448 manufacturing industries, Miller (1978, p. 486) found 'compelling evidence of large economies of scale at the firm level for a major portion of American industry'. As in the vast majority of industries the productivity of the four largest firms was significantly greater than that of all other firms, Miller concluded that enlarging the scale of production (by constructing larger plants) results in higher productivity.[9]

Considering that attempts to boost productivity by enlarging the scale of operations are still in full swing, it should come as little surprise that there is a growing interest in how the adverse effects of increasing scale can be mitigated. The most promising perhaps of all technical solutions considered in this connection

7 For example, Chamberlin's (1974) critique is based on the role of the non-rivalness of consumption; Coleman's (1990) is based on the role of social networks; Gibson's (1991) on the role of social incentives such as fun; Goldstone's (1994) on tipping effects; Lohmann's (1994) on informational cascades; and Oliver and Marwell's (1988) on the jointness of supply.

8 For an economic treatment, see for example Holmstrom (1982). For a social psychology experiment, see Williams et al. (1981).

9 Miller's (1978) results were as follows: (a) in 409 out of 448 industries, 'on average the largest firms had an output per plant employee that was 39% greater than that for all other firms in the industry'; (b) in 400 out of 448 industries, 'on average the four largest firms had a value added per worker that was 37% higher than the remainder of the industry'; (c) in 431 out of 448 industries, 'on average the top four firms were able to handle 43% more material inputs per employee than the remainder of the industry'; and (d) in 369 out of 448 industries, 'on average, the four largest firms had profits per employee that were 57% greater than those for the remainder of the industry' (pp. 473-477).

is *modularity*: a design principle for managing complexity and reducing the need for coordination, implemented by breaking down a product into independent components, which can be developed autonomously without undercutting the functionality of the product as a whole. Stated in economic terms, product modularity is 'one very powerful technique...to reduce diseconomies of scale by reducing scale' (Boehm 1981, p. 194).[10] Specifically, it mitigates the adverse effects of increasing scale by reducing the need for communication and active coordination across the development of distinct product components. By attenuating the need for central coordination, modularity is held to impart scalability to the production system. This dissertation sets out to put this argument to the test by studying a phenomenon which combines both scale and modularity: free and open source software (FOSS) development. Its leading question is this: *Does modularity mitigate the adverse effects of increasing scale in FOSS development*?

In the next chapter, we delve more deeply into the literature of modularity, summing up its claimed benefits in research hypotheses conducive for empirical study.

10 As a side note, in the statement quoted Boehm seems to conflate scale diseconomies with decreasing returns to scale.

CHAPTER 2: LITERATURE REVIEW

THE PRODUCTIVITY PARADOX IN SOFTWARE DEVELOPMENT

How to speed up the development of large projects has long been a pressing question in the software industry. Past attempts to accelerate the rate of development by assigning more programmers to work on the project have often met with failure. Of them, the experience of IBM in the development of the OS/360 in the 1960s stands out for the legendary status it enjoys among software engineers. Responsible administratively for that programming effort was Frederick Brooks who, facing a project behind schedule, resolved to feed it with more programmers. The problem presented itself to Brooks in the shape of a dilemma well known among software developers:

> For efficiency and conceptual integrity, one prefers a few good minds doing design and construction. Yet for large systems one wants a way to bring considerable manpower to bear, so that the product can make a timely appearance. How can these two needs be reconciled? (Brooks 1995, p. 31).

However, rather than stepping up development, the additional inflow of programmers further derailed the project's schedule. Labour productivity decreased while the task of coordinating work flows became increasingly more difficult as more programmers joined the project. It did not take Brooks long to figure out why: Adding more developers to a project entails considerable organisational costs. First, freshly hired project members are not fully productive. They need to be trained by old developers, who, in taking on the mentor's role, channel part of their time away from their primary job responsibilities. Hence, not only are new developers not fully productive when they join the project, but in consequence of the training on the job given them by veterans, the productivity of the old-timers declines as well. Second, a communication overhead is incurred by adding more developers. The need to train and communicate with new members translates into additional

communication paths, thus increasing the complexity of communication in the project. As more developers join the project, the portion of the working day consumed in communication grows at the expense of the time devoted to product development. Consequently, the production process manifests decreasing returns on scale: productivity declines. In the light of these constraints, Brooks formulated his famous dictum: 'adding manpower to a late software project makes it later' (Brooks 1995, p. 25). Now commonly known as *Brooks' Law*, the adverse effect of increasing size on group performance is considered a ruling maxim of software engineering.

The root cause of the problem, as Brooks discovered, is that as new nodes are added to the communication network, the number of connections among them rises exponentially. This inevitably runs up against a limit beyond which the cost of adding one more node outweighs the expected benefit. Spreading out the work over too many participants could be counter-productive, short-circuiting communication channels and overloading a project's capacity to coordinate the contributions of participants. In the end, Brooks resorted to circumventing this division of labour problem by means of 'surgical teams' where 'one does the cutting and the others give him every support that will enhance his effectiveness and productivity' (Brooks 1995, p.32). The separation of high-level architectural design from the low-level task of code implementation, characteristic of this organisational configuration, aims at checking the communication overhead caused by enlarging the base of developers. Although these organisational costs are still operant, by decomposing the project into smaller sub-projects and assigning each to a surgical team, Brooks found an approximate way to balance the trade-off between speed of development and project staffing (Brooks 1995, pp. 35-37).

Considering that more than three decades have elapsed since the development of the IBM OS/360, it appears indeed a lasting insight of Brooks that a project's communication and coordination costs rise with the square of the number of participants (while the work done rises linearly). A comprehensive 1981 study of sixty-three software projects in the aerospace industry confirmed Brooks' assertion that the trade-off between men and months is far from linear (Boehm 1981). In 1989 Abdel-Hamid developed a system dynamics model of the software development process to put this thesis to the test. He found that 'adding more people to a late project always causes it to become more costly but does not always cause it to complete later' (Abdel-Hamid 1989). In his model, the schedule of the project suffers only when members are added during the final stages of

development. However, his results were criticised on methodological grounds for not taking account of sequential constraints between development tasks: according to Hsia et al. (1999), 'the maximum number of staff members depends upon the number of independent subtasks'. In 1996 a global survey of managers in software-related industries reported that increasing team size has a negative effect on productivity and development speed: firms with smaller teams of software developers tend to be faster and more productive, supporting 'the view that larger teams diminish productivity because of inefficiencies created by the difficulty of communicating within a large number of people' (Blackburn & Scudder 1996, p. 883). To the same conclusion points a 2006 study of 117 software projects which found that non-modular code increases the maximum team size, which, in turn, decreases productivity (Blackburn et al. 2006).

Meanwhile, efforts to enhance the flexibility of the practice of software development led to a more radical solution. The notion of modular programming, which gained currency with the development of the Unix operating system from the late 1960s onwards, envisaged a segmentation of projects into clearly defined tasks where each task is a program module and each module the responsibility of the programmer assigned to it (Raymond 2003). Its practice was given a strong impetus in 1972 by David Parnas, who established the definitive criterion for decomposing a software system into modules. According to Parnas (1972), decompositions based on flowcharts are inappropriate for large systems. Instead one should aim at minimising interdependencies among modules by hiding within a module information (such as design decisions subject to change) which should not be propagated to other modules. Encapsulated, that information cannot affect other parts of the system. This approach, like Brooks', attempts to constrain the presence of interdependencies in the development process, anticipating that (the development of a large software system is so complex that) many design decisions will have to be modified later in the course of production. But aside from that, the two approaches represent fundamentally different software development philosophies as well as different principles of organisation. For Brooks, programming was a 'public practice': he reckoned 'that exposing all the work to everybody's gaze helps quality control, both by peer pressure to do things well and by peers actually spotting flaws and bugs', which presupposes that developers have access to all parts of the software system so that they can test them, repair their defects and improve them (Brooks 1995, pp. 33, 271). By contrast, the principle of *information hiding* postulates that

> every module...is characterised by its knowledge of a design criterion which it hides from all others. Its interface or description [is] chosen to reveal as little as possible about its inner workings (Parnas 1972).

The underlying assumption, as Brooks (1995, p. 78) notes, is that 'the programmer is most effective if shielded from, rather than exposed to the details of construction of system parts other than his own'. The next twenty years, Brooks admitted in 1995, prove the effectiveness of Parnas' method in raising productivity and stepping up development (Brooks 1995, p. 272). By that time modularity had been established in the software industry as the dominant design principle for large projects.

MODULARITY IN ORGANISATION THEORY

These ideas were not foreign to organisation theorists, who, since the time of Frederick Taylor and Henry Ford, knew full well that task decomposition affords substantial productivity gains consequent upon the simplification of the labour process. In fact, from the 1950s onwards a current of ideas was developing at the intersections of general systems theory and organisation studies, preparing the ground for a general organisation theory of modularity. Emblematic of this tendency, Herbert Simon's work was fundamental in laying the foundations for a methodical study of modularity. Simon (1962) held that to analyse a complex system one must measure its degree of decomposability by distinguishing between interactions *within* subsystems and interactions *among* subsystems. Systems galvanised by strong interactions among their components are non-decomposable. Nearly decomposable, on the contrary, are those systems in which inter-component linkages are weak (though non-negligible). Arguably, a (nearly) decomposable system whose components can be removed and recombined without compromising its operation is more resilient to change than a system in which changing one component necessitates extensive changes in other components. The ability to mix-and-match components in different configurations vastly expands the design space within which the system searches for new solutions. Hence, as the fitness of complex systems is conditioned by their degree of decomposability, it is desirable to minimise interdependencies among subsystems by enclosing interactions (within

subsystems). Under the prism of Simon's analysis, *information hiding* – the encapsulation of interactions within subsystems – appears to be a principle of organisation crucial to all complex systems' ability to evolve. Its importance lies in effecting conditions of (near) decomposability. His discussion of the division of labour in the firm is characteristic: from 'the information processing point of view', he writes, 'division of labor means factoring the total system of decisions that need to be made into relatively independent subsystems, each one of which can be designed with only minimal concern for its interaction with the others' (Simon 1973, p. 270).

Needless to say, Simon was not alone in mapping out the new terrain. Toward the same direction pushed the contributions of many others, such as Alexander (1964), Ashby (1960) or Weick (1976), who dwelled on computer science concepts and turned them upstream. Of particular interest is the concept of *coupling*, which in computer science refers to the degree that a module depends on other modules. Weick (1976) introduced the concept in organisation studies to describe the relation of interdependence among the constituent parts of organisational systems, stressing the capacity for adaptation and innovation of loosely-coupled teams compared to the rigidity of tightly-coupled organisational configurations.

The next thirty years saw the gradual emergence of an organisation theory of modularity. Ideas long circulating within the streams of organisation theory were now given precise formulation. In 1992 Langlois and Robertson wrote that product modularity 'enlists the division of labor in the service of innovation...by allowing specialist producers (and sometimes specialist users) to concentrate their attention on particular components' (Langlois & Robertson 1992, p. 302). In the microcomputer and stereo component industries that formed the epicentre of their study, the adoption of modular product architectures set in motion a process of vertical and horizontal disintegration, promoting 'autonomous innovation, that is, innovation requiring little coordination among stages' (Langlois & Robertson 1992). In 1995 Garud and Kumaraswamy pointed out that in industries characterised by perpetual innovation and systemic products (that is, products composed of many components such that it is difficult, if not impossible, for any one firm to manufacture all of them), firms adopt modular product architectures to realise significant 'economies of substitution' by reusing existing components in developing higher-performance products. The same year Ulrich (1995, p. 437) underlined the significance of product modularity in enabling 'a bureaucratic approach to organizing and managing development', which 'allows the complexity

of the product development process to be dramatically reduced'. In 1996 Sanchez and Mahoney argued that product modularity is a key enabler of 'strategic flexibility': it allows production processes 'to be carried out *concurrently and autonomously* by geographically dispersed, loosely coupled development groups...thereby increasing the absorptive capacity of the firm' (Sanchez & Mahoney 1996, p. 70, emphasis in original). As production processes can be decoupled and performed by self-managing organisational units, product modularity

> can reduce the need for much overt exercise of managerial authority across the interfaces of organizational units developing components, thereby reducing the intensity and complexity of a firm's managerial task in product development and giving it greater flexibility to take on a larger number and/or greater variety of product creation projects (Sanchez & Mahoney 1996, p. 73).

According to Sanchez and Mahoney (1996, p. 73), a modular product architecture 'embeds coordination in fully specified and standardized component interfaces'. In this way, product modularity confers modularity on the development process. By definition, modularity is a form of product design using standardised interfaces among components to make up a decentralised system in which components are highly independent of one another (i.e. loosely coupled). In other words, the engineering concept of product modularity is devoid of meaning unless standardised interfaces are presupposed (Mikkola 2006). Sanchez and Mahoney conceptualise this point at a higher level of abstraction, contending that it is through the *embedded control* provided by standardised interfaces among components that hierarchical coordination is displaced:

> In essence, the standardized component interfaces in a modular product architecture provide a form of *embedded coordination*[11] that greatly reduces the need for overt exercise of managerial authority to achieve

11 *Embedded coordination* is defined by Sanchez and Mahoney (1996, p. 66) as 'the coordination of organizational processes by any means other than the continuous exercise of managerial authority'.

> coordination of development processes, thereby making possible the concurrent and autonomous development of components by loosely coupled organizational structures (Sanchez & Mahoney 1996, p. 64).

Sanchez and Mahoney's discourse is summed up in the argument that product modularity reduces drastically the need for coordination in the development of the components making up a systemic product, thus making possible their parallel and autonomous development. By implication, the 'strategic flexibility' stemming from the mitigation of coordination costs gives full scope to 'increasing the absorptive capacity of the firm' (p. 70), 'giving it greater flexibility to take on a larger number and/or greater variety of product creation projects' (p. 73). That is, product modularity imparts scalability to the production system.

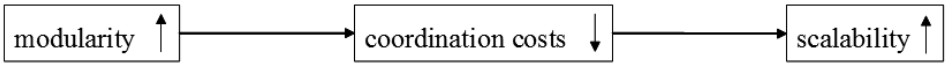


Fig. 2.1: General form of modularity thesis

Scalability means that the production system can enlarge in scale whilst retaining the advantages of organisational flexibility and efficiency peculiar to small-scale activity systems: size does not have to be accompanied by a high organisational price. To put it in terms consonant with Brooks' Law:

> Modularity enables *many* developers to work simultaneously on a project, while at the same time keeping integration and coordination costs low (Osterloh & Rota 2007, p. 160, emphasis ours).

The proposition that product modularity, by reducing coordination costs, allows a greater number of individuals to work on a project than would otherwise be possible of course implies that, given a sufficiently modular architecture, labour productivity in the project is not negatively affected by the expansion of the contributors' group, the effect of which is to speed up production. Osterloh and Rota's (2007) description of the function of product modularity in the development of free and open source software (FOSS) is exemplary of this line of reasoning:

> Because of modularity, the costs of the production of the source code are also kept low. A modular architecture invalidates “Brooks' Law” that “adding manpower to a late software project makes it later”. With a non-modular architecture, having more people involved in a project means higher coordination costs that can in the extreme case, render marginal returns of manpower to productivity negative. Modularization makes useful contributions possible with reasonable integration costs (Osterloh & Rota 2007, p. 166).

Osterloh and Rota's elaboration of the subject leads to the conceptual model illustrated in Fig. 2.2 below, which situates Sanchez and Mahoney's argument in the context of Brooks' Law:

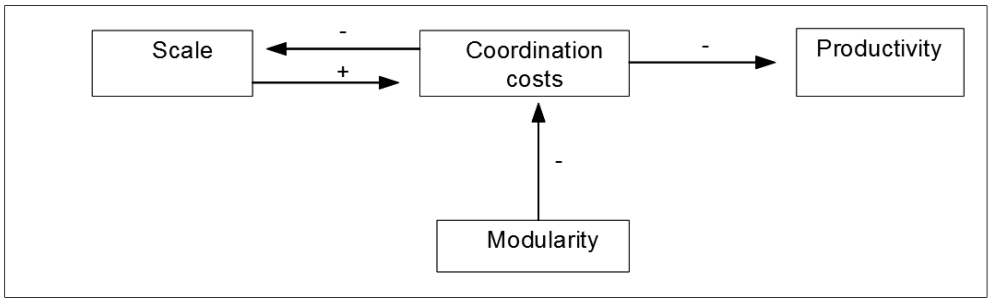


Fig. 2.2: Conceptual model

The intellectual synthesis of the organisational advantages of modular product design finds its most succinct expression in Sanchez and Mahoney's (1996) treatment, which represents the culmination of attempts at theory-building. Theory, however, needs to be substantiated by reference to empirical facts. Let us look more closely therefore at each of the hypothesised benefits of modularity within those streams of organisation theory that focus on their empirical demonstration.

Product modularity and coordination costs

The notion that product modularity reduces coordination costs in the production

process figures prominently in organisation theory. Historically, its roots can be traced back to Simon's work. In the *Architecture of Complexity*, Simon (1962) illustrates the benefits of decomposing a problem into parsimoniously linked sub-problems by using the example of watch-making. Partitioning the architecture of a watch into sub-assemblies allows Simon's hypothetical watchmaker to split the process of producing a watch into stages that can be completed independently, showing thus that the coordination burden, which is created by interdependencies between activities performed to achieve a goal, can be mitigated through architectural decompositions. For obviously, 'if there is no interdependence, there is nothing to coordinate' (Malone & Crowston 1990, p. 362). The emphasis on minimising task interdependence was not lost on subsequent organisation theorists who since have focused attention on partitioning development projects into tasks with that view in mind (e.g. von Hippel 1990). To their credit, decreasing task interdependence in a project has been found to reduce coordination costs and development time (Gomes & Joglekar 2008).

More than anything else, the staggering growth of global outsourcing since the 1980s gave widespread credence to the view that 'the visible hand of managerial coordination is vanishing', its function 'devolving to the mechanisms of modularity and the market' (Langlois 2003). On that point modularity theorists are in agreement, tracing the enabling condition for this industrial transformation to the 'embedded coordination' provided by 'design rules', that is, shared technical standards that effectively reduce governance costs (i.e. search, monitoring and enforcement costs) across the organisational network (Garud & Kumaraswamy 1995; Langlois 2003; Langlois & Robertson 1992; Sanchez & Mahoney 1996). By establishing a 'technical grammar' for collaboration, as Argyres' (1999) study of the development of the B-2 stealth bomber demonstrates, standardised component interfaces allowed the various 'subcontractors to work fairly independently...by "modularizing" the [B-2 design] structure around several' of its components. In this way, 'deep standardization'

> limited the need for hierarchical authority to promote coordination' and 'allowed considerable decentralization of design decision-making' which 'was possible because of the limited need for a central authority to assist in coordination efforts (Argyres 1999, pp. 162, 177).

Importantly, the phenomenon of the disaggregation of productive activities made possible by product modularity is not limited to cutting-edge technology projects (as the one studied by Argyres) but pervades entire industries. In the bicycle industry, for example, thanks to the bicycle's modular architecture, based on 'international standards that define how all of the components fit together to form an operational system...firms have had no real need to coordinate their activities or communicate with each other. With the level of coordination required to manufacture products being very low, market contracts have replaced active coordination, creating an industry made up of highly independent firms' (Galvin & Morkel 2001, p. 44).

Given the enthusiasm manifest in the writings of organisation theorists for the withering away of 'the visible hand of managerial coordination', it should come as no surprise that the mitigation of coordination costs through modularisation has come to occupy a prominent position in full-blown theoretical systems as in Baldwin and Clark's (2006a) modularity theory, which underscores three strategic aims of modularising a systemic product: *to manage complexity, to enable parallel development and encourage experimentation in the face of uncertainty*. In specific, modularity is 'tolerant of uncertainty' and 'welcomes experiments' because it allows 'modules to be changed and improved over time without undercutting the functionality of the system as a whole'. Parallel development occurs as 'work on or in modules can go on simultaneously'. And complexity is rendered manageable through the more effective division of cognitive labour that product modularity brings in its wake. In sum, the effect of splitting a systemic product into modules is to

> move decisions from a central point of control to the individual modules. The newly decentralized system can then evolve in new ways (Baldwin & Clark 2006a, p. 183).

Accordingly 'the new organizational structure imposes a much smaller coordination burden on the overall...endeavor' (Baldwin & Clark 2006a, p. 191). It becomes easily understood, of course, that this theorising is tenable to the extent that modularising a systemic product is presumed to effect conditions of decomposability among its components, thereby allowing their development to become independent from other components. Baldwin and Clark's approach, in particular, is built on the premise that dependencies among components can be

identified and eliminated through design rules (i.e. standards) and encapsulation. There is good reason why this is commonly assumed (especially in theory-building), for this is the ideal outcome of the modularisation process: a refashioned product that can be decomposed into independent components yet function together as a whole.

In practice though, this goal may prove elusive. One of the implications of a radically decentralised industrial structure regulated by standardised component interfaces is that making changes to the product architecture may not be feasible for any one organisational entity participating in its production. In the bicycle industry, for example, 'to change the crank pedal interface would require a supreme level of coordination and no firm is presently strong enough to be able to enforce such a change' (Galvin & Morkel 2001, p. 43). In fact, system-level changes, as opposed to component-level changes, are undesirable to the extent that they destroy compatibility between components (Galvin & Morkel 2001; Garud & Kumaraswamy 1995; Henderson & Clark 1990; Langlois & Robertson 1992, p. 302; Ulrich 1995). More importantly, early modularisations of a product design are often problematic on account of architects' imperfect (ex ante) knowledge of interdependencies that arise as the project unfolds.[12] Contrary to what modularity theory stipulates, an empirical study of seven IT organisations operating in industrial settings where 'interfirm modularity allows the products of different firms to work together in a decentralized system, often configured by the user', found to its astonishment that interdependencies were plainly ubiquitous (Staudenmayer et al. 2005). As interdependencies could not be sufficiently identified in advance or 'emerged throughout the product development process, despite efforts to limit them', managers resorted to dealing with them as they arose rather than trying to eliminate them outright. As a result, the managerial process was burdened with the cost of coordinating external relationships, the complexity of which imposed the creation of additional managerial posts (such as that of a 'relationship manager') as a focal point for coordination (Staudenmayer et al. 2005). The chaotic character of this development setting typifies a systemic product which, in spite of being split into distinct modules, is not decomposable. As dependencies among modules are not negligible, the need for coordination asserts itself.

In view of such cases, a growing body of the literature has come to criticise the

12 'Perfectly modular designs do not spring fully formed from the minds of architects' (Baldwin and Clark 2000, pp. 76-77).

proposition that product modularity reduces coordination costs.[13] In the aircraft engine and chemical engineering industries, Brusoni and Prencipe (2001) observed that the introduction of product modularity did not lessen the need for central coordination, the function of which was subsequently performed by *systems integrators*. In a follow-up study of the division of engineering labour in the chemical industry, Brusoni (2005) re-examined whether modularity at the product level brings about modularity at the level of the organisation that develops it. Again, he found that the modular architecture of chemical plants did not obviate the need for central coordination across the network of organisations engaged in their construction. The need to coordinate a distributed development process involving largely independent teams of specialists consolidated the position of systems integrators, rendering them necessary. In a subsequent study of tire manufacturing, Brusoni and Prencipe (2006) looked at the introduction of a modular manufacturing process at Pirelli Tires in the late 1990s. They found that '*modularization* at the product and plant level led to a process of *demodularization* and integration at the organization level...More specifically, it was integration within the knowledge domain that enabled the effective modularization of the technological domain' (Brusoni & Prencipe 2006, p. 186). As in their prior work, they concluded that one-to-one mapping between product and organisational structure is not possible when the locus of knowledge does not coincide with the partitioning of tasks as modelled on the product architecture. For the software engineers who joined the project, for example, it was impossible to develop the IT infrastructure for the manufacturing process without 'generating new connections among product and process engineers and across organizational units' (Brusoni & Prencipe 2006, p. 186). The need to comprehend and assimilate a diverse body of knowledge forced them to collaborate with other specialists such as tire designers. On the same wavelength, a study of changes in size and resolution of notebook computer displays in relation to the organisational design decisions made by notebook computer makers in the same period (1992-1998) found that 'modular products lead to more reconfigurable organizations' but not to 'shifting activity out of hierarchy' (Hoetker 2006, p. 513).

Summing up, although prior work in organisation theory has dealt with the issue of coordination costs in organisational networks based on modular product architectures, besides the use of such indicators as the coordinating role of

13 For an authoritative index of these 'revisionist' studies up to 2005, see the list of references in Ernst (2005).

intermediaries in the value chain (e.g. 'systems integrators' in the studies of Brusoni and Prencipe) or the frequent occurrence of communication across different organisational departments, no attempt has been made to quantify the effect of modularity on coordination costs. Qualitative indicators are no doubt useful to provide a rich description of the phenomenon under study, based on which hypotheses can be formulated, but notably less so for the purpose of testing hypotheses already formulated by prior research.

Product modularity and productivity

Several empirical studies have examined the impact of product modularity on productivity and organisational performance. An early study of fifty-seven car assembly plants worldwide found that the number of working hours required per vehicle increase in proportion to component interdependence (MacDuffie et al. 1996). Subsequent investigations confirmed the product modularity-performance link. In the home appliance industry, for example, Worren et al. (2002) found that product modularity, by increasing product variety, boosts performance. More recently, a study of fifty-seven North-American manufacturers of automotive components showed that product modularity has a pervasive organisational impact: it leads to cost reductions and improvements in product quality; it enhances the manufacturing system's capacity to handle product variety; it reduces development cycle-time through improved component availability and parallel manufacturing (Jacobs et al. 2007).

These findings lend support to the proposition that product modularity has a positive effect on organisational performance. Nevertheless, though it was in the bosom of the software industry that modularity was first conceived and employed as a method for the development of complex products, there is no empirical test demonstrating this claim in the context of a large-scale software project. Most of the studies available deal with projects developed by small groups, which are not encumbered with the organisational costs of large-scale collaboration. A comparison of two commercial projects by Cain and McCrindle (2002), for example, showed that the project with the lower degree of coupling among its modules was that which exhibited the higher labour productivity, but as none of the projects exceeded fourteen members, this needs to be tested and validated in projects featuring large-scale collaboration.

Product modularity and group size

The link between product modularity and group size was strongly emphasised in an empirical study of the modular re-design of the Mozilla Web browser, which concluded 'that different modes of organization are associated with [product] designs that possess different structures' (MacCormack et al. 2006). Prior to the re-design, Mozilla was developed by a closely-knit group of programmers on the payroll of Netscape Corporation. Then in 1997 Netscape released its source code for free under an open source license in an attempt to undercut competition by distributing production requirements across the network. A modular re-design was deemed necessary to harness the power of distributed development by a loosely-coupled network of volunteer developers scattered all over the world; it was motivated by the need felt for a product architecture conducive for large-scale collaboration over the Internet. In line with the project's expectations, 'the redesign to a more modular form was followed by an increase in the number of contributors' (MacCormack et al. 2006, p. 1028). Although the growth of contributors could be seen as reinforcing the centrality attributed to product modularity in catalysing new organisational structures, the authors, MacCormack, Rusnak and Baldwin, were careful not to overlook the possibility that the structure of the product *evolved* to reflect the production environment in which it was now being developed, the decisive factor of which was an expanding and geographically distributed base of contributors.

Paralleling these results, Crowston and Howison's (2006) analysis of bug-fixing interactions in 174 free software projects from Sourceforge, the GNU Savannah system and the Apache Software Foundation Bugzilla bug tracking systems, suggests that

> Small projects can be centralized or decentralized, but larger projects are decentralized...As projects grow, they have to become more modular, with different people responsible for different modules. In other words, a large project is in fact an aggregate of smaller projects, resulting in what might be described as a “shallot-shaped” structure, with layers around multiple centers (Crowston & Howison 2006, p. 81).

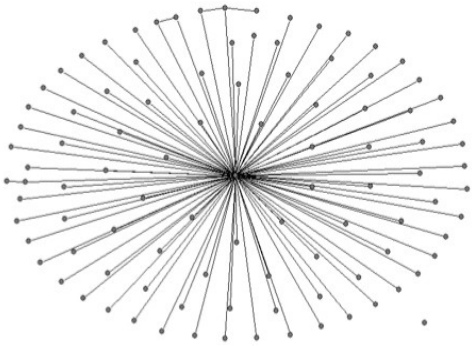


Fig. 2.3: Curl, a centralised project (Source: Crowston & Howison 2006)

Fig. 2.4: Squirrelmail, a decentralised project (Source: Crowston & Howison 2006)

The two plots above (*Fig. 2.3, 2.4*) illustrate this point well: one sees how bug-fixing interactions are clustered around a single central node in Curl, a small project, in contrast to Squirrelmail, a much larger project, where there exist multiple centres corresponding to distinct modules of the organisational system.

Unfortunately, though attesting to the link between group size and modularity, these findings do not go far in elucidating the dynamic of development, limiting thus our ability to draw conclusions. The main problem is that the time-perspective is missing. For instance, Crowston and Howison's analysis groups all interactions among bug-fixers over time in one static network, with the result that dynamic patterns in the data may be overlooked. Similarly, MacCormack et al.'s approach consists essentially in a comparison of different snapshots in time of the software's architectural structure – before and after its modular redesign – knowing in advance that the number of contributors increased perceptibly in that period. To apprehend how changes in product structure affect organisational structure and conversely how organisational changes feed back on product structure, an evolutionary approach based on longitudinal data is required, which is missing from the research literature.

However, aside from their methodological shortcomings, it is no coincidence that both Crowston and Howison's and MacCormack et al.'s work focuses on FOSS development. Drawing upon FOSS as a test-bed for generating and testing hypotheses is increasingly more characteristic of the research literature of modularity. That cannot be accounted for by the free availability of empirical data alone; rather, the emphasis on FOSS springs from the view that modularity is

immanent in the mode of informal and distributed development exemplified by FOSS projects.

STUDYING MODULARITY IN FREE AND OPEN SOURCE SOFTWARE DEVELOPMENT

The importance of a modular product architecture in shaping development dynamics is nowhere more pronounced than in the realm of free and open source software (FOSS). That is not accidental. A landmark event in the history of FOSS and software development more generally is the development of the Unix operating system, whose modular design, as aforementioned, paved the way for the wider adoption of modularity in the software industry. From an organisational perspective, its development was rather innovative: it used new tools, which enabled the application of new techniques;[14] it implemented new ideas and concepts; and it was built in a distributed fashion, owing to the geographical diaspora of its developers across different sites. For Ken Thompson, co-inventor of Unix, the choice of a modular design was dictated by the need to tame the complexity of the undertaking: 'Early Unix programmers became good at modularity because they had to be. An OS [operating system] is one of the most complicated pieces of code around. If it is not well structured, it will fall apart' (quoted in Baldwin & Clark 2000). The successful development of Unix showcased the power of modularity. And the central role Unix played in the software industry for the next three decades only affirmed it. However, it was not until the early 1990s that massively distributed development came into the foreground. The broad availability of consumer connections to the Internet revolutionised the scope for distributed development. Linux, an operating system kernel thriving on the volunteer contributions of a globally distributed community of software developers, was the first project that leveraged the network for this purpose. In 1991, its founder, Linus Torvalds, announced the project on the Internet, calling on the hacker community to join him in the development of a computer 'program for hackers by a hacker' (Torvalds 1991). The feedback was as massive as it was unexpected. Soon hundreds were contributing problem reports and modifications to the project. As the base of participants was rapidly expanding, the need to re-

14 Unix was special in several technical respects: it is perhaps best known for pioneering the use of the C programming language, which since has been diffused massively. Older operating systems were developed with assembly language.

design the software with a view to making it more modular was acutely felt. In the words of Torvalds (1999):

> With the Linux kernel it became clear very quickly that we want to have a system which is as modular as possible. The open-source development model really requires this, because otherwise you can't easily have people working in parallel. It's too painful when you have people working on the same part of the kernel and they clash...So once again managing people and managing code led to the same design decision. To keep the number of people working on Linux coordinated, we needed something like kernel modules.

Modularity, by eliminating dependencies among different parts of the system, allows developers to focus their work on any one module without having to worry about how that will affect or be affected by developers working on other modules, reducing thus the need for central control and coordination in the project. For Torvalds, on account of its function in the development process as a mechanism by which conflicts are tempered, a modular architecture was a precondition for Linux's parallel development. As he explains:

> Without modularity I would have to check every file that changed, which would be a lot, to make sure nothing was changed that would effect anything else. With modularity, when someone sends me patches to do a new filesystem and I don't necessarily trust the patches per se, I can still trust the fact that if nobody's using this filesystem, it's not going to impact anything else (Torvalds 1999).

Ever since Torvalds made these comments, it has been commonly accepted among FOSS developers that the open source development model requires a modular product architecture (e.g. O'Reilly 2001; Raymond 1999). Echoing this view, other practitioners like Jamie Zawinski, former leader of the Mozilla project, are no less categorical that a modular software architecture, by decoupling the work

of different groups of developers, effectively creates independent sub-projects, thus eliminating the need for coordination among them:

> Most of the larger open source projects are also fairly modular, meaning that they are really dozens of different, smaller projects. So when you claim that there are ten zillion people working on the Gnome project, you're lumping together a lot of people who never need to talk to each other, and thus, aren't getting in each other's way (Zawinski quoted in Jones 2000).

The presentation of modularity by the research literature on FOSS is no less panegyric. To give an example, Schweik et al. found a statistically significant positive correlation between the number of developers and project success in a sample of 107747 projects from the sourceforge repository, which they interpreted simplistically as support for the argument 'that the relatively flat, modular system of coordination in FOSS projects allows the addition of programmers without too many coordination costs' (Schweik et al. 2008, p. 424). Not many studies cast a critical doubt upon modularity's presumed moderating effect on the need for central coordination. An exception to the general rule is a study of the Debian project by Garzarelli and Galoppini (2003), who argue that 'the economies of substitution' realised by modular product design are not devoid of coordination costs. The Debian project, in particular, attempts to manage the uncertainty generated by product variation by standardising and formalising the procedure of selection and advancement of project members. In that sense, 'hierarchy in voluntary FS/OSS organization...[is] nothing more than the attempt to balance...the number of contributors and the number of software contributions accepted' (Garzarelli & Galoppini 2003, p. 34). Noteworthy is also Rusovan et al.'s analysis of the Linux ARP module, which, finding that 'the code is poorly documented, the interfaces are complex, and the module cannot be understood without first understanding what should be internal details of other modules', emphasised the potential maintainability and coordination issues caused by modularisations in which the principle of information hiding has not been properly implemented (Rusovan et al. 2005, p. 120). The difficulty of understanding and checking what the ARP module does without looking at the internals of other Linux modules implies that coordination costs in the development process are considerable and

'unless they have already become familiar with it, Linux TCP/IP code is difficult for even the most experienced programmers', limiting their ability to enhance and modify the software (Rusovan et al. 2005, p. 116).

For the most part, however, economic and organisational research in FOSS development has tended so far to view modularity as 'a technical and organizational way to manage complexity' (Osterloh & Rota 2007, p. 160), presuming that the modular architecture of FOSS explains to a large extent how a multitude of programmers, scattered all over the world, can collaborate on projects in which coordination through command-and-control hierarchies is conspicuously absent (e.g. Benkler 2006; Osterloh & Rota 2007; Raymond 1999; Weber 2004). Characteristically, Narduzzo and Rossi (2005, p. 90) contend that the modular architecture of Unix, which Linux inherited, enables parallel development and slashes coordination costs, as developers can 'carry out development of specific parts of the system in autonomy and without any need to coordinate their efforts with other sub-projects'. In consequence, it is because of modularity that productivity in the Linux project has not been negatively affected by the expansion of the group of contributors. Labour productivity falls in the wake of an increase of interdependencies. But since a modular design cuts down on interdependencies,

> A large number of participants in a project may be not a sufficient condition to generate dysfunctional effects, such as diminishing or negative marginal return of manpower to productivity... [since] the key aspect in this regard is represented by the degree of *task interdependency* between the various members belonging to the project...the high productivity...is largely due to the massively modularized structure of the... [Linux] project, enabling the existence of highly independent sub-projects joined by a limited number of developers (Narduzzo & Rossi 2005, p. 91, emphasis ours).

Narduzzo and Rossi's syllogism evinces a logic that is not hard to follow: as the number of individuals that could be simultaneously engaged in a project is a function of the degree of task interdependence in the development process, it follows that modularity is an enabling condition for large-scale collaboration without productivity loss. Development by such a large and geographically

distributed group is possible only because participants can contribute relatively independently of what others are doing in the project. A modular product architecture reduces dependencies among modules. By doing so it reduces dependencies among development tasks and, by extension, among developers. Productivity does not suffer with the expansion of the developers' group, as communication and coordination costs remain low. Under these conditions, the effect of adding more developers is to speed up production, thus raising productivity. In Langlois and Garzarelli's (2008) exploration of modularity in FOSS development, this line of argument is epitomised in full swing:

> A modular system increases the potential number of contributors; and the larger the number of collaborators working independently, the more the system benefits from rapid trial-and-error learning (Langlois & Garzarelli 2008).

Despite the fact that this conception of the role of modularity has come to characterise the full breadth of organisational discourse on FOSS, a careful review of the literature reveals no conclusive proof of the hypothesised moderating effect of product modularity on coordination costs. The claim that coordination costs are mitigated by product modularity is often treated as a self-evident axiom, rather than as an empirically testable proposition. In fact, empirical validation of the benefits of modularity is also lacking with respect to its effect on group size and productivity. Let us take a closer look at the empirical evidence for the benefits of modularity in FOSS development.

H1: Product modularity reduces coordination costs in FOSS projects

It is interesting that while this proposition has been discussed at length in organisational discourse for more than fifteen years, there is no record of a quantitative validation of the moderating effect of product modularity on coordination costs, nor of its falsification.[15] Although several studies have dealt with

15 The lack of scientific proof has not passed unnoticed. In their literature review, Gershenson et al. (2003, p. 307) noted that they 'have not found a single experiment to quantify or at least prove the claimed benefits of modular product design'.

the impact of product modularity on coordination, none has attempted to quantify the claimed benefit of modular product design. The sole exception is a recent study of the degree of collaboration among contributors to KDE, a large FOSS project, in the course of ten years of development by Capiluppi and Adams (2009) who tracked the communication paths among developers over time, weighting a communication path between any two developers according to the number of source code files on which they collaborated. They found that fewer than ten developers participated in the project's early stage of development, which was characterised by extensive communication within the group. Then, as the project started growing and the codebase was restructured with a view to increasing its modularity, 'communication compaction' (i.e. the average weight of path between developers) declined down to one third of its original value. In the last stage, when the number of participating developers exceeded three hundred, the compaction was still the same as when the project had no more than ten developers, that is to say, three hundred developers needed 'the same amount of communication as when the developers were only 10' (p. 274). Capiluppi and Adams (2009) qualified these findings by arguing that while hundreds contribute to large FOSS projects such as KDE, most of the work is actually done by a minority of high-contribution participants commonly referred to as *core developers.*

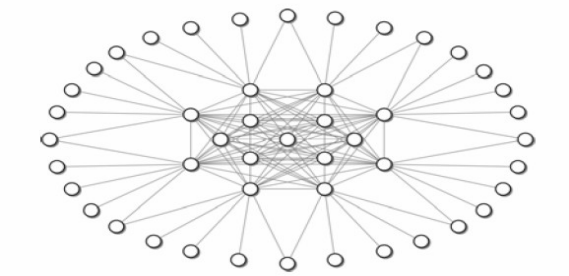


Fig. 2.5: The core-periphery structure of large FOSS projects

From this vantage point, the reason why Brooks' Law seems not to apply in the project is first because the number of core developers is such that the organisational costs of their collaboration (viz. the complexity of their interactions) do not become unmanageable; and second, because their activities are only loosely-coupled with those performed by the majority of 'low-contribution' developers. Whereas adding more developers to the core increases coordination costs, adding more developers to the periphery only increases the likelihood that bugs will be promptly identified

and fixed. That is to say, the positive correlation between scale and complexity holds only in the core but not in the periphery of the project (see also Lee & Cole 2003, p. 643). Different levels of coordination costs inhere in different development tasks and hence in different layers of the FOSS organisational structure. Because of low coordination costs involved in peripheral tasks, even 'low-ability' (that is, relatively unskilled and inexperienced) developers can contribute. And since they do not need to work as a close-knit team, problem-reporting and debugging are 'parallelisable': an infinite number of individuals can be simultaneously engaged in reporting bugs and fixing them: 'although debugging requires debuggers to communicate with some coordinating developer, it doesn't require significant coordination between debuggers. Thus it doesn't fall prey to the same quadratic complexity and management costs that make adding developers problematic' (Raymond 1999). What modularity does is simplify debugging by facilitating understanding of the internals of modules, as weakly-coupled components are easier to understand and thus easier to change and debug (Bernstein 2011). Put differently, the coordination costs involved are independent of group size. Yet this holds only for peripheral functions. The degree of collaboration required for the development of new functionality is significantly higher, and so are the respective coordination costs. In the light of this analysis, modularity is what allows FOSS projects to integrate a plethoric stream of minute contributions – in the form of problem-reports and fixes – without exacerbating the organisational costs of collaboration among core developers (Benkler 2006; Capra et al 2008, p. 769). Although that is without doubt an important perspective on the function of modularity in FOSS development, however by so qualifying their results, Capiluppi and Adams (2009) leave the question unanswered of how core developers are distributed across the increasing number of modules making up KDE and whether modularity mitigates the need for active coordination between distinct KDE modules and by extension between the developers working on them.[16]

16 An equally serious flaw in their work lies in the confusing, and at times contradictory, interpretation given to the results of their analysis. Consider, for instance, the results they report in a follow-up paper in which 'communication compaction' is phrased as 'coordination cohesion'. Here they find that 'in this first phase [in the development of KDE], fewer than 10 developers produce high cohesion scores, greater than 20' (Adams et al. 2009, p. 322). But when turning to the third and final stage of KDE's development, they mention that 'an apparent critical mass is achieved, requesting a coordination cohesion vastly larger than when found when the project had only 10 developers' (Ibid., p. 322) (indeed, by looking at the relevant plot in Fig. 2 in p. 323, one observes that cohesion rises from 20 up to 160 over time). This result, by showing that the volume of communication among developers rises over time, obviously contradicts their previous finding that communication compaction in the final stage is the same as in the first stage.

H2: Product modularity increases the potential number of contributors to FOSS projects

According to Sanchez and Mahoney (1996), product modularity increases the 'absorptive capacity of the firm': it imparts scalability to the development process. Seen in the context of Brooks' Law, the hypothesis holds that product modularity increases the potential number of contributors to a project without negatively impacting labour productivity. In the more theoretical strands of the research literature, the link between product modularity and group size has nowhere been attested to more emphatically than in a simulation study by Baldwin and Clark (2006b) of the interplay between code architecture and degree of participation in the development of FOSS:

> Projects not worth undertaking under a monolithic architecture may attract tens or even hundreds of self-interested developers under a sufficiently modular architecture (Baldwin & Clark 2006b, p. 1123).

Because changes can be made to specific modules without undercutting the functionality of the system as a whole, a modular architecture enhances the 'value options'[17] embedded in a codebase, as opposed to a monolithic (i.e. non-modular) architecture where option values are low, as changes tend to ramify throughout the system. In consequence, 'as the number of modules and the option values embedded in the system increase, more developers will work in equilibrium' (Baldwin & Clark 2006b, p. 1122).

Empirical backing for Baldwin and Clark's proposition is not lacking. Den Besten et al. (2006) examined ten large FOSS projects[18] spanning a period of five to

17 An *option*, according to modern finance theory, is 'the right but not the obligation to choose a course of action and obtain an associated payoff' (Baldwin & Clark 2006b, p. 1117). This conceptual instrument is used by Baldwin and Clark to model the value of modular product design upon the assumption that 'a new design creates the ability but not the necessity – the right but not the obligation – to do something in a new way...In this sense a new design is an option' (Ibid.). Thus, the analysis of value options in their work is geared to assessing the extent that the architecture of a systemic product encourages experimentation with regard to viable alternatives (i.e. substitutes) at the module-level. The same analytical approach can be found in Sullivan et al. (2001) and LaMantia et al. (2008).

18 The projects included in the analysis were: NetBSD, PostgreSQL, Apache, Mozilla, Gaim,

ten years of development to investigate whether collaboration is influenced by several characteristics of source code at the file level, and found that 'more modular code – here, more functions in files' – is associated with a greater average number of developers making changes per month, whereas more complex files attract less developers 'maybe because they induce a more exclusive selection of who could maintain a given piece of specially complex code' (den Besten et al 2006, p. 239). Nevertheless, these results do not constitute proof. The problem is that den Besten et al. conflate decent coding practice with modularity. In a sense, at file-level, the two could be considered the same, but in general the notion of modular code implies that modules contain several files that bear as little outward dependency as possible towards the rest of the code. This means that to examine modularity in the context of a software project, the scope of analysis should be at a higher level than the number of functions in individual files.[19]

The proposition that product modularity increases the potential number of contributors to a project is strongly supported by the findings of MacCormack et al. (2006) who examined the original Mozilla web browser (developed as a proprietary product by Netscape) and its evolution as an open source project after 1997 when it was redesigned with a strong focus on modularity. They found significant differences in their design structures: the redesigned Mozilla software had a markedly more modular structure. Moreover, in line with the project's expectations, the modular redesign was accompanied by an increase of contributors to the project. While this result seems to confirm the proposition that modular product design increases the potential number of project contributors, MacCormack et al. stressed that part of its explanation lies in the physiognomy of the FOSS development environment. FOSS development is distributed across a multitude of programmers scattered the world over, with limited or no possibility for face-to-face communication – the need therefore for a product architecture that facilitates coordination in a distributed, informal and virtual group is critical. Underlining the importance of considering the broader context of the FOSS development model when interpreting the relationship between Mozilla's software structure and the

OpenSSH, Python, GCC, Ghostscript and CVS.

19 A secondary criticism of den Besten et al.'s (2006) measurement of modularity could be advanced on the grounds that it is at odds with software engineering definitions of modularity that lay stress on minimising the number of functions per component. By taking the latter definition as point of departure, it could well be argued that 'more functions in files' indicate a less modular software system – rather than a more modular one as den Besten et al. perceive it. For a definition of modularity that puts emphasis on minimising functions per component, see for example Ishii et al. (1995).

number of project contributors, MacCormack et al. argued that while product modularity is required for distributed development by a large group, it is equally plausible that the design structure of the software *evolved* to reflect the environment in which it was now being developed, thus mirroring the organisational structure of the Mozilla development process. From this point of view, modular design is both a requirement and a consequence of the FOSS development model. This conclusion is reinforced by Capra et al.'s (2008) analysis of 75 FOSS projects (including large projects such as MySQL, Mozilla and OpenOffice), which highlighted the catalytic role of modular product design in enabling the governance structure typical of FOSS projects, whose informal and distributed character simultaneously acts as a catalyst for higher levels of code modularity (see also Capra 2008). Reinforcing the interpretation that the pattern of interactions among contributors, though driven by architectural design for the most part, determines – at least to some extent – the software system's dependency relations, a follow-up study by the same researchers compared five paired software products with similar function and level of sophistication, finding that 'larger, more distributed teams tend to develop products with more modular architectures' (MacCormack et al. 2008a, p. 2).

Arguably, by laying stress on how different modes of organisation are associated with product designs that possess different structures, the interpretation Capra et al. (2008) and MacCormack et al. (2006, 2008a) place upon these findings reverses the terms of the hypothesis so that the claimed direction of causality is from group dynamics to product structure:

> *An increase of contributors to a FOSS project results in an increase of modularity* (H2 reversed)

However, there are threats to the validity of this conjecture. The propagation cost metric that MacCormack et al. use to measure modularity provides a fairly accurate view of the complexity of the software as a whole, counting both direct and indirect (i.e. through a chain of dependencies across them) dependencies among files, but it does not distinguish dependencies within modules from dependencies among modules – that is, it does not take account of *clustering*: a modular design should minimise the interactions between modules more than interactions in general (Parnas 1972; Sharma & Yassine 2004, p. 40; Simon 1962; Wheeler 2007). Instead, it assumes that all dependencies between files, both direct

and indirect, incur the same cost, regardless of where the files are located or how long the path length is between them. That is clearly not a detail of minor importance, for the essential aim of modularising a system is to minimise interactions among modules by encapsulating them within modules (Ethiraj & Levinthal 2004, p. 161; Simon 1962; Sinha & Van de Ven 2005, p. 399). Because the criticality of the distinction between interactions *among* and *within* modules eludes their measurement method, MacCormack et al.'s analysis conflates the need for coordination among files – which the propagation cost reflects – with the need for coordination among modules, which ought to be the actual object of inquiry.

H3: Product modularity has a positive effect on labour productivity in FOSS projects

The argument that product modularity has a positive effect on labour productivity is often raised in organisational studies of FOSS development. According to Narduzzo and Rossi's (2005) study of modularity in the Linux project,

> the high productivity...is largely due to the massively modularized structure of the...project, enabling the existence of highly independent sub-projects joined by a limited number of developers (Narduzzo & Rossi 2005, p. 91).

Although this claim has been reiterated in other studies (e.g. Langlois & Garzarelli 2008), there is no empirical proof supporting it. Attempting to bridge this gap, increasingly more investigations turn to (the analysis of) software repositories such as version control systems (e.g. CVS), defect tracking systems (e.g. GNATS) and archived project communications (i.e. mailing lists) for sources of data amenable to quantitative measurements. A case in point is Giuri et al.'s (2008) analysis of 54229 FOSS projects from the Sourceforge repository, which showed that product modularity has a positive effect on the number of project members and labour productivity as captured by the sum total of contributions to the project (i.e. problem reports, patches and feature requests). At first glance, this result seems to provide a large-scale empirical demonstration of the benefits of product modularity in FOSS development. Its validity, however, is undermined by the methodological set-up of the study. Measuring a project's modularity by counting its number of

modules as Giuri et al. did is inadequate, for it does not take modules' degree of coupling (i.e. their interdependence) into account. The problem is that a modular system, understood in the limited sense of a system composed of smaller sub-systems, is not necessarily decomposable: it is not the number of modules that determines the system's degree of modularity but their degree of coupling (Frenken 2006, p. 303; Langlois 2002, p. 22). Giuri et al.'s measurement of productivity is equally problematic for our purposes: it misses the point that the variable of interest is the *returns to scale* exhibited by the production process. What needs to be estimated is not the sum total of contributions to the project but the extent to which an increase in inputs (i.e. contributors) results in a less than proportionate increase in outputs (i.e. contributions) (Banker 1984; Banker et al. 1994; Banker & Slaughter 1997; Robinson 1934). If the marginal returns of an additional unit of input are below the average returns, average productivity is decreasing – that is, the production process exhibits decreasing returns on scale. Conversely, increasing returns on scale prevail when average productivity is increasing. In short, the key variable is *average productivity*. To recap: nothing, based on their data, can be said about the effect of modularity on group size or productivity. What their results actually suggest is that products made up of a large number of components tend to be developed by larger groups than those with fewer components.

For more than thirty years, product modularity has been employed by software engineers as 'one very powerful technique...to reduce diseconomies of scale by reducing scale' (Boehm 1981, p. 194): 'for example, if a software project's size [is] in the region of decreasing returns to scale, a manager could choose to divide the project into several smaller projects in order to increase the productivity' (Banker et al. 1994, p. 275). The cause of decreasing returns to scale is well known: communication path increases, complex interface requirements, project slack. As is their effect: productivity plummets, product development gets bogged down. Yet in spite of the consensus among FOSS theorists and practitioners that product modularity, by mitigating the adverse effects of increasing scale, increases productivity, empirical confirmation of this benefit is still wanting.

CONCLUDING REMARKS

The literature makes a compelling argument: modular product design increases the potential number of individuals that could work on a project and has a positive effect on their labour productivity because it allows them to work independently of

the activities performed by one another in (different parts of) the project, with little or no need for central coordination. But there are gaps in the literature: these are concentrated not so much in the theory as in the want of empirical demonstration. Even the most penetrating works (e.g. MacCormack et al. [2006]) are not longitudinal but static. To capture forces of causality at work, we need to examine how these factors co-evolve over time. In order to study these relationships in a more rigorous manner, we need a case that covers a time span in which scale has increased dramatically. Moreover, the level of analysis adopted by prior work, in its overwhelming majority, is that of the project as a monolithic (i.e. integrated) organisational entity: for instance, MacCormack et al. (2006) look at how Mozilla's degree of code modularity affects the aggregate number of project contributors. In consequence, the effect on the level of the modules making up the project is insufficiently explored, though it is precisely at the level of modules – by enabling their independent development by autonomous groups of developers – that the organisational impact of modularity is considered to be most significant.

The next chapter describes the research methods we use in the present study to examine how modularity affects coordination costs, group size and labour productivity in the context of FOSS development.

CHAPTER 3: RESEARCH METHODOLOGY

ANALYTICAL FRAMEWORK

As our review of the literature in chapter 2 demonstrates, what is missing from the literature is a longitudinal study covering a time-span in which scale has increased dramatically so that the relationship between product modularity, coordination costs, group size and productivity in the development process can be examined in a rigorous fashion.

Research Design

The present dissertation adopts a case study research design, a research strategy used to understand dynamics within single settings (Eisenhardt 1989, p. 534). Case studies can include single or multiple cases and multiple levels of analysis (Yin 1984). In our case, we study a single case, the FreeBSD project, which we analyse both at the system-level (i.e. project-level) and component-level (i.e. module-level): thus, in addition to probing the effect of the degree of modularity of the entire FreeBSD codebase on the total number of developers with commit rights in the project and their labour productivity, we examine how the degree of modularity of individual modules affects the number of committers contributing to them and labour productivity specific to the development of these modules.

It is customary for case studies to use different data collection methods (e.g. archives, interviews, surveys, observations), resulting in evidence that is qualitative or quantitative or both (Eisenhardt 1989). For this study, we use archives of development activity logs and project communications, questionnaires and observations, which provide us with both qualitative and quantitative results. In specific, we use quantitative methods to analyse archives of activity logs and project communications, complemented by qualitative evidence gleaned from observations, questionnaires and relevant literature.

Case studies can serve different purposes: they can be used to describe a phenomenon, to test or generate theories (Eisenhardt 1989). Although more frequently used to generate hypotheses, the strength of case studies is not limited to theory-building. For example, Anderson (1983) used a single case study – the

Cuban missile crisis – to test which theory of decision making best explained the actual decision making process followed in that setting. Similarly, our motivation is to test a theory, according to which product modularity, by mitigating coordination costs in the development process of a software project, increases the potential number of individuals that could work on the project and their labour productivity (because they can work independently of one another without needing to coordinate their activities).

We test this theory in the context of FOSS development on account of its *uniqueness*: for it is precisely the uniqueness of the empirical setting which 'permits particular insights that allow one to draw inferences about more normal firms' (Siggelkow 2007, p. 21). Among the characteristics setting it apart from conventional organisations is that FOSS projects are neither profit-driven, nor are participants bound to these projects by a contractual employment relationship. Consequently, the function of management is modified by the volunteer nature of participation, for as participants are not paid to work on FOSS, command-and-control is virtually non-existent in this environment: developers undertake tasks as their interests best dictate, that is to say, decision making labour (i.e. administrative tasks) is not decoupled from the labour of execution (i.e. performance tasks). The reason we study the role of modularity in this setting is because modularity is held to be required for the decentralised mode of FOSS development.

Individual cases suffice to test and falsify a general theory, thereby spurring further research and justifying more refined conceptualisations (Siggelkow 2007). Compared to large-sample empirical works, a central advantage of this approach is that it allows for delving deeply into constructs, enabling thus the researcher to illustrate causal relationships more directly. That, of course, is fundamental in the context of longitudinal research like ours 'that tries to unravel the underlying dynamics of phenomena that play out over time' (Siggelkow 2007, p. 22).

Object of investigation

The object of our investigation is the interplay between product modularity, coordination costs, group size and labour productivity in the development process of large FOSS projects. FOSS comprises a large and sprawling ecosystem of software projects. Empirically, we focus on the case of the FreeBSD project, for reasons we outline below. According to the research literature, the interrelation between these factors can be modelled as follows, where the direction of the arrows indicates the

hypothesised directionality of effects:

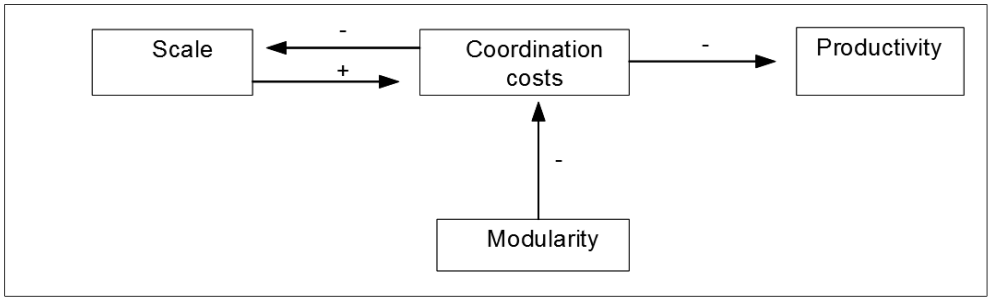


Fig. 3.1: Conceptual model

However, the data on coordination costs we were able to collect were not sufficient to run a regression analysis.[20] As a result, the effect of modularity on coordination costs as well as the effect of coordination costs on scale (proxy-measured by the number of active FreeBSD developers) and productivity could not be tested statistically. Confronted with this constraint, we set out to test these relations directly, as the research model in Fig. 3.2 below illustrates:

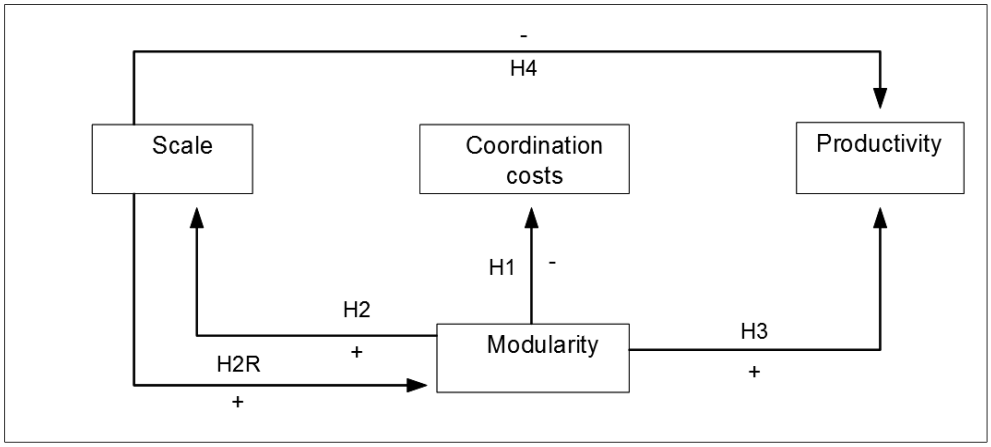


Fig. 3.2: Empirical model

Specifically, the research model consists of the following hypotheses:

20 For an extensive discussion of this problem, see chapter 5.

#	Hypothesis
H1	Product modularity reduces coordination costs in FreeBSD
H2	Product modularity increases the potential number of contributors to FreeBSD
H2R	An increase of contributors to FreeBSD results in an increase of modularity
H3	Product modularity has a positive effect on labour productivity in FreeBSD
H4	An increase of contributors to FreeBSD has a negative effect on labour productivity

Level of analysis

The analysis presented in this study draws upon both qualitative and quantitative results. In line with prior studies of modularity in FOSS development (e.g. Baldwin & Clark 2006b; MacCormack et al. 2006), we carry out a qualitative analysis of descriptive statistics based on activity logs collected from FreeBSD's software repository to examine how the degree of modularity of the codebase as a whole relates to the total number of committers working on the project, their labour productivity and coordination costs in the project. In addition – and this is where our study deviates from prior works that focused exclusively on the relationship between product modularity, group size and productivity at the level of the entire project – we perform a quantitative analysis of the hypotheses at the level of the components (modules) making up FreeBSD because it is only at that level that the effect of the degree of interdependence of the modules on the number of contributors and their productivity can be rigorously examined. In specific, we use regression analyses to test how the degree of modularity of the individual modules making up FreeBSD affects the number of committers contributing to these modules, their labour productivity and the coordination costs specific to the development of these modules. To sum up, we look into the hypotheses at two levels: the qualitative component of our analysis examines the hypotheses at the project-level, while the quantitative analysis focuses on the module-level.

The *regression analyses* that comprise the quantitative analysis are described in the section **Statistical analysis** towards the end of this chapter. The *indicators* of

modularity, coordination costs, group size and productivity used in the analysis are described in the sections **Measuring modularity**, **Measuring coordination costs**, **Measuring developers group size** and **Measuring labour productivity** respectively.

WHY THE FREEBSD PROJECT?

FreeBSD is a Unix-like operating system derived from the Berkeley Software Distribution (BSD), the version of Unix developed at the University of California, Berkeley. Its development begun in 1992 when 386BSD split into two versions, FreeBSD and NetBSD, as a result of developers' frustration with the pace at which 386BSD incorporated patches. Since, FreeBSD has been established as the most popular BSD-descendant with a proven track record in mission-critical deployments. Nowadays, the project thrives on the volunteer contributions of a community of developers spread the world over.[21]

Of all FOSS projects, why did we choose FreeBSD? Our selection criteria were the following: a project characterised by:

(a) large scale (as reflected in a large base of developers);
(b) a modular product architecture and
(c) available logs of development activity.

The first two criteria are derived from the literature review: we wanted to find a product which is (or held to be) modular and which is developed by a large group of developers. The last criterion is purely methodological: empirical data had to be accessible too. FreeBSD met all these criteria:

(a) it is currently developed by a group of about four hundred individuals (known as committers) vested with the right to integrate changes in the project's code repository;
(b) the software is partitioned in different modules[22] and
(c) it has publicly-accessible logs of its development activity dating back to

21 For an elaborate description of the FreeBSD project, see chapter 4.

22 FreeBSD can be characterised as modular in two ways. First, on account of its modular code architecture (i.e. independently of its degree of decomposability, the product is partitioned in distinct modules); and second, it is modular from the end-user point of view, as it is up to the users to decide which modules of the operating system to load at runtime according to the functionality they need.

1994.

The next four sections elaborate on the methods we use in this study to measure modularity, coordination costs, group size and productivity through activity logs collected from FreeBSD's software repositories.

MEASURING MODULARITY

The study of modularity formally begins with Parnas' (1972) design principle of *information hiding* as the definitive criterion for decomposing a system into modules. The concept was elaborated by subsequent research, which developed metrics to assess the degree of *coupling* between modules and the degree of *cohesion* within modules (Dhama 1995; Selby & Basili 1988; Stevens et al. 1974). In general, attempts to measure software modularity focus on the level of coupling between system components, following either of the following two directions. They either analyse specific types of dependencies between components, as for example the number of function calls (Banker & Slaughter 2000; MacCormack et al. 2006; Rusovan et al. 2005) or global variables (Feitelson et al. 2007; Schach et al. 2002; Yu et al. 2006). Or, alternatively, they infer the existence of dependencies between components by assessing whether they tend to be modified at the same time. In the present study, modularity is estimated by analysing *function call dependencies*.

Studying modularity as function call dependencies typically revolves around the application of a Design Structure Matrix (DSM),[23] a system modelling technique widely used in software engineering to outline the structure of a design based on the information exchange and dependency patterns between its constituent elements. The technique has been used in prior research to compare different architectures (Sosa et al. 2007) and assess the degree to which design interfaces and team interactions are aligned (Sosa et al. 2003, 2004). The application of the DSM begins with specifying the unit of analysis. In characterising software structure, that can be the *directory* level, which corresponds to a group of source files pertaining to a specific subsystem; the *source file,* which corresponds to a set of programming instructions performing a related set of functions; or the *function* level, which corresponds to a set of programming instructions performing a specific task. Our analysis focuses on the source file level for the following reasons. First, the tools

23 DSM also stands for Dependency Structure Matrix – we use both terms interchangeably. The technique was pioneered in the early 1980s by Stewart (1981).

used by developers for version control (e.g. CVS) use the source file as the unit of analysis: that is to say, developers keep track of the software system's development by examining changes at the source file level. Second, source files are the locus of development, as tasks and responsibilities are commonly allocated to developers at the source file level. Source files, therefore, constitute a logical unit of analysis for studies of software evolution (MacCormack et al. 2006).

There are several types of dependencies between source files. In the present study we focus on one important dependency type: the *function call*. A Function Call is an instruction requesting the execution of a task. If the function called is not located within the source file where the request originates, then a dependency exists between two source files. For example, if FunctionA in SourceFile1 calls FunctionB in SourceFile2, then SourceFile1 depends on SourceFile2. The dependency is marked in location (1,2) in the DSM.[24] To capture function calls, we input a product's source code into a tool called Call Graph Extractor. Function calls can be extracted statically or dynamically. Here we use a static call extractor, *cscope*,[25] because it uses source code as input, does not depend on the state of the software and captures its structure from the programmer's perspective.

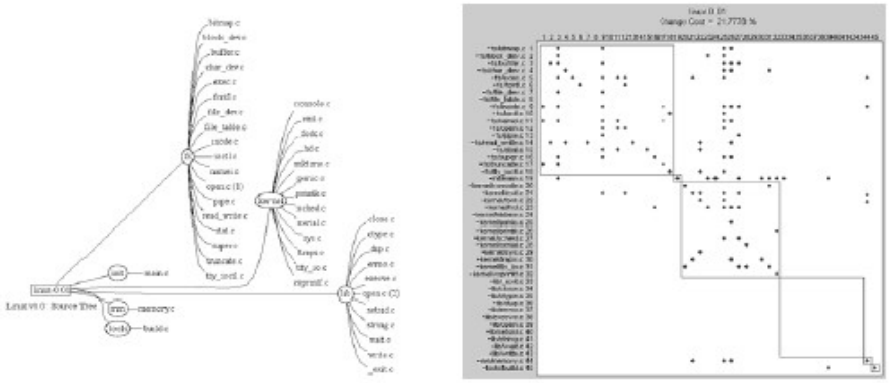


Fig. 3.3: The directory structure and architectural view of Linux (Source: MacCormack et al. 2006)

The function call dependencies thus extracted populate the DSM, which can be visually examined from the software system's architectural view. The architectural

24 For DSM entries, the standard convention is used (row number, column number).

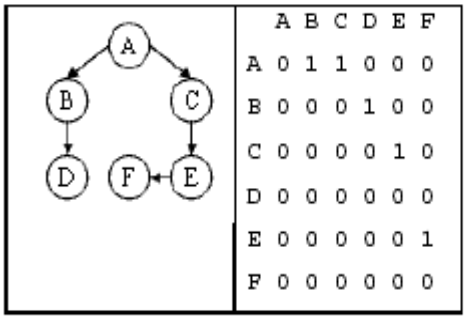
25 Cscope is available online at <http://cscope.sourceforge.net>. In addition, we wrote custom Perl scripts and a small C program to extend the functionality offered by Cscope.

view of a DSM groups each source file into a series of nested clusters defined by the directory structure, with boxes drawn around each successive layer in the hierarchy. The result is a map of dependencies, organised by the programmer's perception of the design. For the purpose of illustration, Fig. 3.3 above depicts the Directory Structure (left) and Architectural View (right) for Linux. In the architectural view, each dot represents a dependency between two source files.

To assess the impact of modularity at the level of the components (modules) making up FreeBSD, we develop three empirical proxies for the degree of modularity:

(1) component visibility (as captured by the *propagation cost* metric);
(2) the number of external dependencies of components and
(3) the ratio of internal dependencies to external dependencies of components (referred to as *integrality index*). We discuss each in more detail below.

First, we measure *visibility* (Sharman & Yassine 2004) by assessing the number of both direct and indirect dependencies that a module has. We characterise therefore the structure of a design by measuring the level of coupling among its components based on the degree to which a change in a system element (i.e. file) causes a potential change in other system elements directly or indirectly (i.e. through a chain of dependencies across them).



	A	B	C	D	E	F
A	0	1	1	0	0	0
B	0	0	0	1	0	0
C	0	0	0	0	1	0
D	0	0	0	0	0	0
E	0	0	0	0	0	1
F	0	0	0	0	0	0

Fig. 3.4: Example system in graphical and DSM form

To illustrate, consider the hypothetical system depicted in the form of a diagram and a DSM in Fig. 3.4 above. We see that element A depends on (that is, calls functions in) elements B and C, hence a change in C could have a direct effect on A. In turn, element C depends on E, hence a change in E could have a direct effect on

C but also an indirect effect on element A with a path length of two. Likewise, a change in F could have a direct effect on E and an indirect effect on both C and A with path lengths of two and three respectively (In this system there are no indirect dependencies between elements for path lengths of four or more).

The calculation of external dependencies is derived directly from the DSM. For example, element A has two direct and three indirect external dependencies. To assess the *visibility*[26] of any one element, we use the technique of matrix multiplication: we raise the DSM to successive powers of N. Then we sum these matrices together to derive the *visibility matrix V*, which shows the dependencies, both direct and indirect, existing for all possible path lengths.[27]

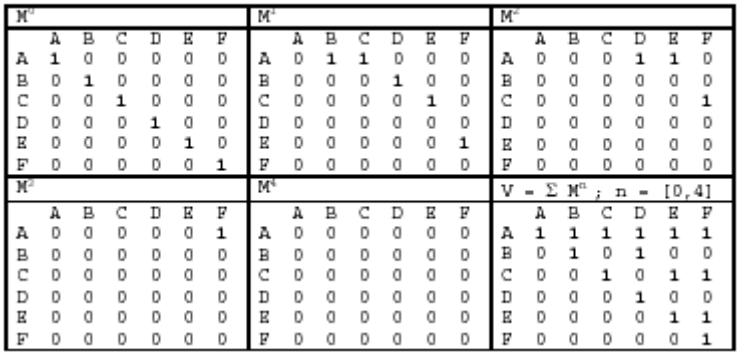


Fig. 3.5: The derivation of the visibility matrix

The visibility matrix displays the dependencies between all system elements for all possible path lengths up to the maximum – determined by the size of the DSM (denoted by N). The visibility measures are derived directly from it. To measure the (direct and indirect) dependencies that flow out of a component, we calculate the *Fan-Out Visibility* by summing along the rows of the visibility matrix and dividing by the total number of elements. The higher the fan-out visibility of an element the greater the number of elements it depends on. Visibility can be alternatively measured by calculating the *Fan-In Visibility,* which measures the dependencies that flow into a component and which is derived by summing down the columns of the visibility matrix and dividing by the total number of elements. The higher the

26 I.e. the more visible an element is in the system the greater the number of elements that depend on it.

27 The visibility matrix is referred to elsewhere (e.g., in Sharman and Yassine [2004] and Warfield [1973]) as reachability matrix.

Fan-In Visibility of an element the greater the number of elements that depend on it. To continue with the same example as before, element A has a Fan-Out visibility of 6/6th (or 100%), meaning that it depends on all other system elements, and a Fan-In Visibility of 1/6th, indicating that it is visible only to itself, meaning that no other elements depend on it. In the present study, we measured visibility by calculating fan-out visibility.[28]

To summarise these data for each module, we compute the density of every module's visibility matrix (i.e. the DSM multiplied by itself to fill in all the indirect dependencies), referred to as *propagation cost* – a metric which intuitively captures the percentage of files that are likely to be affected on average when a change is made to a randomly selected file (MacCormack et al. 2006; Milev et al. 2009). Specifically, the propagation cost computes the average Fan-Out and Fan-In Visibility of all elements (which are identical, as for every fan-out there is a corresponding fan-in). In our example, the propagation cost can be calculated from Fan-Out Visibility as (6+2+3+1+2+1)/6*6=42% or by using Fan-In Visibility as (1+2+2+3+3+4)/6*6=42%.

In addition to calculating each module's propagation cost, we assess the propagation cost of FreeBSD as a whole, which we use as an indicator of modularity for the aggregate development process of the full product. Considering that the individual development of modules is embedded within the development process of the entire project, the latter indicator allows us to examine the extent to which the dynamic of development of individual modules is affected by the complexity of the broader production environment. Another proxy we use for modularity at the project-level is the *ratio of external dependencies to the number of modules* making up the software. Because external dependencies increase as of logical necessity when new modules are added to the software product,[29] examining their growth in

28 It is open to debate which of the two methods of computing visibility – fan-in or fan-out visibility – results in a more robust indicator of the extent of coupling of product components. A recent empirical study by MacCormack et al. (2008b, p. 20) showed that 'fan-in visibility is more dominant [than fan-out visibility] in explaining survival' of modules across successive versions of the software, which suggests that fan-in visibility is a more reliable indicator. However, other recent works have found that estimating visibility through fan-out yields results of a higher explanatory power: for example, von Krogh et al. (2009, p. 26) showed that 'the effect of in-degrees [number of components that use the focal component] is tiny compared to the effect that out-degrees [number of components used by the focal component] exert, trumping it by a factor of about 20'.

29 In spite of the moderating effect of modularity, the addition of new modules to a software product is bound to result in new external dependencies, as new modules would still need to interact to some extent with (viz. use functionality embedded in) pre-existing modules.

relation to that of modules makes for a more refined indicator of modularity than just measuring the number of external dependencies independently of the number of modules contained in the product. The metric is also theoretically derived, namely from Simon's (1962) theorisation of complexity as characterised by two factors: the number of parts in a system and the interconnections or interdependencies between these parts. Simon (1962, p. 468) defined a complex system as follows:

> By a complex system I mean one made up of a large number of parts that interact in a nonsimple way. In such systems, the whole is more than the sum of the parts, not in an ultimate, metaphysical sense, but in the important pragmatic sense that, given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole.

In discussing how complex systems can be described, Simon treats the concept of complexity as synonymous to task interdependence: postulating, that is, that the complexity of a problem-solving process can be estimated by measuring the degree of interdependence between the distinct tasks comprising the process. Following conceptually Simon, we use the ratio of external dependencies to modules to capture the degree of component (module) interdependence.[30]

Component interdependence = external dependencies / modules

Further theoretical grounding for using this metric to capture interdependence comes from the seminal work of Kauffman and Levine (1987) on performance measurement in complex systems. Their *NK model* counts the number of modules in a system (*N*) and measures the degree of interaction (i.e. interdependence) among modules (*K*): so that the lower *K* is compared to *N*, the more independent are the modules comprising the system. Conversely, when *K* is high compared to *N*, modifying a module is likely to affect other modules, often in dysfunctional ways,

30 It is evident that by the same logic, the ratio of external dependencies to modules could be employed as an indicator of complexity for the development process of the project as a whole. By following Simon's logic of description, in fact, task interdependence and complexity are but different names for describing the same relation of interdependence (among the constituent tasks/stages of a process or the components of a system).

in which case the system is non-decomposable.[31] Our measure of component interdependence is operationalised in the same manner: a low ratio of external dependencies to modules indicates a modular system and vice versa.

Besides the propagation cost and the number of external dependencies, we use one more proxy for modularity at the component level: that is the ratio of external dependencies to internal dependencies, which we call *integrality index.*

Integrality index = external dependencies / internal dependencies

The rationale for using this ratio as a measure of modularity is derived from theory: according to Parnas (1972), the definitive criterion for decomposing a system into modules is encapsulation (i.e. information hiding), meaning that interactions among system components are to be eliminated through their encapsulation within modules. As Sharman and Yassine (2004, p. 40) explain, the goal of modularising a system is to find modules or clusters of system elements

> that are mutually exclusive or minimally interacting. This process is referred to as *clustering*. In other words, clusters contain most, if not all, of the interactions internally and the interactions or links between separate clusters are eliminated or minimized. In which case, the blocks become analogous to team formations or independent modules of a system (i.e. product architecture).

To use Simon's (1962, p. 477) formulation, the act of decomposing a system into modules is intended to have

> the effect of separating the high-frequency dynamics – involving the internal structure of the components – from the low frequency dynamics – involving interaction among components.

31 Originally proposed for the study of biological systems' evolution, the NK model has since been extensively applied to the analysis of technological and social systems. For three recent works of organisation theory using the NK model or one of its variants, see Brusoni et al. (2007), Ethiraj and Levinthal (2004) and Siggelkow and Levinthal (2003).

In short, a successful modularisation implies that interactions (i.e. dependencies) have been *localised* into modules.[32] It follows directly from this description of the operational logic of modularity that a characteristic of modular systems is that the internal dependencies of modules (i.e. interactions within modules) well exceed their external dependencies (i.e. interactions among modules). The opposite would indicate a non-modular (i.e. monolithic) system. Hence, the higher the ratio of external dependencies to internal dependencies the less modular the system. That is why we named the metric *integrality index*: because the higher the ratio, the more monolithic (integrated) the system. And conversely, the lower the ratio of external dependencies to internal dependencies the more modular the system.

MEASURING COORDINATION COSTS

Although prior work in organisation theory has dealt with the issue of coordination costs in environments characterised by modular product architectures, it has proven to be very difficult to measure the effect on coordination costs quantitatively. Indicatively, in their study of the introduction of a modular production process in a tyre manufacturer, Brusoni and Prencipe (2006) interpret the frequent occurrence of communication flows across different departments as an indicator of high coordination costs. Similarly, in examining whether product modularity moderates the need for central coordination across a network of organisations engaged in the construction of chemical plants, the emergence of a new actor on the network – personified by systems integrators – is taken for evidence of high coordination costs (Brusoni 2005). Although such indicators (as the communication linkages among distinct functional departments of an organisation or an actor taking on the specialist role of coordinating an inter-organisational network) are not without scientific merit, their imperviousness to quantification limits decisively their explanatory power. Despite the need for quantitative measures of coordination costs, they are by and large missing from prior works in this research field. To our knowledge, only two studies have attempted to quantify coordination costs in a software development project by tracing communication paths among developers over time and weighting a communication path between any two developers according to the number of source code files on which they collaborated (Adams et al. 2009; Capiluppi &

32 For a more elaborate discussion of the principle of *interaction locality* (also known as dependency locality), see Yu et al. (2009) and Yu and Ramaswamy (2009).

Adams 2009).

In the present study, we measure coordination costs through the volume of communication among developers in the project. In specific, we count the number of emails sent over mailing lists used in the project for coordinating the development process. For the task of measuring coordination costs, our choice of metrics and data sources is dictated by project-specific considerations. As mailing lists constitute the primary communication fora in FreeBSD (FreeBSD 2011b; Watson 2006), the number of emails exchanged by developers is the most direct measure of coordination costs in this setting. A minor complication is that, as the project uses a multitude of mailing lists,[33] each geared to different aspects of the project, identifying the one(s) centred on coordinating development processes is crucial. By reviewing the relevant literature, we were able to identify the *freebsd-current* mailing list as the central forum for coordination issues related to the current branch. According to FreeBSD researchers Holck and Jørgensen (2004),

> For developers working on *CURRENT*, the mailing list freebsd-current is particularly important, as this is where all announcements of important changes to *CURRENT* will be given. Also, problems in building or running *CURRENT* will be posted to and discussed in this forum; these seem to account for around 75% of the list threads.

The formal description of the *freebsd-current* mailing list by the FreeBSD (2011b) project is as follows:

> This is the mailing list for users of FreeBSD-CURRENT. It includes warnings about new features coming out in -CURRENT that will affect the users, and instructions on steps that must be taken to remain -CURRENT. Anyone running "CURRENT" must subscribe to this list. This is a technical mailing list for which strictly technical content is expected.

To make sure that *freebsd-current* is in fact centred on coordinating

33 As of August 2011, there are 144 public mailing lists in use (<http://lists.freebsd.org/mailman/listinfo>)

development processes, we selected in random 100 emails sent over the list during a time-period of five years (from March 2003 to March 2008). Their majority (i.e. approx. 70%) relates to coordination costs triggered by changes in the product such as, for example, integration breakdowns ('broken build') caused by product modifications; problem-reports and suggested problem-solutions (i.e. bug-fixes) that need to be reviewed and tested by more developers before they can be incorporated into the (official version of the) product; or modifications rendered necessary or desirable by changes in the broader technological environment such as the development of new hardware. To illustrate, consider the four emails below, which appertain to: (i) a problem-report; (ii) another problem-report; (iv) a fix ('patch') designed to solve a problem, which needs to be tested and reviewed by more developers and (iii) yet another problem-report, which, to be fixed, requires coordination with the group working on the *usb* module.

Subject: pear broken on current.
Sent by: eculp at bafirst.com eculp at bafirst.com
Date: Sat Jun 4 10:51:02 GMT 2005

In /usr/src/UPDATING, there was a change:

20050528: Kernel parsing of extra options on '#!' first lines of shell scripts has changed.

and documented at: http://people.freebsd.org/~gad/Updating-20050528.txt

After a week of rebuilding, changing versions of pear, php, apache and all other dependencies and looking everywhere except at this change, Finally, thanks to Manfred Antar <null at pozo.com> and Thierry Thomas <thierry at freebsd.org>, I was able to understand that this was my problem with pear but I still don't know what the solution is. I assume that the port will need to be changed or am I missing something?

Thanks,
ed
P.S. I have submitted a PR

Subject: lockmgr panic on shutdown
Sent by: Doug White dwhite at gumbysoft.com
Date: Sat Nov 1 17:25:27 PST 2003

I can confirm the lockmgr panic on shutdown reported by someone else earlier (whose message I mistakenly deleted).

It looks like swapper is trying to undo a lock from pagedaemon and runs into trouble. This is probably related to the Giant pushdown of vm_pageout() that alc did last week.

I'm building with INVARIANTS to see if that will catch more info. Will report back soon.

Subject: usbd not opening all usb busses for event watching.
Sent by: John Baldwin jhb at FreeBSD.org
Date: Mon Jun 20 20:49:43 GMT 2005

On Thursday 26 May 2005 12:42 am, Darren Pilgrim wrote:
> I appear to be running up against the hard-coded limit of four usb hubs in
> usbd. The problems were devices not attaching properly. The machines in
> question are a new notebook with four USB 2.0 ports and an older desktop
> with onboard USB 1.1 and a USB 2.0 card. The notebook produces 5 hubs and
> the desktop produces 7.
>
> The problems disappeared after I increased MAXUSBDEV to match the number of
> hubs present. This isn't really a bug, so I wasn't sure if send-pr was
> appropriate. Should I file a PR for this?

Actually, it does sound like a bug. :) I would file a PR and then post a message with the PR to usb at FreeBSD.org as that is the list of folks who look after the USB code.

Subject: ULE Interactivity perf patch
Sent by: Jeff Roberson jroberson at chesapeake.net
Date: Fri Dec 19 06:30:11 PST 2003

```
I realized a pitfal in the way that I'm doing slice
assignment for interactive tasks.  I'd like to have as many
people test this as possible, in case there are unintended
consequences.  What this patch does is allow interactive
tasks to have longer time-slices so that they may be more
efficient.

This patch is intended to fix the poor performance of some
interactive processes while under high load, especially
high load with other interactive tasks present.

http://www.chesapeake.net/~jroberson/interact.diff

Thanks,
Jeff
```

As the above emails demonstrate that the communication occurring on the mailing list is related to the coordination of tasks in the FreeBSD development process, we capture total coordination costs in the project through the indicator of the volume of emails sent over the *freebsd-current* mailing list.[34] For the quantitative analysis, we capture the coordination costs involved in the development of each module through the indicator of the volume of emails sent over the mailing lists centred on their development. To illustrate, the *freebsd-firewire* mailing list focuses on the development of the *firewire* module,[35] the *freebsd-usb* mailing list focuses on the *usb* module[36] and so forth.

MEASURING DEVELOPERS GROUP SIZE

Estimating how many individuals contribute to the development of a FOSS project like FreeBSD is rather straightforward, though definitely more complicated in comparison with a corporate software development environment where detailed

34 The freebsd-current mailing list is archived online at <http://lists.freebsd.org/mailman/listinfo/freebsd-current>
35 The freebsd-firewire mailing list is archived online at <http://lists.freebsd.org/mailman/listinfo/freebsd-firewire>
36 The freebsd-usb mailing list is archived online at <http://lists.freebsd.org/mailman/listinfo/freebsd-usb>

information typically is readily available about the exact number of individuals working on a project, their degree of participation (i.e. full-time or part-time employment) and the nature of the tasks assigned to every one of them, that is, the division of labour in the software project. By contrast, it follows from the predominantly volunteer and fluid character of participation in FOSS projects that a completely different approach is required.

A method used in several studies to estimate the number of individuals contributing to a FOSS project is by examining so called *credit files* (e.g. Bowman 1998; Koren 2006; Moon & Sproull 2000; Tuomi 2004). It is a common trait of FOSS projects that contributions are credited – for instance, in a credit file – as prescribed by community etiquette. The method we use in the present study is similar. In the FreeBSD project, all developers vested with the privilege to commit changes to the code repository – known as *committers* – are listed in a file, which is updated whenever a person is granted commit privileges, becoming thus a committer, or when a person's commit rights are revoked.[37] Therefore, as the credit file reflects changes in the composition of the committers group, we count the number of individuals listed in the file in order to estimate the number of committers in the project. In addition to using the credit file, we measure the number of *active committers*, that is, those committers who practised their commit rights by actually making changes to the codebase, through activity logs in the project's software repository (i.e. CVS). In that way, by subtracting the number of active committers from the total number of committers listed in the credit file, we can measure the extent of free-riding within the group, thus allowing us to further refine our measurement of group size.

Contrasting the number of committers listed in the credit file with the number of active committers over time (in Fig. 3.6 below) shows that the extent to which free-riding occurs has not increased with the passage of time due to the expansion of the group (indicatively, according to the credit file, the number of committers increased from 105 in 1997 to 224 in 2007). In other words, the historical enlargement of the committers group has not resulted in an increase in the number of free-riders (as predicted by Olsen's [2002] 'size principle'). In fact, by looking at the ratio of active committers to all committers in Fig. 3.7 below, we see that the proportion of free-riders in the group decreases over time.

37 The file listing FreeBSD committers is accessible at <http://www.freebsd.org/cgi/cvsweb.cgi/CVSROOT-src/access>.

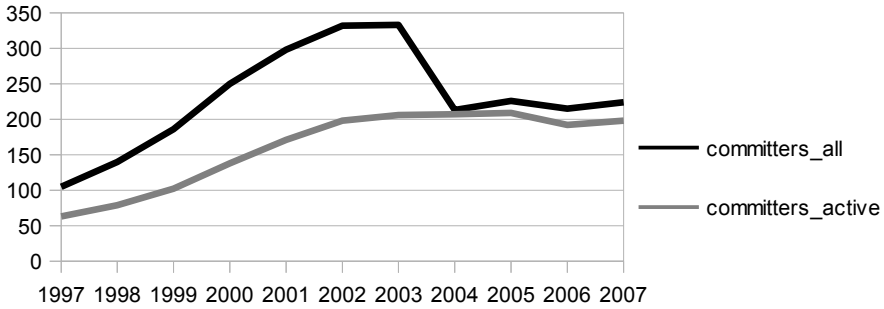


Fig. 3.6: Free-riding in FreeBSD committers group

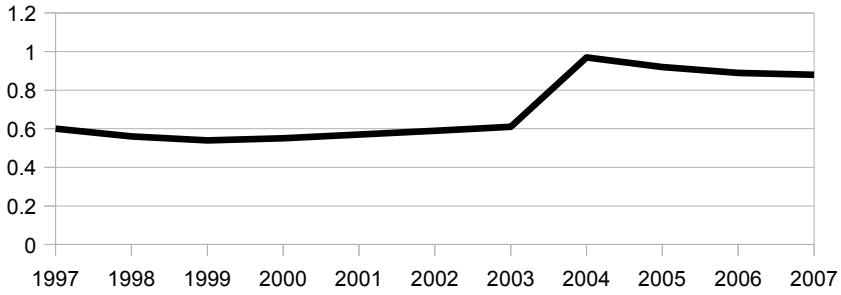


Fig. 3.7: Ratio of active committers to all committers

The advantage of our measurement method, compared to other alternatives, as Ghosh (2003) explains, is that it permits

> a more detailed and less biased (but also less formal) method of author attribution [which] is used by developers themselves during the development process. Either through a version-control system, such as CVS or Bitkeeper, or simply through a plain-text "ChangeLog" file, changes are recorded between progressive versions of a software application. Each change is noted, usually with some identification of the person making the change — in the case of a version control system this identification, together with the date, time and size of change is more or less automatically recorded.

On the other hand, our method can be criticised on the grounds that

> most projects limit to a small number the people who can actually "commit" changes, and it is their names that are recorded, while the names of the actual authors of such changes may or may not be (Ghosh 2003).

The thrust of this criticism is that large FOSS projects like FreeBSD thrive on the contributions of a multitude of individuals, not all of whom have the right to integrate changes to the project repository. Thus, those who do – the so-called committers – are responsible for reviewing and committing the modifications sent them by those without commit rights. But as the repository only logs the names of the committers rather than the originating contributors, some committers might appear to be considerably more active than they really are, given that a single person (with commit rights) might be credited for contributions originating in others. The problem, remarks Ghosh, is that basing an analysis of authorship on activity logs from project repositories could lead to measurement errors and ultimately to erroneous conclusions. Ghosh is right to draw attention to distortion effects that the analysis of activity logs from software repositories might entail. Yet author attribution is not a problem that admits of no solution. It would have been a serious problem, had the right to commit been treated as a privilege to be defended against newcomers. But that is not so. Granting commit rights is an integral part of the process by which one joins a FOSS project and advances from peripheral, though necessary, activities such as problem-reporting and problem-fixing to the development of new functionality. As Michael Lucas (2000), FreeBSD committer, puts it: 'if you submit enough useful and correct PRs [that is, problem-reports with fixes attached], eventually some committer will get sick of taking care of your work and will ask you if you want to be able to commit them yourself'. Lucas' description of the process by which commit rights are granted conveys a crucial point: commit rights are granted to those who make more than just occasional contributions to the project. It follows from the character of the recruitment process that committers are engaged extensively in code development as well as that anyone making a substantial contribution is given commit rights. Consequently, it is safe to assume that there are very few, if any, high-contribution participants in the FreeBSD project outside of the committers group. That, of course, suffices for our primary purpose, which is to examine how the work of regular contributors is affected by

modular product design.

MEASURING LABOUR PRODUCTIVITY

Productivity in software development projects has been traditionally measured, as in most industries, as the amount of output of the production process per unit of input used.[38] Characteristically, the IEEE (1993) defines software productivity as 'the ratio of units of output divided by units of input'. In this context, software size is typically used as the output of the production process and effort as the input. Thus, by measuring the size of the produced software and the effort required to produce it, productivity can be computed as follows:

$$Productivity = Size / Effort \qquad (1)$$

All models by which productivity in software projects is measured are based on this definition, though somewhat different variants of the above equation have been used in studies attempting to capture productivity from different angles.[39] To measure size, the most commonly used metric is source lines-of-code (LOC) (Boehm 1981; den Besten et al. 2006; Blackburn & Scudder 1996; Blackburn et al. 2006; Curtis et al. 1988; Spinellis 2006). Effort is typically measured in working hours, days, months or years expended in the production of the software. In that way, productivity can be computed as the number of LOC divided by some unit of labour time (Boehm 1987), as for example:

- LOC per man-hour (Spinellis 2006; Walton & Felix 1977)
- LOC per man-month (Blackburn & Scudder 1996)
- LOC per total man-months (Blackburn & Scudder 1996)
- LOC per man-years (Cain & McCrindle 2002; Cusumano & Kemerer 1990)

However, in FOSS projects it is impossible to measure effort directly, as in that setting the volunteer (i.e. unwaged) character of participation makes it impossible to estimate directly the number of working hours expended by contributors. For

38 For an informative introduction to the topic based on a summary of software productivity measurement studies, see Scacchi (1995).

39 For example, *functional productivity* has been calculated as the amount of functionality (in function points) divided by effort, and economic productivity as the value (exchange-value) of a unit of product divided by the production cost per unit (in wages paid). See Card (2006).

that reason, studies of FOSS development have used LOC as a measure of development effort and activity (Mockus et al. 2002; Koch 2004, 2008; Spinellis 2006). Another proxy of development effort used in studies of FOSS projects is the volume of changes made to the codebase, that is, the number of code contributions to the project (den Besten et al. 2006; Dinh-Trong & Bieman 2005; Koch 2004; Michlmayr et al. 2007; Mockus et al. 2002; Spinellis 2006). Besides LOC and code contributions, an alternative method of measuring size, effort and productivity is through function points (Albrecht & Gaffney 1983; Banker & Slaughter 1997; Blackburn et al. 2006; Kemerer 1993; Koch 2008; Perry 1986). For example, Blackburn et al. (2006) use the number of function points per man-month as a measure of productivity. Size in KB has been proposed as yet another measure (Ghosh & David 2003). Unfortunately, no metric is flawless. There has been extensive criticism of LOC as a measure of productivity on account of its tendency to emphasise larger rather than efficient or high-quality products. Simply put, that one software program is made up of more LOC than another might be an indication of more verbose code rather than of more functionality or a higher level of sophistication (Jones 1978; McAllister 2011). The same criticism applies to KB while the function points method has been criticised for being complicated to estimate and dependent on the analyst's subjective judgement of the importance of various complexity factors (U.S.A. Air Force Dept. 2000). A practical, though admittedly rough, solution to this problem is to use more than just one measure of size so as to be able to identify potential inconsistencies or contradictions in the results those metrics yield (Card 2006; see also Kitchenham & Mendes 2004).

In the present study, we use three alternative measures of production output: the number of a) LOC, b) KB, and c) commits (i.e. code contributions). Because we are interested in the returns to scale exhibited by the production process, that is, in the effect on productivity of adding more developers to a FOSS project (and because in that setting the time spent by contributors cannot be estimated directly), we use the number of active committers as an indicator of input. Thus, this metric captures average labour productivity in the project, which we calculate as follows:

$$Average\ productivity = LOC / committers \quad (2)$$

$$Average\ Productivity = KB / committers \quad (3)$$

$$Average\ Productivity = Commits / committers \quad (4)$$

Summing up, our method could be criticised for not including function points in

its repertoire of metrics. But we do not consider that to be a grave deficiency. For as Boehm (1987) concluded in his evaluation of various different metrics, LOC may not be a perfect measure of development effort and productivity, yet none of the other measures is fundamentally more informative.

The next section explains the derivation of the statistical analysis framework from the research model synthesising our hypotheses.

STATISTICAL ANALYSIS

The quantitative analysis is performed by means of regression analyses. Fig. 3.2 depicts the research model, which is summed up in the following hypotheses:

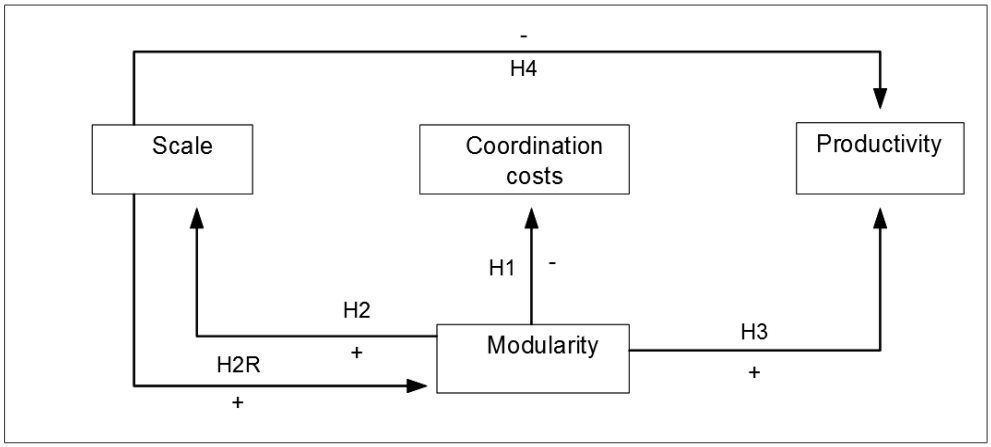


Fig. 3.2: Research model

#	Hypothesis
H1	Product modularity reduces coordination costs in FreeBSD
H2	Product modularity increases the potential number of contributors to FreeBSD
H2R	An increase of contributors to FreeBSD results in an increase of modularity

H3	Product modularity has a positive effect on labour productivity in FreeBSD
H4	An increase of contributors to FreeBSD has a negative effect on labour productivity

To test the hypotheses, we use panel data (also known as longitudinal or cross-sectional time-series data) collected from the FreeBSD project's software repositories. The reason we turned to longitudinal data is because we are interested in examining the effect of modularity on coordination costs, group size and productivity *over time.* This evolutionary approach is designed to capture the effect of scale on the dynamic of the development process, which can only be probed through the perspective of time, that is, over the course of development. For that reason, we 'partitioned' FreeBSD's development process (reconstructed through activity logs collected from FreeBSD's software repository) into fourteen consecutive years of development activity from 1994 until 2008. Thus, our analysis is based on yearly measurements: we estimate the degree of modularity per year (i.e. modularity in year 1, year 2,..,year 14), coordination costs per year (i.e. coordination costs in year 1, year 2,..,year 14), number of committers per year (i.e. committers in year 1, year 2,..,year 14) and productivity per year (productivity in year 1, year 2,..,year 14).

Sample selection

Panel data is a dataset in which the behaviour of some entities is observed across time. In our case, these entities are individual FreeBSD modules. Our dataset does not include all FreeBSD modules but only thirty of them which we selected through *stratified random sampling*. This means that we first categorised FreeBSD's 387 modules (at the time of selection)[40] into three non-overlapping groups, called strata, based on their scale (*small-scale, medium-scale, large-scale*) as reflected in the number of developers contributing to them (N=387, H=3). Following this step, we selected ten modules from each category in random, that is, thirty modules in total. Table 3.1 lists the modules included in the sample.

40 The dataset used for our analysis ends December 2007, at which time FreeBSD included 387 modules.

Small-scale	Medium-scale	Large-scale
cardbus	aac	cd9660
digi	agp	coda
joy	devfs	firewire
netatalk	hpfs	netinet6
netipsec	msdosfs	nfsclient
nfs4client	net80211	nfsserver
pccard	netncp	procsfs
random	ntfs	usb
rpc	nwfs	vm
xe	pseudofs	-[41]

Table 3.1: Sample of thirty FreeBSD modules

Because not all modules were added to the codebase at the same time, our dataset is *unbalanced*, meaning that some of the 29 modules comprising our sample do not have data for some years. The dataset is also *disproportionate*, meaning that the sample size of each stratum is disproportionate to the population size of the stratum. The majority of modules contained in the FreeBSD codebase are developed by small groups of no more than ten committers, but as modules produced by larger groups have probably more variability, we decided to allocate more than a proportionate share of the sample to the 'medium-scale' and 'large-scale' strata. The use of such disproportionate stratification is typical of cases like ours in which one wishes to give more precision to the estimates made for those strata with a small population (Piazza 2010).

Random-effects GLS regression

The statistical techniques most commonly used to analyse panel data are fixed-effects and random-effects regression. Regression is an approach to modelling the

41 One module was excluded due to insufficient observations. That is why the final sample includes twenty-nine rather than thirty modules.

relationship between a dependent (or scalar) variable and one or more independent (or explanatory) variables. To test our hypotheses, we use random-effects regressions. Unlike the fixed-effects model which exploits within-group variation, the random-effects model accounts for variation both within and between groups (Torres-Rayna 2008). That is, we opted for random-effects because we believe that differences between modules have a significant influence on our results. To make sure that the random-effects model is the right one, we ran a Hausman test for every regression we performed, which is commonly used to decide between fixed and random effects. This confirmed that our choice of random-effects is appropriate. To illustrate this procedure, consider the below Hausman test, which we ran for the regression of committers (indicator of group size) on integrality index (indicator of modularity):

```
                    ---- Coefficients ----
                |      (b)          (B)            (b-B)     sqrt(diag(V_b-V_B))
                |     fixed        random       Difference          S.E.
----------------+----------------------------------------------------------------
integrality_index |   -.0747161    -.1508722        .0761562        .0567604
------------------------------------------------------------------------------
                          b = consistent under Ho and Ha; obtained from xtreg
           B = inconsistent under Ha, efficient under Ho; obtained from xtreg

    Test:  Ho:  difference in coefficients not systematic

                chi2(1) = (b-B)'[(V_b-V_B)^(-1)](b-B)
                        =        1.80
              Prob>chi2 =      0.1797
```

Table 3.2: Hausman test for regression of committers on integrality index

In the test the null hypothesis is that the preferred model is random-effects. This is calculated by estimating random-effects and fixed-effects and then comparing the estimates. We see that the Prob>chi2 value is 0.1797. If that were smaller than 0.05 (i.e. significant), then fixed-effects would be the preferred model. In our case, Prob>chi2 is greater than 0.05, thereby confirming our choice of random-effects. The Hausman tests we ran for the other regressions gave similar results, Prob>chi2 being consistently greater than 0.05.

For the estimation of regression coefficients, we use the method of *Generalized Least Squares* (GLS). GLS is an extension of Ordinary Least Squares (the most common estimation method for regressions) which is commonly used for random-effects regressions, as OLS cannot simulate random-effects and is therefore unsuitable for our analytical purposes (Fox & Weisberg 2011).[42]

42 In analysing such a system of relations, an alternative to random-effects regression would be to

Operationalisation

As our research model consists of five hypotheses, we developed five regression models to test them at the component-level to which we turn now.

H1: Product modularity reduces coordination costs in FOSS projects

Fig. 3.8 below illustrates the hypothesised relationship between the variables included in the random-effects GLS regression model.

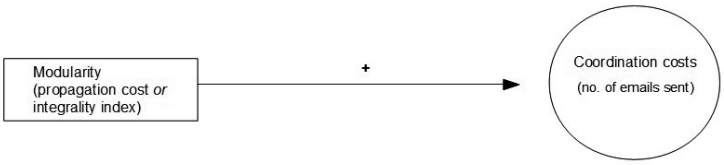


Fig. 3.8: Empirical model for H1

H2: Product modularity increases the potential number of contributors to FOSS projects

Fig. 3.9 illustrates the hypothesised relationship between the variables included in the random-effects GLS regression model.

Fig. 3.9: Empirical model for H2

H2R: An increase of contributors to a FOSS project results in an increase of modularity

Fig. 3.10 illustrates the hypothesised relationship between the variables included in the random-effects GLS regression model.

---

use Structural Equation Modeling (SEM). We chose not to do so, as relations between variables in the latter model would be harder to disentangle. But it is certainly a promising avenue for future research.

Fig. 3.10: Empirical model for H2R

H3: Product modularity has a positive effect on labour productivity in FOSS projects

Fig. 3.11 illustrates the hypothesised relationship between the variables included in the random-effects GLS regression model.

Fig. 3.11: Empirical model for H3

H4: Increasing group size has a negative effect on labour productivity

Fig. 3.12 illustrates the hypothesised relationship between the variables included in the random-effects GLS regression model.

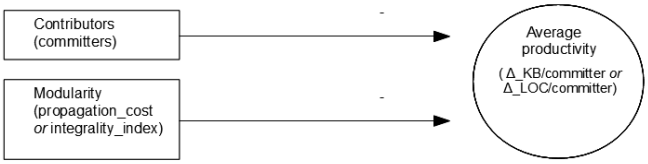


Fig. 3.12: Empirical model for H4

To test hypothesis *H4*, we ran a multiple regression, that is, a regression with more than one independent variable. The rationale is that to rigorously analyse the effect of group size on average labour productivity, we must control for the effect of modularity. Thus, an indicator of modularity was included in the regression as a control variable. A frequent problem in multiple regression is that of multicollinearity: the independent variables are near perfect linear combinations of

one another and so the estimates of the regression model cannot be precisely computed (Chen et al. 2003, chapter 2). To make sure that this is not the case with our regression model, we ran a Variance Inflation Factor (VIF) test, which confirmed there is no problem including both independent variables in the regression.

To illustrate this procedure, consider the below VIF test, which we ran for the regression of LOC added per committer (indicator of average productivity) on committers (indicator of group size) and integrality index (indicator of modularity). As can be seen in Table 3.3, the VIF value for committers and integrality index is 1.08. Heuristically, a variable whose VIF value is greater than 10 is problematic (Chen et al. 2003, chapter 2). Since the VIF value for both committers and integrality index is smaller than 10, there is no problem including both predictors in the regression model.

Variable	VIF	1/VIF
committers	1.08	0.925636
integrality_index	1.08	0.925636
Mean VIF	1.08	

Table 3.3: VIF test for regression of LOC added per commiter on committers, integrality index

It is important to mention that to explore the time-structure of causal processes, we also tested the empirical models with lagged (independent) variables in the regression tests.

The next section of this chapter summarises our data sources, the statistical instruments used for testing the hypotheses and the variables included in the regression analyses.

Summary of data sources, statistical tests and variables

Table 3.4 lists our data sources. Table 3.5 summarises the statistical tests used for evaluating the hypotheses.

Primary data	Secondary data
• Activity logs collected from FreeBSD code repository • Archived project	• Information released by FreeBSD (e.g. on FreeBSD website) or published by

communications (FreeBSD mailing lists) • Survey of FreeBSD core developers	FreeBSD developers • Past surveys of FreeBSD developers and organisation studies of FreeBSD

Table 3.4: Data sources

Hypothesis	Independent variables (i)	Dependent variables	Statistical instrument	N
H1	Propagation_cost, integrality_index	vol_of_emails	Descriptive statistics	N=Raw dataset
H2	propagation_cost, integrality_index	committers	Descriptive statistics, regression analysis	N=242
H2R	committers	propagation_cost, integrality_index	Descriptive statistics, regression analysis	N (*small-scale/ large-scale*) = 148/123
H3	propagation_cost, integrality_index	Commits/committer, Δ_LOC/committer, Δ_KB/committer	Descriptive statistics, regression analysis	N (*small-scale/ large-scale*) = 121/121
H4	committers	Commits/committer, Δ_LOC/committer, Δ_KB/committer	Descriptive statistics, regression analysis	N=277

Table 3.5: Summary of statistical tests

Table 3.6 lists all the variables, both independent and dependent, that will be included in the regression analyses. Table 3.7 shows the summary statistics.

Variable (i)	H1	H2	H2R	H3	H4
Modularity (propagation cost)	IV	IV	DV	IV	IV
Modularity (integrality index)	IV	IV	DV	IV	IV
Coordination costs (emails sent)	DV	-	-	-	-
Group size (committers)	-	DV	IV	-	IV
Average productivity (commits per committer)	-	-	-	DV	DV
Average productivity (LOC per committer)	-	-	-	DV	DV
Average productivity (KB per committer)	-	-	-	DV	DV

Table 3.6: Variables (i) used in regression analyses
(*Notes on the table.* IV: independent variable; DV: dependent variable)

Variable	Obs	Mean	Std. Dev.
propagation_cost	280	.3821786	.148343
integrality_index	271	4.175121	4.765997
ext_dependencies	280	76.80714	41.76078
ext_dependencies_per_module_j	280	45.03814	7.225033
propagation_cost_j	280	.1431786	.0453292
committers	280	9.2	7.167429
D_KB_per_committer	277	10.55867	56.7478
D_LOC_per_committer	277	259.8819	1106.289
commits_per_committer	280	8.55925	8.853674

Table 3.7 Summary statistics for variables used in regression analyses

Before we proceed to the data analysis, the next chapter describes the empirical setting of the research: the FreeBSD Project.

CHAPTER 4: EMPIRICAL SETTING

HISTORICAL BACKGROUND

FreeBSD is a free/open source[43] operating system descended from the Berkeley Software Distribution (BSD), the version of Unix developed at the University of California at Berkeley.

Unix was born in 1969. When AT&T's Bell Telephone Labs (BTL) pulled out from Multics – a joint project of BTL, General Electrics and MIT aimed at developing an operating system capable of supporting simultaneously multiple users – some BTL programmers took it upon themselves to develop it without the support or even endorsement of their employer. So, the development of Unix began in a informal and anti-bureaucratic fashion. Bypassing BTL's corporate hierarchy, the programmers who spearheaded the making of Unix coordinated their work through their 'mutual adjustment' without any supervision or involvement on the part of their formal superordinates. But not being able to tap into the resources – whether administrative, technical or financial – controlled by BTL's corporate bureaucracy, they resorted to enlisting the participation of the hacker community, 'opening up' the Unix development process to anyone willing to contribute. As a result of their willingness to share their work with other researchers, a network of users interested in enhancing Unix rapidly began to take shape in computer research institutes and universities around the globe. Of all development hubs outside AT&T, the most influential was the University of California at Berkeley, acting as a clearing-house for Unix research (Raymond 2003; Ritchie 1984; Salus 1994).

In 1973 Dennis Ritchie and Ken Thompson, the two BTL programmers chiefly responsible for the early development of Unix, presented a conference paper about Unix at Purdue University. Bob Fabry of the University of California, who was in attendance, took an immediate interest in it and brought a copy of Unix with him back to Berkeley (McKusick 1999, p. 31). The arrival of Thompson to Berkeley in 1975 (who had himself graduated from Berkeley in 1966) as a visiting professor reinforced the popularity of Unix at Berkeley and acted as a catalyst for the formation of a research group of graduate students and staff researchers who were

43 FreeBSD is distributed under the terms of the FreeBSD license. See **Appendix I: The FreeBSD License**.

to spearhead a software development effort culminating in what became known as the Berkeley Software Distribution or BSD for short (Leonard 2000). In his capacity as computer science professor, Fabry was the one responsible for making sure that the group was equipped with the requisite resources by 'manoeuvring through the formidable bureaucracy of the University of California and AT&T' (Leonard 2000), while a small group, led by Bill Joy, who arrived to Berkeley in 1975 to attend graduate school, concentrated upon the task of developing the software. The first release of BSD occurred in 1977 with Joy as 'distribution secretary'. The next year (1978) Joy put together the 'Second Berkeley Software Distribution', shortened to 2BSD, followed by 3BSD in 1979. The same year Fabry secured a contract to develop an enhanced version of 3BSD for the Defence Advanced Research Projects Agency's (DARPA) fledging computer network (which evolved into what we now call the Internet). Under the auspices of this contract, he set up the Computer Science Research Group (CSRG), to which he appointed Joy as project leader. The improved version was released in October 1980 as 4BSD, followed by 4.1BSD in 1981. When Joy departed in 1982, Sam Leffler – Joy's second-in-command – shouldered the responsibility of completing the release of 4.2BSD. Following its completion in August 1983, Leffler was replaced by Mike Karels, who was joined by Kirk McKusick a year later in December 1984. For the next seven years since Karels and McKusick picked up the reins, five more major BSD releases were made: 4.3BSD was released in 1986, 4.3BSD-Tahoe in 1988, Net1 in 1989, 4.3BSD-Reno in 1990 and Net2 in 1991. The popularity of BSD rose higher with each one of them, in tandem with the rise in the number of its users and co-developers around the world.[44] Two of them, named Lynne and Bill Jolitz, took the initiative to adapt BSD (using the latest release 4.3BSD Net2) to the Intel x86-based PC architecture.[45] Thus, in 1992 they released on the Internet a fully functioning system for the 386PC to which they gave the name 386BSD. The feedback was truly overwhelming: the Jolitzes were inundated with a plethora of bug fixes and

44 For an elaborate chronicle of the development of BSD by one of its leading figures, see McKusick (1999) or McKusick et al. (1996, chapter 1).

45 According to FreeBSD developer Rich Murphy, 'Berkeley contracted Bill Jolitz to port BSD to the x86 platform, and he negotiated terms of his contract that required his source code to be released publicly' (Asterisk News 2004). Chalmers (2000) includes a rather revealing passage on the motivation of the Jolitzes to port BSD to the x86 architecture: 'In 1989 or 1990, Lynne remembers, the Berkeley distribution had gone from being available on the most relevant machines to being limited to what the Jolitzes saw as the most irrelevant. "There was an HP [Hewlett-Packard] port in progress and nothing else," she says. "Since we were looking for recreation, we offered to do one for the 386." "Completely as a lark," Bill adds'.

enhancements to 386BSD to the point that they could not keep up with it. Confronted with their lack of responsiveness, a group of users began collecting bug fixes and enhancements, distributing them as the 'unofficial 386BSD patchkit'. Initially the 'patchkit' was supposed to be a temporary solution to Jolitz's problematic handling of patches: its coordinators believed that its contents would eventually be merged into the next 386BSD release. But when in 1993 Jolitz withdrew his support from the project, the last three coordinators of the patchkit – Rod Grimes, Jordan Hubbard and Nate Williams – decided to form the FreeBSD group to coordinate its further development. Almost simultaneously, frustration with the pace of work in 386BSD led to the formation of one more splinter group, NetBSD, which began a parallel development effort, focusing on the adaptation of 386BSD to non-x86 architectures (Chalmers 2000; FreeBSD 2011b, chapter 1; Howard 2001; McKusick 1999).

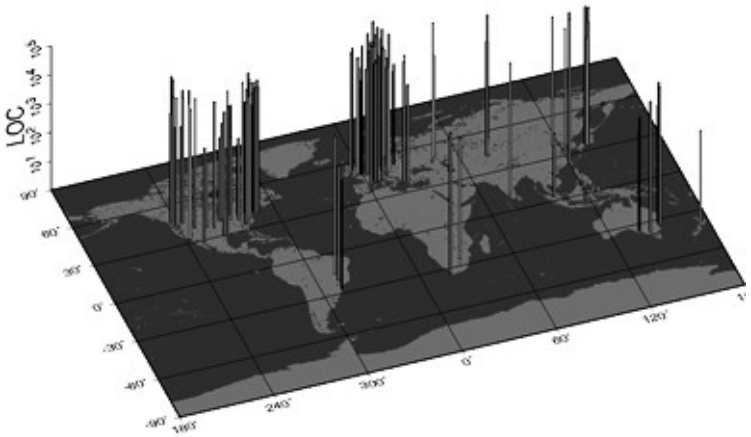


Fig. 4.1: International development (Source: Spinellis 2006)

The first version of FreeBSD was released in December 1993. Shortly thereafter the project was unwillingly enmeshed in the legal conflict between the University of California and Unix System Laboratories (USL), an AT&T subsidiary which was later acquired by Novell, when in 1992 USL sued the University of California for distributing what it claimed to be proprietary AT&T code. When it turned out that USL's distribution contained code written by BSD, the University of California responded likewise. Faced with this problem, FreeBSD hackers rewrote major parts of the software in order to get rid of the contentious code, using 4.4BSD-Lite r2 –

the latest release made by the CSRG[46] – as the basis of that reworking (FreeBSD 2011b, chapter 1; McKusick 1999). The result was released as version *2* in November 1994. Since, FreeBSD has been established as the most popular BSD-descendant with a proven track record in mission-critical deployments.[47] The latest release of the project, version *9.0*, was made in January 2012 (FreeBSD 2011h). Nowadays, the project thrives on the contributions of a community of software developers spread the world over. Though development effort is heavily concentrated in North America and Europe (see Fig. 4.1 above), FreeBSD development takes place in 34 countries on six continents (Spinellis 2006; Watson 2006).

ORGANISATIONAL STRUCTURE

The organisational structure of FreeBSD is to large extent inherited from BSD, often credited for codifying a template for what is now known as the open source development model.

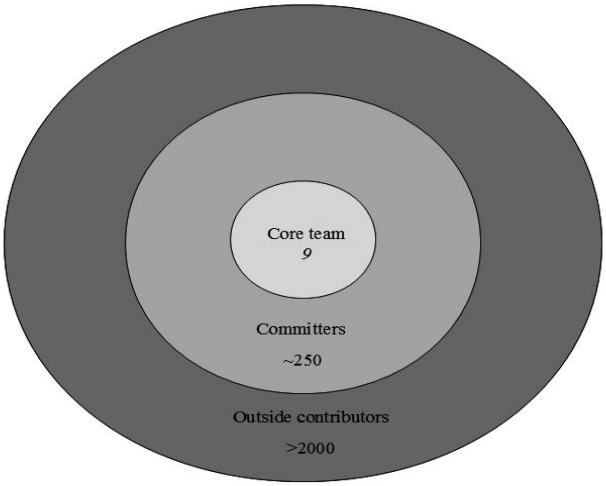


Fig. 4.2: FreeBSD organisational structure

As Kirk McKusick, one of the BSD hackers and long-standing FreeBSD developer, says:

46 Following the release of 4.4BSD-Lite Release 2 in June 1995, the CSRG was disbanded.
47 See <http://www.bsdstats.org> and BSD Certification Group (2005).

> the contribution that we made, ultimately was in developing a model for doing open-source software...We figured out how you could take a small group of people and coordinate a software project where you have several hundred people working on it (quoted in Leonard 2000).

This structure has a *core team* at its centre: a small group of programmers who control access to the codebase, vested with authority to grant or revoke the right to integrate changes to the project's code repository. Spreading out from them are the *committers*, who have the right to check in changes, framed by the wider community of *outside contributors*.

Core team

The *core team* is responsible for assigning commit[48] privileges to developers ('awarding commit bits' in FreeBSD terminology) as well as suspending them, for resolving conflicts between them and appointing sub-committees for specific tasks (e.g. release engineering, security officer, webmaster). In this sense, the core team serves as the project's 'Board of Directors'. However, the role of the core team is not supposed to be merely administrative; its members are engaged extensively in software development, contributing code to the project.

In 1993 the FreeBSD core team numbered 13 members: the tree founders of the project – Jordan Hubbard, Nate Williams and Rod Grimes – plus the most active then-committers. Now, it consists of nine members elected to a two-year term by and amongst active committers. Active are considered committers who have made at least one commit in the last twelve months, all of whom are eligible to vote and run as candidates.

In the beginning, following the tradition established by BSD, 'those who hacked most became part of the “core group” or “core team”' (Lehey 2002). However, as FreeBSD committer Greg Lehey (2002) explains, 'by 2000, the core team was no longer the most active group of committers'. In parallel, concerns of a perceived illegitimacy in the exercise of authority by the core team, which to some extent had always been present, assumed crisis proportions. In an attempt to weather the storm, Hubbard proposed a number of alternatives about the future of the core

48 'When a change is integrated, it is called a commit' (Saers 2005).

team – ranging from disbanding the core team completely to keeping it intact – and called on committers to decide by vote. It was thus decided to adopt an elected model, based on the following bylaws drafted to regulate core team elections (FreeBSD 2000; Lehey 2002):

• The core consists of nine elected active committers and election is held every two years
• Core members and committers may be ejected by a 2/3 vote of core
• If the size of core falls below 7, an early election is held.
• A petition of 1/3 of active committers can trigger an early election.
• These rules can be changed by a 2/3 majority of committers if at least 50% of active committers cast their vote.

Table 4.1: Core bylaws (Source: FreeBSD 2000)

Approved by a vote of active committers (passed by 117 yes votes to 5 no votes [Lehey 2002]) on 28 August 2000, these bylaws established criteria of eligibility (all active committers), the size of core team (nine committers), the periodicity of elections (fixed at every two years) and the conditions under which: (a) early elections are held (on the petition of 1/3 of active committers or if size of core falls below 7), (b) a core team member or committer can be expelled from the project (by a 2/3 vote of core) and (c) these bylaws can be modified (FreeBSD 2002). The first core team formed in that way through elections consisted of five former core members (Satoshi Asami, David Greenman, Jordan Hubbard, Doug Rabson, Peter Wemm) plus four new ones (Greg Lehey, Warner Losh, Mike Smith, Robert Watson).

The first serious test of the ability of the reformed core team to manage conflicts between committers occurred in February 2002 when a committer made significant changes to the SMP[49] module despite the fact that other committers had pointed out, when he announced his intention to do so, that his changes conflicted with those that John Baldwin – the most active then-SMP developer – was testing and that he should refrain from committing his changes before consulting with Baldwin. The core team stepped in, threatening to suspend his commit privileges if he did not back out his changes. He removed the changes and asked the core team

49 The goal of the SMP project was to introduce parallelism into the kernel so that FreeBSD could be run on multiprocessor computer hardware architectures.

to resolve the issue. In the end, after a month of discussion, the core team took the side of Baldwin, delegating authority to him to approve or reject changes to the SMP code. This experience led the core team to formulate disciplinary rules for the suspension of commit rights.[50] These rules are as follows:

1. Committing during code freezes results in a suspension of commit bits for two days.
2. Committing to the security branch without approval results in a suspension of commit privileges for 2 days.
3. Commit wars will result in both parties having their commit bits suspended for 5 days.
4. Impolite or inappropriate behaviour results in suspension of commit bits for 5 days.
5. Any single member of core or appropriate other teams can implement the suspension without the need for a formal vote.
6. Core reserves the right to impose harsher penalties for repeat offenders, including longer suspension terms and the permanent removal of commit privileges. These penalties are subject to a 2/3 majority vote in core.
7. In each case, the suspension will be published on the developers mailing list.

Table 4.2: Rules for the suspension of commit rights (Source: FreeBSD 2011d; Lehey 2002)

However, in order for the decisions of the core team to be received as legitimate, they must be (perceived as) consistent with the consensus of the opinions of the committers. Characteristically, in June 2002 the core team received another complaint about the same committer. Once again he had committed changes to an area of the codebase without the approval of the committer who was responsible for it. The core team decided to suspend his commit privileges for five days in accordance with the aforementioned disciplinary rules. But 'public reaction was unfavourable': the decision was censured for being politically-motivated, as core elections were underway and the suspended committer was a candidate. Under these circumstances, the core team was forced to reprieve the suspension after two

50 For a first-hand account of the implementation of SMP in FreeBSD, see Lehey (2003). In connection with the specific conflict related in the text, see Lehey (2002). For a treatment from the perspective of organisation studies, see Holck & Jørgensen (2003/2004, p. 46) and Jørgensen (2001, p. 5; 2005, p. 234).

days (Lehey 2002). However, not all conflicts are so hard to resolve. According to core team member Robert Watson, the vast majority of disputes between committers are resolved informally without requiring the mediation of the core team because 'the community is self-selecting, and primary criteria in evaluating candidates to join the developer team are not just technical skills...but also the candidate's ability to work successful as part of a larger development team' (Watson 2006).

In 2002 elections were held again as the core team was left with six members following the resignations of Satoshi Asami, Jordan Hubbard and Mike Smith. The new core team had five new members (John Baldwin, Jun Kuriyama, Mark Murray, Wes Peters, Murray Stokely) and four from the previous one formed in 2000 (Greg Lehey, Warner Losh, Robert Watson, Peter Wemm). Of its nine members, only one – Peter Wemm – was part of the original core team. Elections have been held four more times since. The last one in 2010 resulted in the following core team: John Baldwin, Konstantin Belousov, Warner Losh, Pav Lucistnik, Colin Percival, Wilko Bulte, Brooks Davis, Hiroki Sato and Robert Watson.

Committers

Committers are the FreeBSD developers who have the right to commit changes directly to the project's code repository. They are also responsible for integrating code that contributors without commit privileges send them. Outside contributors advance to the ranks of committers when their nomination by an existing committer is approved by the core team, which alone has authority to grant commit privileges.[51] This procedure, as committer Michael Lucas explains, is 'fairly straightforward':

> if you submit enough useful and correct PRs [problem reports], eventually some committer will get sick of taking care of your work and will ask you if you want to be able to commit them yourself. This process serves multiple purposes; after all, the FreeBSD community is made up of people who do the work. For committers, the work consists of creating useful and correct patches. If you don't consistently and regularly create good patches,

51 This applies to *src* committers. Ports and documentation committers are approved by the Port Management Team and the Documentation Engineering Team respectively.

> there's no point in giving you commit access, now is there?...By the time you've submitted several dozen PRs, you'll either work well with the FreeBSD team or everyone will understand that you and the team just can't get along. Direct-commit access is either an obvious next step, or an obviously bad move (Lucas 2002).

New committers are assigned a *mentor*, typically the same person who recommended them for commit privileges. Mentors are responsible for everything their protégés do in the project, including answering their questions, reviewing their changes and familiarising them with FreeBSD's 'rules and conventions'. The period of mentorship, which could last for several months, ends when the mentor 'releases' formally the new committer, feeling that he has proven he can work harmoniously with others in the project (FreeBSD 2011a; Lucas 2002).

Committers focus on either of the three main areas of development at FreeBSD: *src* (kernel and userland), *ports* or *documentation*. Indicatively, a breakdown of the 275 committers who made commits in 2002 (from 31 December 2001 to 31 December 2002) reveals the following division of labour: 201 src committers, 144 ports committers and 41 documentation committers (Saers 2005; see also Watson 2006).[52]Their age varies between 17 and 58 years, with a mean age of 32 and median age of 30; the standard deviation is 7.2 years (Watson 2006).

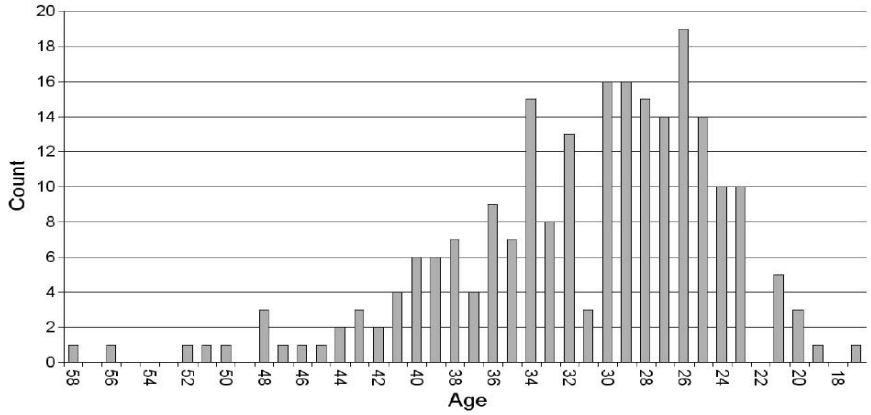


Fig. 4.3: Age distribution of committers (Source: Watson 2006)

52 The subsequent analyses in chapters 6, 7 and 8 focus on *src* committers alone. This analytical choice was made on the grounds that the other two areas of work on FreeBSD (*ports* and *documentation*) pertain less to new code development and more to peripheral, though necessary, activities.

Although FreeBSD is a volunteer organisation and committers receive no remuneration from the FreeBSD project for their contributions, many of them are seasoned professionals working in the IT industry. Thus, it is not surprising that, for some of them, working on FreeBSD is part of their professional work. In a survey of 72 FreeBSD committers (constituting 35 percent of all committers) conducted in 2000, 21 percent...said that work on their latest contribution had been fully paid for, and another 22 percent partially paid for' (Jørgensen 2005, p. 233). Warner Losh, sitting member of the core team, is one of them. In his opinion, getting paid to work on FreeBSD is not uncommon. As he says: 'my current employer, for example, allows me a certain amount of time each month to work on FreeBSD bugs that impact our ability to deploy a system. These get fed back into the base FreeBSD from time to time. Many other people are in a similar situation' (Losh interviewed in Loli-Queru 2003). For other FreeBSD committers, however, the importance of economic incentives should not be over-emphasised, for, as former core team member Greg Lehey says, 'a lot of people are motivated more than by money to work on FreeBSD. It is their hobby or passion. They find an itch to scratch using FreeBSD and FreeBSD benefits' (Lehey interviewed in Loli-Queru 2003).

Outside contributors

Outside contributors constitute the third layer of the FreeBSD organisational structure. They are those who contribute to the project but do not have commit privileges. Indicatively, in 2001 there were 1181 contributors without commit rights on the periphery of the project (FreeBSD 2001a), 1399 in 2003 (FreeBSD 2003), 2018 in 2006 (FreeBSD 2006) and 2162 in 2010 (FreeBSD 2010d).

Ad hoc teams

In addition to the core team, FreeBSD is supported administratively by an extensive array of ad hoc teams. The Documentation Engineering Team (4 members) and the FreeBSD Port Management Team (9 members) complement the core team in the context of general project management (FreeBSD 2011e). Beside the *Primary Release Engineering Team* (9 members), which is responsible for managing releases, there are seven more release engineering teams corresponding to different

architectures.[53] Four more teams centre on donations (9 members), marketing (12 members), security (11 members) and vendor relations (7 members) respectively. And last, thirteen teams deal with matters of internal administration (e.g. administering and maintaining project websites, FTP servers, CVS, GNATS) (FreeBSD 2011e). All teams are manned by committers assigned by the core team (on a voluntary basis of course), to which they are accountable. The relations between these teams are summed up in the organisational chart in Fig. 4.4:

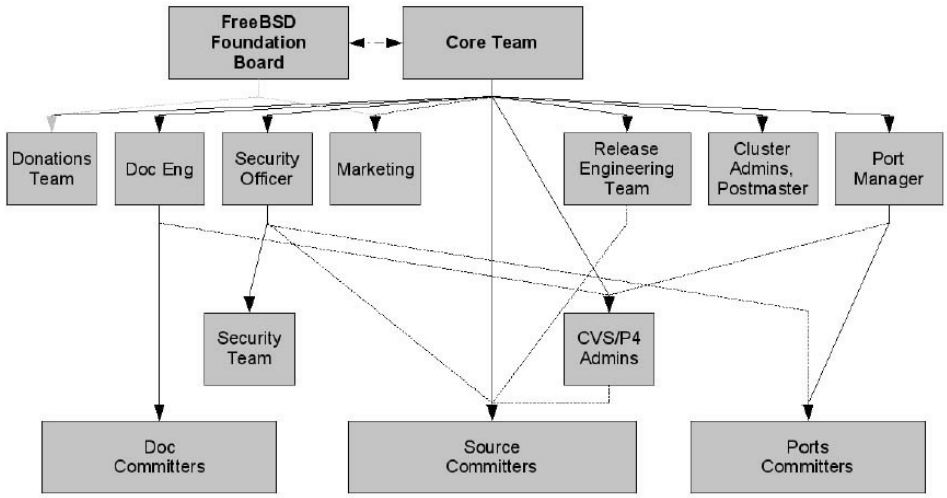


Fig. 4.4: FreeBSD organisational chart (Source: Watson 2006)

Though existing as a separate legal entity,[54] among the aforementioned teams could also be considered the *FreeBSD Foundation,* which was founded in 2000 to support the development and popularisation of FreeBSD (FreeBSD Foundation 2011). More specifically, whereas the core team is managing the development process, the Foundation, whose eight directors are drawn from the FreeBSD committers base, is responsible for the financial (e.g. fund raising) and legal aspects

53 As of June 2011, these include the Alpha Release Engineering Team (2 members), the AMD64 Release Engineering Team (1 member), the IA-64 Release Engineering Team (1 member), the i386 Release Engineering Team (2 members), the pc98 Release Engineering Team (1 member), the PowerPC Release Engineering Team (2 members), and the sparc64 Release Engineering Team (7 members). See FreeBSD (2010b).

54 Due to uncertainty over the Foundation's long-term organisational viability, it was decided to set it up as a separate legal entity (based in Boulder, Colorado, USA) so that the project would not depend on the viability of the Foundation (Watson 2006).

of the project.

Considering, however, the volunteer character of participation in the project, the arrows in the organisational chart do not signify top-down authority relations as conventionally understood in the context of hierarchical organisations. As Lehey (2002) says, 'the FreeBSD project is a volunteer organization, so the core team does not have a mandate to tell anybody to do anything'. Rather, as 'the organization is volunteer-driven' and the core team is elected by and amongst committers, 'delegation of responsibility occurs up as much as down' (Watson 2006).

Hats

Committers appointed by the core team to be responsible for some area of the code are called 'hats': they are expected to guide development in that area of the codebase and review submitted code (Losh 2006; Saers 2005, chapter 5). Hats may also pertain to tasks of internal administration such as Perforce Repository Administrators, CVS src Repository Managers, Bugmeisters or GNATS Administrators; and hats purportedly bearing a rather heavy work-load tend to be assigned to teams of committers rather than a single person. Examples of such hats are all the aforementioned *ad hoc teams* that support the core team in matters of project management. Some of these hats have been formalised over time: for example, the hat of FreeBSD Security Officer, which is currently appointed to a team of eleven committers, has been subject to the FreeBSD Security Officer Charter since 2002, which specifies its duties and responsibilities.[55] Most of the hats, however, have no charter attached to their functioning.

Maintainers

The most common hat to which committers are appointed is that of *maintainer.* Maintainers are committers vested with authority by the core team to review code submissions in a certain area of the codebase. A maintainer is thus expected to be responsible for that area of the codebase in which, as demonstrated through his participation in the project, he is an expert. Consequently, should some committer wish to make a change to an area of the code that is being maintained by someone else, it is advisable to send that change to him as he would have done before

55 FreeBSD Security Officer Charter. Accessible online at <http://www.freebsd.org/security/charter.html>

becoming a committer (FreeBSD 2011a; FreeBSD 1996). The maintainer's role, as the 'maintainers file' contained in the code repository explains, can be likened to that of a 'caretaker':

> In return for their active caretaking of the code it is polite to coordinate changes with them...this is not a 'big stick', it is an offer to help and a source of guidance. It does not override the communal nature of the tree. It is not a registry of 'turf' or private property (FreeBSD 2011i).

It becomes readily understood that the notion of responsibility in the case of FreeBSD maintainers should not be conflated with a mode of ownership (or stewardship) configured around the right to exclude others from modifying the codebase. The job of maintainers is to coordinate the process of integrating changes that impact the area of the code for which they are responsible, not to stall its further development.

TECHNICAL INFRASTRUCTURE

The development of FreeBSD would have been unthinkable had not been for the Internet. Since its inception, the project has been thriving on the contributions of a loosely coupled community of software developers spread the world over, connected only by the electronic strands of the Internet.

Communication channels

In consequence of the extremely limited scope for face-to-face communications, the vast majority of project activities occur on the Internet. Project members communicate primarily through mailing lists, which constitute the 'life-blood of the project' (Watson 2006). With the exception of a few mailing lists which are 'private' (such as freebsd-core which is intended for discussion of confidential matters by the core team), most mailing lists used in the project are 'public' so that anyone can browse their archives and read the messages exchanged via them.[56] The repertoire of communication tools used by FreeBSD developers is complemented

56 As of August 2011, there are 144 public mailing lists in use (<http://lists.freebsd.org/mailman/listinfo>)

with Internet Relay Chat (IRC) channels and – since 2008 – web forums.

Revision control

The FreeBSD project uses a parallel development process (which will be analysed in greater detail below), which means that development continues on two parallel tracks. The development of new functionality occurs in FreeBSD-Current, while a more stable branch is also maintained, known as FreeBSD-Stable. To coordinate work on the two branches, the project has been using the CVS revision control system to track and provide control over changes to them since its launch in 1993.[57] However, in order to more effectively accommodate massively parallel development, the project has been experimenting since 2003 with the use of multiple revision control systems, migrating increasingly more development activities centred on new features from the CVS environment to Perforce and Subversion (SVN) over time (FreeBSD 2011a, 2011b; Long 2010; Watson 2006; Wemm 2008).

Reporting & managing defects

FreeBSD uses the GNATS bug-tracking database to report problems and keep track of their resolution.

Testing

To test whether the evolving product is kept in a working state, FreeBSD uses three so-called Tinderbox servers that automatically build the most recent version of the software every few hours.[58] The results are posted on the web and on project mailing lists, notifying committers of 'tinderbox failures'.

57 As Hubbard (1998b) explains: 'CVS lets us keep the different threads of development separate while assisting us with the merge process when something from the experimental track has had sufficient testing to enter the mainstream product'.

58 The results of the daily build process are accessible online at <http://tinderbox.freebsd.org>. Indicatively, on 21 June 2011, tinderbox machines performed builds of the *-current* version and of six officially released versions of FreeBSD on nine different hardware platforms.

Distribution channels

FreeBSD software is distributed through various ways on the Internet: it can be downloaded through bitTorrent,[59] anonymous FTP,[60] anonymous CVS, anonymous SVN or CVSup. In addition, CDs and DVDs are available from several online retailers (e.g. FreeBSD Mall at <http://www.freebsdmall.com>)(FreeBSD 2011b).

DEVELOPMENT PROCESS

Like several other large FOSS projects, FreeBSD has a parallel development structure. There are two simultaneous development processes underway, crystallised in two different branches of the software. The *stable* branch represents the official released version, aimed at a stable and bug-free product. The *current* branch,[61] on the other hand, is experimental: it is where most cutting-edge developments and significant changes (e.g. new features) are first tried out. Fig. 4.5 illustrates the development model based on the process by which changes are integrated in the repository.

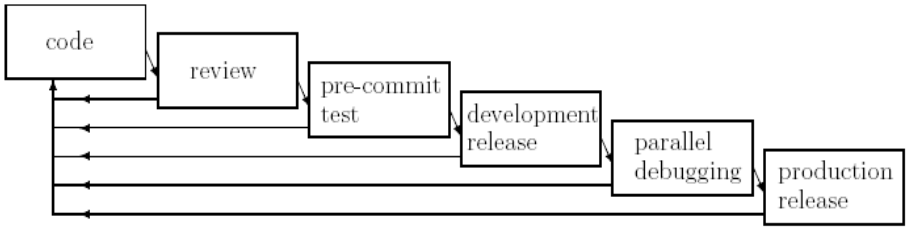


Fig. 4.5: Change integration process (Source: Jørgensen 2001)

Prior to committing their changes to the repository, committers are expected to ask for community review (FreeBSD 2011d). This practice usually generates a relatively modest amount of feedback,[62] based on which they either have to revisit

59 The torrent files can be downloaded from the FreeBSD BitTorrent tracker at <http://torrents.freebsd.org:8080/>

60 In addition to the central FTP server (ftp://ftp.FreeBSD.org/pub/FreeBSD/), the software is available from a worldwide set of mirror sites listed at <http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/mirrors-ftp.html>

61 FreeBSD-Current is also known as *HEAD* or *trunk*.

62 In a survey of 72 FreeBSD committers (constituting 35% of all committers) conducted in 2000, 86% said they received feedback from two or more reviewers (Jørgensen 2001).

their code or proceed to testing it on their own systems (by doing a trial build).[63] Next, they commit the changes to the *current* branch, from which a development release is built and made available for download every few hours. This release is tested and debugged concurrently by many more users and developers who download the software, resulting therefore in significant improvement. Once sufficiently tested and deemed mature enough, the code is merged by the committer in the *stable* branch,[64] from which a production release is made about every four months.[65]

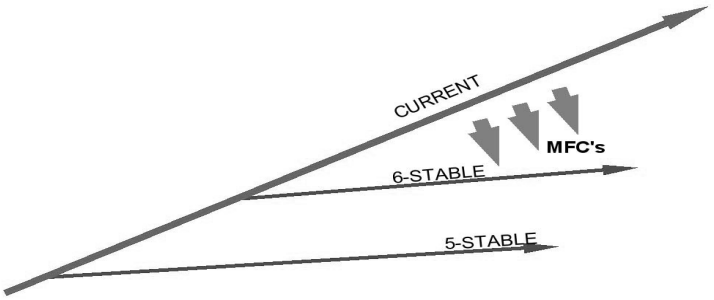


Fig. 4.6: Branching: stable releases are branched from *Current*; features trickle from *Current* to stable branches as they stabilise (Source: Watson 2006)

The process, despite its incremental character, is recursive: each stage of the process might require of the committer to return to his code for further changes, thereby re-initiating the process. Naturally, as developers work mostly individually,[66] the model is used in parallel by multiple developers (Holck & Jørgensen 2004; Saers 2005).

Thirty days before the anticipated release date, the repository enters a *code slush*. During this time, only corrective changes (i.e. bug-fixes) can be checked in and they have to be approved by the Release Engineering Team. After the first

63 Doing a build is an automated process by which (human-readable) source code is compiled to an executable program. If the compilation fails, then the build is said to be broken.

64 The process of merging code from the current branch to the stable branch is known as *Merged From Current* (MFC).

65 The project has been using a schedule with fixed timelines since the start of the 6-CURRENT development branch in 2004 (see Quarterly status reports, 2004).

66 In a survey of 72 FreeBSD committers (constituting 35% of the group of committers) conducted in 2000, '65% said that their last task had been worked on largely by themselves only, with teams consisting of 2 and 3 committers each representing 14%' (Jørgensen 2001).

fifteen days of the code slush, a *release candidate* is released and at the same time the repository enters a *code freeze*, after which point further changes to it become almost impossible. The release candidate is further tested until considered ready by the Release Engineering Team, which then releases it as the official production release (Jørgensen 2001; Stokely 2011; Watson 2006).

As can be seen in Fig. 4.7, which shows the time that elapsed between successive FreeBSD releases from the release of version 1 in 1993 until the release of version 5 in 2003, the FreeBSD development process results in a new release being made on average every 96.2 days (with a standard deviation of 62.9 days).[67]

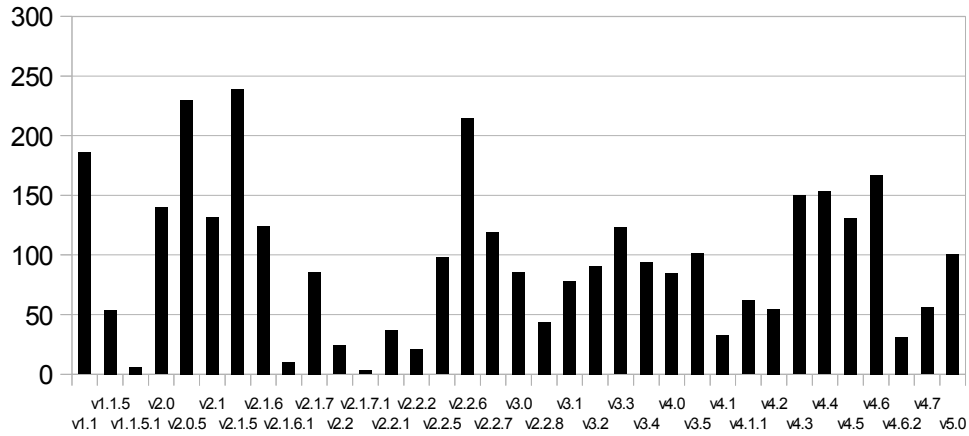


Fig. 4.7: Days between releases

SCALE

In the space of fourteen years from 1994 to 2008, the scale of the project has increased remarkably. As Fig. 4.8 below illustrates, the size of the *current* branch in KB has increased by about 2350% and by 2855% if measured by lines of code (LOC). The increase of its size and functionality is also reflected in the number of modules comprising it, which manifest an increase by 1370%. Similarly, the expansion of scale is mirrored in the enlargement of the (*src*) committers' base (see Fig. 4.9 below). Whereas in 1994 only 16 developers checked in code, their number rises over time to 198 in 2007, an increase by 1250%.

67 See **Appendix II: Release rate**.

Year	KB	LOC
1994	9916	112960
1995	32284	496613
1996	42504	657172
1997	48152	755999
1998	58068	921744
1999	80532	1283254
2000	106224	1652801
2001	124452	1942366
2002	132280	1979032
2003	156548	2331529
2004	168572	2481646
2005	182992	2602123
2006	199384	2828727
2007	218316	3034654
2008	243080	3339072

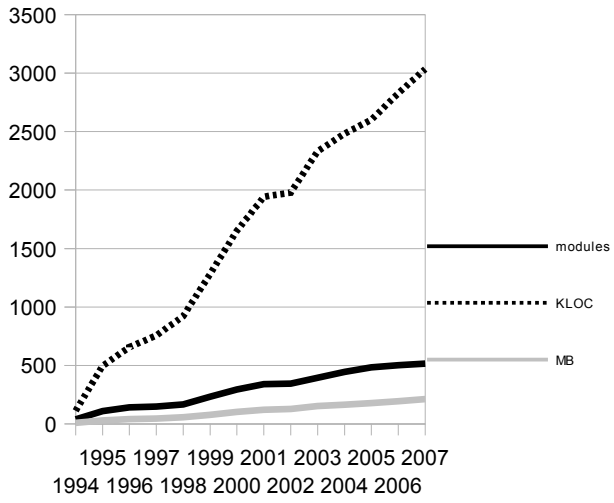


Fig. 4.8: Codebase evolution (Current branch, *src*, 1994-2007)

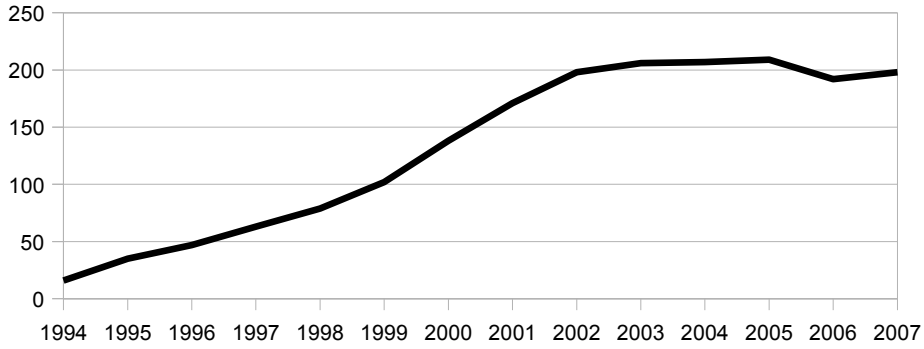


Fig. 4.9: (*src*) committers

More specifically, new committers are continuously added to the group, but only a small fraction of them ever opts out. Characteristically, as can be seen in Fig. 4.10 and Fig. 4.11 below, in the space of three years from January 2000 to January 2003, 142 developers were given commit-rights, while only 24 were removed from the group.[68]

68 See **Appendix III: Committers added and removed per month**.

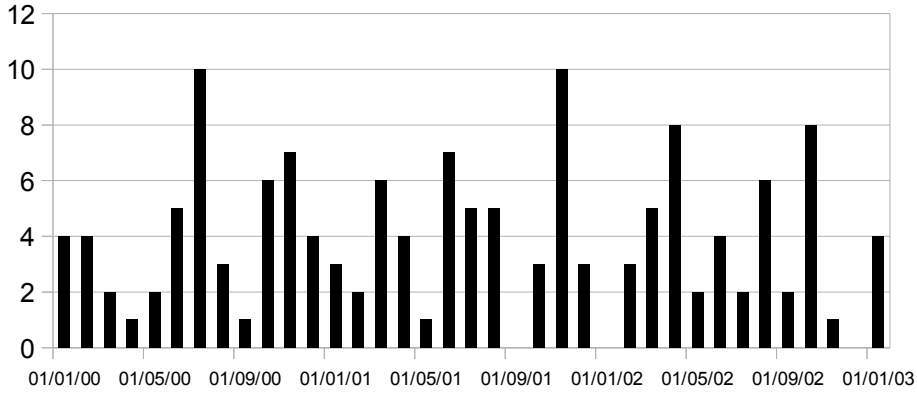


Fig. 4.10: New committers per month

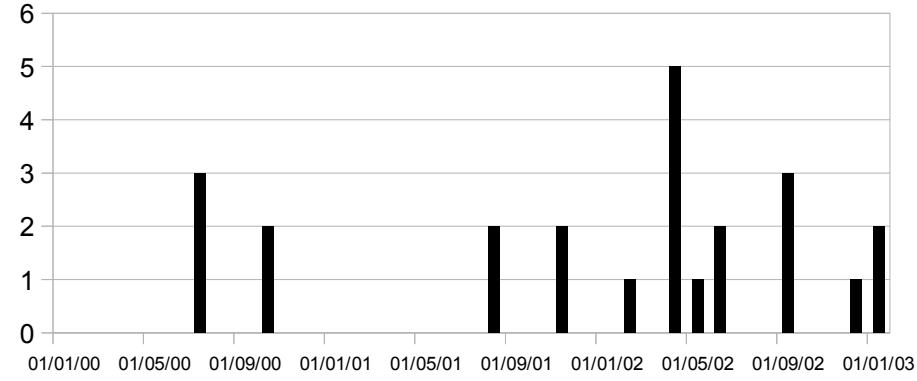


Fig. 4.11: Removed committers per month

To test the first hypothesis derived from the literature review in chapter 2, the next chapter examines the effect of modularity on coordination costs.

CHAPTER 5: MODULARITY AND COORDINATION COSTS IN FREEBSD

INTRODUCTION

An assumption that figures prominently in the literature of modularity is that modular product design mitigates the need for active coordination between distinct product components, thereby reducing the coordination costs involved in the product development process. Paradigmatic of this literature stream is Baldwin and Clark's (2006a) modularity theory, which holds that the end result of the modularisation process is to 'move decisions from a central point of control to the individual modules. The newly decentralized system can then evolve in new ways' (Baldwin & Clark 2006a, p. 183).

Consequently, 'the new organizational structure imposes a much smaller coordination burden on the overall...endeavour' (Baldwin & Clark 2006a, p. 191). To formulate it as a hypothesis:

Product modularity reduces coordination costs

Fig. 5.1 illustrates the hypothesised effect in the broader context of the research model that sums up the hypotheses derived from the literature review in chapter 2:

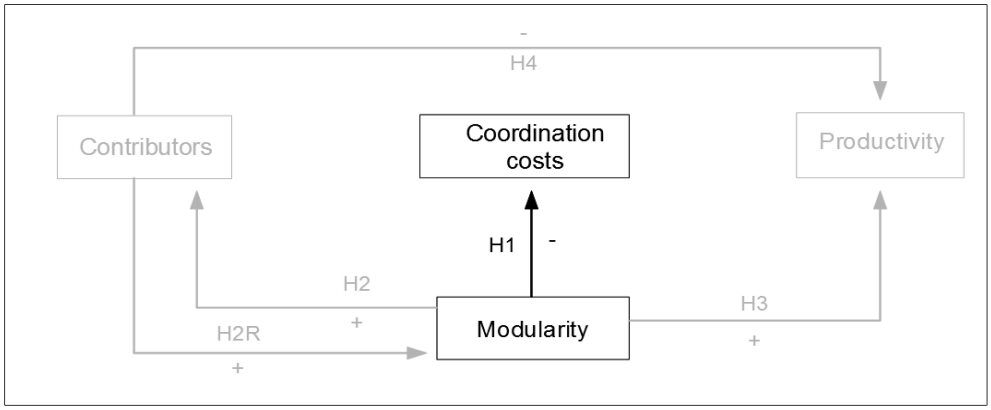


Fig. 5.1: Research model

Oddly enough, though this proposition has been reiterated time and again in the literature during the last fifteen years, there exists no record of a quantitative validation of the moderating effect of product modularity on coordination costs, nor of its falsification.[69] Despite the fact that several studies have looked at the effect of product modularity on coordination, none has attempted to quantify the claimed benefit of modular design. The only one using quantitative measures of coordination costs is Capiluppi and Adams' (2009) study of collaboration in the KDE Project, a large FOSS project. By tracking the communication paths among developers over time, weighting a communication path between any two developers based on the number of source code files on which they collaborated, Capiluppi and Adams (2009) ascertained that fewer than ten developers participated in the project's early stage of development, which was characterised by extensive communication within the group. But as the project started growing and the codebase was restructured with a view to increasing its modularity, 'communication compaction' (i.e. the average weight of path between developers) declined down to one third of its original value. In the last stage, when more than three hundred developers coalesced around the project, the compaction was still the same as when the project had no more than ten developers, that is to say, three hundred developers needed 'the same amount of communication as when the developers were only 10' (Capiluppi & Adams 2009, p. 274). Capiluppi and Adams qualified these findings by arguing that while hundreds contribute to large FOSS projects such as KDE, most of the work is actually done by a close-knit group of high-contribution participants known as core developers. These developers cannot dispense with active coordination: the need to coordinate their activities is made necessary by the extent of their involvement in the project. But unlike core developers, the coordination costs that encumber the work of peripheral contributors are considerably lower. The tasks they perform – reporting problems and contributing fixes – do not require of them to work as a close-knit group. Hence, those tasks are 'parallelisable': an infinite number of individuals can be simultaneously engaged in reporting bugs and fixing them (see also Raymond 1999). Put another way, the coordination costs involved in the periphery of the project are independent of group size. However, the degree of collaboration required for the development of new functionality is significantly higher, and so are the respective

69 Gershenson et al. (2003, p. 307) close their literature review by noting that they 'have not found a single experiment to quantify or at least prove the claimed benefits of modular product design'.

coordination costs. In the light of this analysis, modularity is what allows large FOSS projects to integrate a plethora of minute contributions – in the form of problem-reports and fixes – without exacerbating the organisational costs of collaboration among core developers (see also Benkler 2006; Capra et al 2008, p. 769). Although that is without doubt an important perspective on the function of modularity in FOSS development, however by so qualifying their results, Capiluppi and Adams (2009) evade the question whether modularity mitigates the need for active coordination between distinct product components and by extension between the developers working on them.[70]

Although the method employed by Capiluppi and Adams (2009) to collect and analyse activity logs from KDE's code repository as a factual documentation of economic activity, despite the shortcomings of their analysis, is not devoid of merit, we take a somewhat different approach to estimating coordination costs in FreeBSD. For that purpose, our choice of metrics and data sources is dictated by project-specific considerations. As mailing lists are the primary communication fora in FreeBSD (FreeBSD 2011b), the number of emails exchanged by developers is the most direct measure of coordination costs available in this setting. However, because project communications occur on a multitude of mailing lists,[71] each geared to different aspects of the project, identifying the one(s) centred on coordinating the development process is crucial. Through our review of the relevant literature, we were able to identify the *freebsd-current* mailing list as the central forum for coordination issues related to the current branch.[72] As researchers Holck and Jørgensen (2004) explain:

---

70 Actually, Capiluppi and Adams (2009) do not measure modularity: they assume that the restructuring of the codebase resulted in increased modularity. However, a more serious flaw in their work lies in the confusing, and at times contradictory, interpretation placed upon their findings. Consider, for instance, the findings they report in a follow-up paper in which 'communication compaction' is phrased as 'coordination cohesion'. Here they find that 'in this first phase [of KDE], fewer than 10 developers produce high cohesion scores, greater than 20' (Adams et al. 2009, p. 322). But when turning to the third and final stage of KDE's development, they mention that 'an apparent critical mass is achieved, requesting a coordination cohesion vastly larger than when found when the project had only 10 developers' (Ibid., p. 322) (indeed, by looking at the relevant plot in Fig. 2 in p. 323, one observes that cohesion rises from 20 up to 160 over time). This result, by showing that the volume of communication among developers rises over time, obviously contradicts their previous finding that communication compaction in the final stage is the same as in the first stage.

71 As of August 2011, there are 144 public mailing lists in use (<http://lists.freebsd.org/mailman/listinfo>)

72 The freebsd-current mailing list is archived online at <http://lists.freebsd.org/pipermail/freebsd-current/>

> For developers working on CURRENT, the mailing list freebsd-current is particularly important, as this is where all announcements of important changes to CURRENT will be given. Also, problems in building or running CURRENT will be posted to and discussed in this forum; these seem to account for around 75% of the list threads.

The FreeBSD Project (2011b) describes the list as follows:

> This is the mailing list for users of FreeBSD-CURRENT. It includes warnings about new features coming out in -CURRENT that will affect the users, and instructions on steps that must be taken to remain -CURRENT. Anyone running "CURRENT" must subscribe to this list. This is a technical mailing list for which strictly technical content is expected.

To ensure that freebsd-current is indeed centred on coordinating development processes, we selected in random 100 emails sent over the list during a time-period of five years (from March 2003 to March 2008). Most of them (i.e. approx. 70%) were indeed related to coordination costs triggered by changes in the product such as integration breakdowns ('broken builds') or problem-reports and suggested problem-solutions (i.e. bug-fixes) that need to be reviewed and tested by more developers before they can be incorporated into the project repository.[73] Having thus identified the freebsd-current mailing list as being the one most relevant to our inquiry, we resorted to using the number of emails sent over this list as an indicator of coordination costs in the FreeBSD development process. Hence, as coordination costs are proxy-measured by the number of emails sent over the list, the hypothesis can be reformulated as:

Product modularity reduces the number of emails sent

To further refine the hypothesis, we use the propagation cost of the codebase – which captures the percentage of files that are likely to be affected on average when a change is made to a randomly selected file – as an indicator of modularity

73 Some examples are exhibited in section **Measuring coordination costs** in chapter 3.

(as in MacCormack et al. [2006] and Milev et al. [2009]).[74] Therefore, since a decrease of propagation cost is tantamount to an increase of modularity, the hypothesis can be readily operationalised as follows:

> *As the propagation cost decreases, the number of emails sent decreases* (H1-operationalised)

QUALITATIVE ANALYSIS

Our analysis of descriptive statistics begins with an estimation of coordination costs based on the number of emails sent over the freebsd-current mailing list over time. As there are no archives of the list available for the years before 2003, our analysis focuses on the five year period from 2003 to 2008. As we can see in Fig. 5.2 below, 17656 emails were sent over the list in 2003. The volume of communication on the list increased by 52.3% to 26890 emails in 2004, but has since declined steadily down to 9839 emails in 2008. Supposing that the number of emails sent over the list is a valid indicator of coordination costs in the project, these numbers suggest that coordination costs in the project have decreased over time.

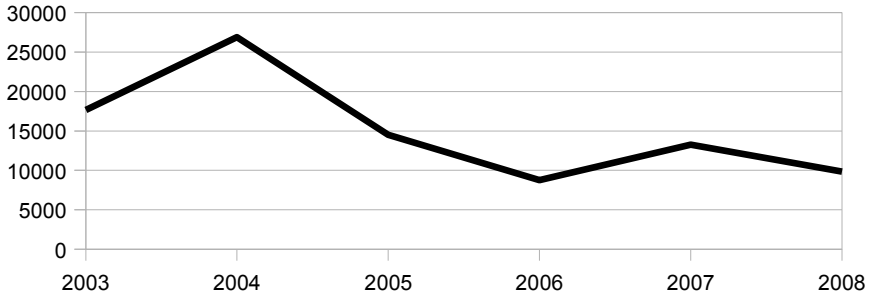


Fig. 5.2: Number of emails sent over freebsd-current mailing list, 2003-2008

Having ascertained that coordination costs decrease over time, let us now examine the degree of modularity of FreeBSD as reflected in the propagation cost of the codebase. As Fig. 5.3 below illustrates, the propagation cost doubled from 10%

74 For an elaborate discussion of the propagation cost metric, see section **Measuring modularity** in chapter 3.

in 2003 to 21% in 2006, at which point it tends to stabilise since. The increase of propagation cost in the space of the first three years implies that FreeBSD evolved toward lower levels of modularity in that period.

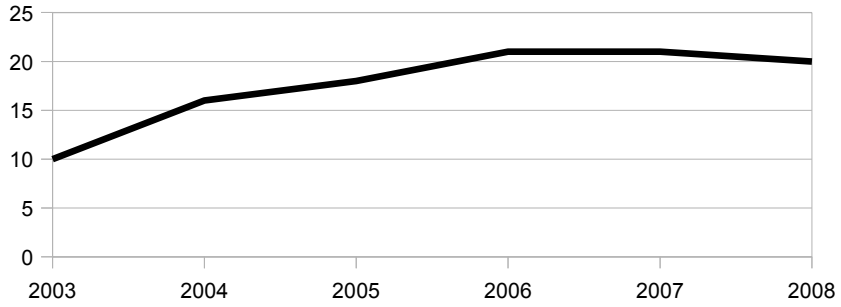


Fig. 5.3: Propagation cost (%)

Now, let us contrast the number of emails sent over the list with the propagation cost of the FreeBSD codebase in Fig. 5.4 below. We see that the number of emails has been decreasing since 2004 while the propagation cost, by contrast, has been increasing. As an increase of propagation cost reflects a decrease of modularity, contrasting the propagation cost of the codebase with the volume of emails sent over the mailing list indicates that decreasing levels of modularity correlate with lower – rather than higher, as one would expect – coordination costs.

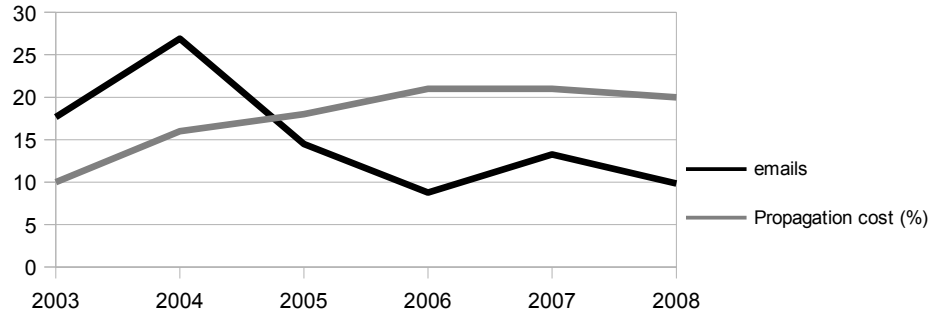


Fig. 5.4: Coordination costs (emails sent over freebsd-current mailing list) versus modularity (proxy-measured by propagation cost)

The above results, should they be taken at face value, lead to the conclusion that both modularity and coordination costs tend to decrease over time. This conclusion is, of course, nothing short of counter-intuitive: modularity theory holds that a

decrease of modularity – as that observed by looking at the tendential rise in propagation cost in Fig. 5.4 – is impossible to bring about a decrease of coordination costs (such as that implied by the decrease in the number of emails sent over time). From the perspective of modularity theory, the increase in the space of three years in the percentage of files likely to be affected when a change is made to any one file by 100% implies that the need for active coordination increased analogously. Yet, our results are unsupportive of that syllogism: our analysis of descriptive statistics, by showing a tendential fall in the levels of both modularity and coordination costs, contradicts the claims made in the literature.

It is difficult for these findings to be squared with the preliminary conclusions gleaned from prior descriptive research in FreeBSD as well as from internal documents released by the project. There are a number of strong indications that militate against the conjecture that coordination costs in the project are decreasing over time. First, in 2001 the project started using quarterly status reports, citing the need to alleviate problems of information overload attendant upon increasing group size. As the first of these reports stated, 'the FreeBSD developer community has grown, and the rate of both mailing list traffic and tree modifications has increased, making it difficult even for the most dedicated developer to remain on top of all the work going on in the tree...[The] Status Report attempts to address this problem' (FreeBSD 2001b). Second, both Jordan Hubband and Mike Smith underlined the increasing difficulty of resolving conflicts among committers as the cause of their resignation from the core team in April and May 2002 respectively (KernelTrap 2002; Lehey 2002). Third, since 2002 Murray Stokely (2002), primary release engineer for most of the FreeBSD 4.x releases, has been constantly stressing the need to formalise FreeBSD's release engineering activities as a response to the coordination costs accompanying increased scale. What, in other words, militates against the conjecture – derived from the decrease in the number of emails sent via the freebsd-current mailing list over time – that coordination costs tend to decline is the increased scale of the project. As we have seen, the (*src*) committers base has expanded dramatically from 16 committers in 1994 to 209 in 2005 (see **Fig. 4.9: Committers** in chapter 4). Since, according to *Brooks' Law*, adding more developers to a project results in an exponential increase in coordination costs (Boehm 1981; Brooks 1995), the sheer magnitude of the increase in the number of FreeBSD developers with commit rights is strongly indicative of a concomitant increase in coordination costs. All the indications of rising coordination costs so far enumerated point to the possibility that the number of emails sent over the freebsd-

current mailing list is problematic as an indicator of coordination costs in the project. It is possible, as FreeBSD committers make frequent and systematic use of more than just one mailing lists to coordinate their activities, that the email traffic carried through the freebsd-current mailing list captures only a small portion of the overall coordination costs in the project. For example, a mailing list to which all committers are subscribed is *freebsd-developers*. However, because it is used for 'discussions of work in progress [that] are not suitable for open publication and may harm FreeBSD', discussions on the list are closed to the public (FreeBSD 2011a). But as there is no record of them available, there is no way of estimating the volume of communication occuring on the list.

Not being sure how to interpret the aforementioned findings, we proceeded to contrast coordination costs with modularity at a more refined level of analysis, focusing on individual modules rather than on the codebase as a whole. *Firewire* was the first module we examined.[75]

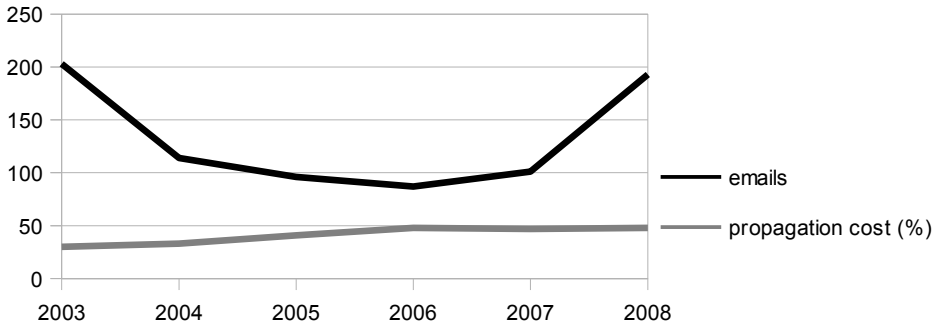


Fig. 5.5: Coordination costs (emails sent over freebsd-firewire mailing list) versus modularity (proxy-measured by propagation cost) in freebsd-firewire

However, the results were similar. As can be seen in Fig. 5.5 above, the propagation cost of the firewire module increased from 30% in 2003 to 48% in 2006, at which point it has tended to stabilise. The number of emails sent over the freebsd-firewire mailing list, by constrast, decreased from 203 in 2003 to 87 in 2006, thereafter increasing up to 193 emails in 2008. These data indicate that although the degree of modularity of the firewire component decreased in the first three

75 The freebsd-firewire mailing list is archived online at <http://lists.freebsd.org/mailman/listinfo/freebsd-firewire>. As there are no archives of the list available for the years before 2003, our analysis is limited to the five year period from 2003 until 2008.

years, the coordination costs involved in its development in that period decreased as well.

After firewire, we looked at the *usb* module.[76] As Fig. 5.6 shows, the propagation cost of usb hovered at about the same levels from 2005 until 2008, while the number of emails sent over the freebsd-usb mailing list decreased in the first year from 1298 to 1036 messages, thereafter increasing up to 1582 emails in 2008.

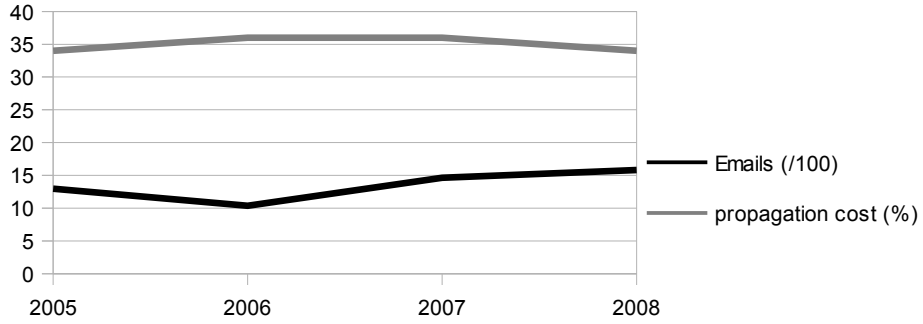


Fig. 5.6: Coordination costs (emails sent over freebsd-usb mailing list) versus modularity (proxy-measured by propagation cost) in freebsd-usb

Let us look at these results more closely. From 2005 until 2006, both the degree of modularity of the usb component (as shown by the increase of propagation cost from 34% to 36%) and the coordination costs involved in its development (from 1298 to 1036 messages) decreased. From 2007 until 2008, both the degree of modularity (as shown by the decrease of propagation cost from 36% to 34%) and the coordination costs involved in its development (from 1463 to 1582 messages) increased. Hence, rather than lead to an increase of coordination costs as theorised in the literature, we see that the decrease of modularity in the period 2005-2006 is accompanied by a decrease of coordination costs. Similarly, the evolution of usb toward higher levels of modularity in the period 2007-2008 is paralleled by an increase of coordination costs, rather than by a decrease as predicted by modularity theory.

We were hoping that the statistical analysis, by focusing on the level of individual modules rather than on the entire codebase, would permit a more

76 The freebsd-usb mailing list is archived online at <http://lists.freebsd.org/mailman/listinfo/freebsd-usb>. As there are no archives of the list available for the months before October 2004, our analysis is limited to the three year period from 2005 until 2008.

rigorous treatment of these questions. Unfortunately though, with the exception of freebsd-firewire and freebsd-usb, no other mailing list of those specific to the twenty-nine FreeBSD modules included in our sample[77] is publicly archived. Hence, the number of observations we were able to collect – based on five years of publicly archived data for freebsd-firewire and three years for freebsd-usb – are insufficient in order to perform a regression analysis. Consequently, it is impossible to test statistically the effect of the propagation cost of individual modules on the number of emails sent over the mailing lists centred on their development.

CONCLUDING REMARKS

Some remarks need to be made at this point. First, the qualitative analysis we conducted using descriptive statistics yielded results that challenge the validity of the hypothesis that modularity reduces coordination costs. Some of these results – notably, the decrease in the number of emails sent over time via the freebsd-current mailing list – may appear contradictory in the light of strong indications of rising coordination costs furnished by bibliographical research into documents released by the FreeBSD project and its developers. A tentative explanation for the tendential fall in the number of emails is that freebsd-current is not the only channel of coordination used by committers. Consequently, measuring the number of emails sent over freebsd-current captures but a portion of the overall coordination costs in the project. Second, a more rigorous examination of the effect of modularity on coordination costs at the level of individual modules could not be successfully attempted. As archives of communications are available for only two of the twenty-nine modules included in our sample, the number of observations we were able to collect is not sufficient for statistical analysis. Not being able to estimate coordination costs for the modules comprising our sample, the hypothesis could not be statistically tested.

However, the fact that we looked at three instances (i.e. at the level of the project as a whole and at two components separately) and found no empirical support for the hypothesis that higher levels of modularity correlate with lower levels of coordination costs points to a possible 'over-simplification' of the hypothesis as formulated in modularity theory. The fact that in all three instances both modularity and coordination costs move in the same direction, providing thus no evidence for the theoretical prediction that higher levels of modularity lead to

77 For the procedure used to draw the sample, see section **Sample construction** in chapter 3.

lower levels of coordination costs, implies that the hypothesis cannot be confirmed.

Based on the results of our analysis of descriptive statistics, *H1* cannot be confirmed.

In the next chapter, we attempt to test hypothesis *H2*, which holds that product modularity increases the potential number of contributors to a project.

CHAPTER 6: MODULARITY AND GROUP SIZE IN FREEBSD

INTRODUCTION

An assumption underlying much of the research in modularity from an organisational and software engineering perspective – a real-world demonstration of which large free and open source software (FOSS) projects are considered to offer – is that modular product design is required for large-scale collaboration in a distributed product development environment.

Testifying to the link between product modularity and group size, a simulation study of the interplay between codebase architecture and degree of participation in FOSS development by Baldwin and Clark (2006b) found that:

> Projects not worth undertaking under a monolithic architecture may attract tens or even hundreds of self-interested developers under a sufficiently modular architecture (Baldwin & Clark 2006b, p. 1123).

Because changes can be made to distinct modules without undermining the functionality of the product as a whole, a modular architecture enhances the 'value options'[78] embedded in a codebase, as opposed to a monolithic (i.e. non-modular) architecture where the tendency of changes to propagate throughout the product results in low option values (Baldwin & Clark 2006b, pp. 1117–1118). Hence, 'as the number of modules and the option values embedded in the system increase, more developers will work in equilibrium' (Baldwin & Clark 2006b, p. 1122). Accordingly, the effect of product modularity on the size of the group developing

78 An *option*, according to modern finance theory, is 'the right but not the obligation to choose a course of action and obtain an associated payoff' (Baldwin & Clark 2006b, p. 1117). This conceptual instrument is used by Baldwin and Clark to model the value of modular product design upon the assumption that 'a new design creates the ability but not the necessity – the right but not the obligation – to do something in a new way...In this sense a new design is an option' (Ibid.). Thus, the analysis of value options in their work is geared to assessing the extent that the architecture of a systemic product encourages experimentation with regard to viable alternatives (i.e. substitutes) at the module-level. The same analytical approach can be found in Sullivan et al. (2001) and LaMantia et al. (2008).

the software is so significant that,

> Open source codebases that are more modular or have more option value will attract more voluntary contributions (effort) than codebases that are monolithic or have low option value' so that 'the more modular and option-rich the underlying designs, the larger and more active the user-innovator communities are likely to be (Baldwin & Clark 2006b, p. 1126).

In the final analysis, as Langlois and Garzarelli (2008) put it more recently, 'a modular system increases the potential number of contributors'. On the basis of these claims, the following hypothesis can be stated:

Product modularity increases the potential number of contributors to FreeBSD (H2)

Fig. 6.1 situates the hypothesis in the context of the research model derived from the review of the literature on modularity in chapter 2:

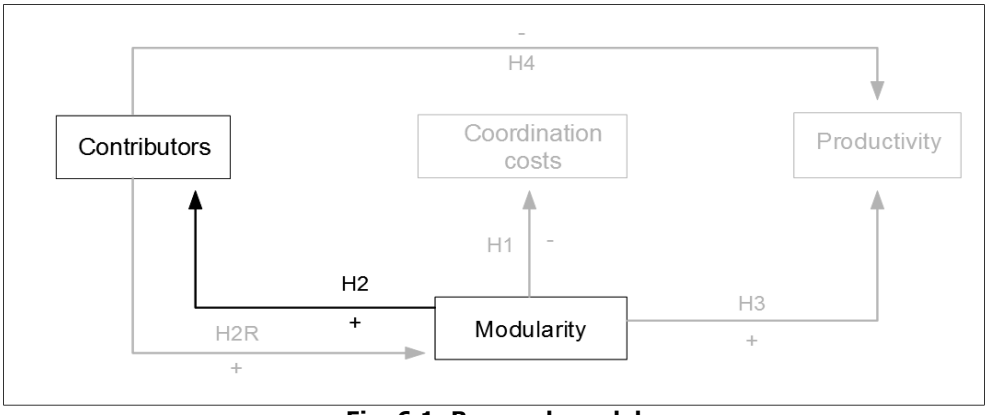


Fig. 6.1: Research model

The link between product modularity and group size was underlined in an empirical study of the modular re-design of the Mozilla Web browser, which concluded 'that different modes of organization are associated with [product]

designs that possess different structures' (MacCormack et al. 2006). Prior to the re-design (in 1998), Mozilla was developed by a close-knit group of programmers on the payroll of Netscape Corporation. Then, in 1997 Netscape released its source code for free under an open source license in an attempt to undercut competition by distributing production requirements across the network. A modular re-design was deemed necessary to harness the power of distributed development by a loosely-coupled network of volunteer programmers scattered around the world. It was motivated by the conscious need for a product architecture conducive for large-scale collaboration over the Internet. Consistent with the project's expectations,

> the redesign to a more modular form was followed by an increase in the number of contributors (MacCormack et al. 2006, p. 1028).[79]

The authors of the study, MacCormack, Rusnak and Baldwin refrained however from an one-sided, monocausal interpretation. The results of their inquiry, they pointed out, doubtlessly reinforce the importance conferred upon product modularity for giving shape to decentralised organisational structures. But at the same time they were attentive to the possibility that product structure *evolved* to reflect the production environment in which it was now being developed, the decisive factor of which was a large, informally organised and geographically distributed developers' base. By emphasising the effect of group dynamics on product structure, the terms of the proposition are reversed and the proposition can be thereby reformulated as follows:

An increase of contributors to a FOSS project results in an increase of modularity

The diagram in Fig. 6.2 illustrates the hypothesis by reference to the research model into which the claimed benefits of modularity crystallise:

79 Of note, the findings of Mockus et al. (2002) corroborate the view that Mozilla's modular re-design led to an increase of contributors.

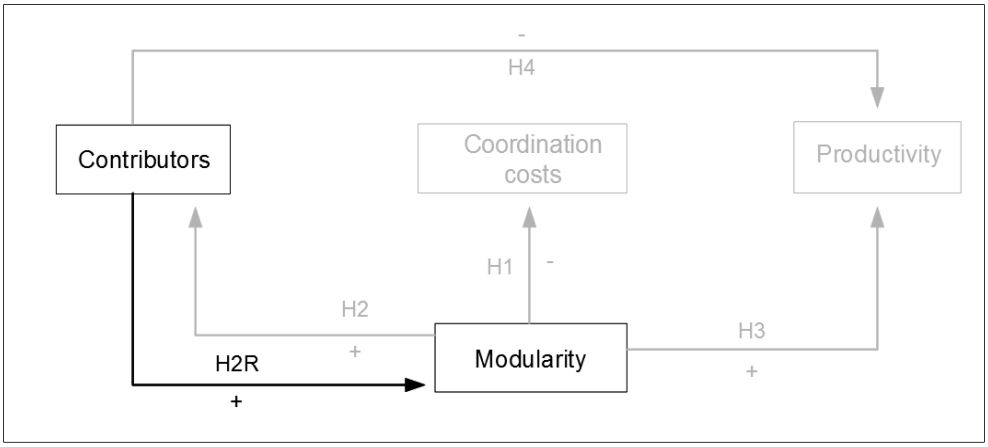


Fig. 6.2: Research model

Further empirical support for the hypothesis (*H2R*) that an increase of contributors to a FOSS project leads to higher levels of product modularity comes from a follow-up study by the same researchers, which compared five paired software products with similar functions and levels of sophistication, concluding that,

> larger, more distributed teams tend to develop products with more modular architectures (MacCormack et al. 2008a, p. 2).

In all five pairs they examined, using the products' propagation cost[80] as a proxy for modularity, they found that,

> the open source product is more modular than that of a product of comparable size developed by a smaller, more centralized team. Furthermore, in the one open source product that possesses a relatively high propagation cost, the anomaly can be explained [by that it] is not the result

80 As defined by MacCormack et al. (2006, p. 1020), the *propagation cost* captures 'the degree to which a change to any single element [that is, file] causes a (potential) change to other elements in the system, either directly or indirectly (i.e. through a chain of dependencies that exist across elements)'.

> of a large, distributed team. Rather, the pattern of development is more consistent with that of a small co-located team (MacCormack et al. 2008a, pp. 20-21).

In a nutshell, *the larger the group of contributors to a FOSS project the more modular the product.* Since in the work of MacCormack et al. the degree of product modularity is captured by the products' propagation cost, the proposition can be stated alternatively as follows: *the larger the group of contributors the lower the propagation cost,* which, being quantitatively measurable, forms a hypothesis we can immediately test:

> *As the number of contributors to FreeBSD increases, propagation cost decreases* (H2R-operationalised)

QUALITATIVE ANALYSIS

Let us begin with *H2* which holds that modularity increases the potential number of contributors. As in the studies by MacCormack et al., modularity is proxy-measured by propagation cost. By examining FreeBSD's propagation cost over time in Fig. 6.3 below, if we exclude the period from 1999 until 2002 during which it declines, we see that propagation cost tends to rise over time, thereby indicating that FreeBSD becomes *less* modular over time.

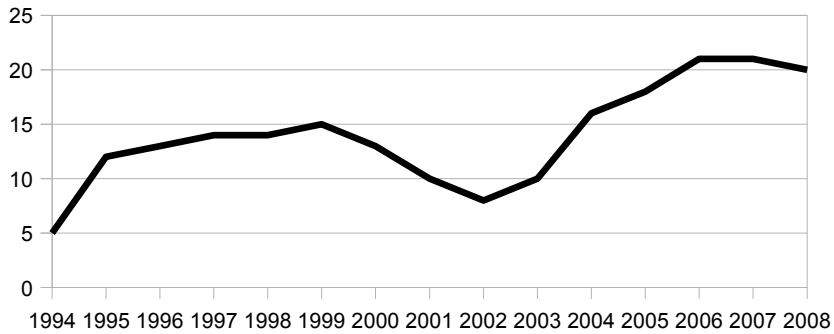


Fig. 6.3: Propagation cost (%)

Let us now examine the number of committers (i.e. contributors who have the

right to commit code to the project repository) who contribute code to the project.[81] Looking at the growth of the committers base in Fig. 6.4 below, one observes that the number of committers checking-in code to the repository increased more than tenfold (1250%), from 16 committers in 1994 to 198 in 2007, peaking at 209 in 2005.

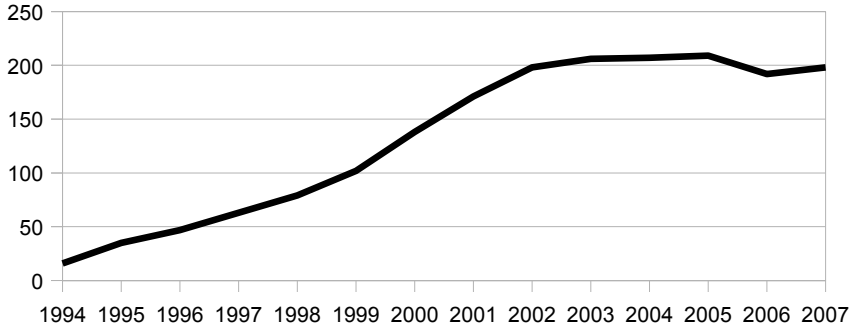


Fig. 6.4: Committers (*src*)

Given that in the space of thirteen years the committers base has grown considerably from a small group of sixteen to about two hundred, one would have expected to find a tendency for the software's propagation cost to fall over time, assuming that the hypothesis holds. Our data however do not point to this direction. As can be seen in Fig. 6.3 above, the propagation cost of FreeBSD tends to rise over time. Finding therefore that FreeBSD's propagation cost does not decline as committers increase, weakens the support for the hypothesis (*H2*) that modularity increases the potential number of contributors. Although the number of contributors to the project does indeed increase, this phenomenon is not accompanied by a concurrent increase of modularity. Examining the growth of the committers group alongside the propagation cost of the codebase (*Fig. 6.3, 6.4 above*) shows that both committers and propagation cost increase in the course of development, which of course runs counter to the results reported by MacCormack et al. It is evident that the expansion of the committers group from 16 to about 200 members is not accounted for by increasing levels of modularity, as would be suggested by a tendential fall in propagation cost. Quite the contrary, we observe a

81 We counted only committers who contribute to the *src* tree and excluded those involved in the *ports* and *documentation* tree. The rationale for this choice was that work on the latter two areas does not consist in new code development.

tendency for the propagation cost to rise, signifying thus lower levels of modularity. Assuming that propagation cost is a valid indicator of modularity, then it suffices to contrast it with the growth of group size to challenge the hypothesis (*H2*) that product modularity increases the potential number of contributors. And conversely, the same qualitative analysis of descriptive statistics suffices to cast doubt upon the hypothesis (*H2R*) that an increase of contributors to a FOSS project leads to higher levels of product modularity.

Hence, replicating the MacCormack et al. methodology for FreeBSD leads to conclusions diametrically opposite to those MacCormack et al. arrived at in their study of Mozilla. Whereas a relation of inverse proportionality between propagation cost and committers is manifest in the Mozilla project, that is by no means the case in FreeBSD. How can this discrepancy in results be explained? On first impression, a likely explanation is that the two products, despite being developed by large groups, simply differ in their architectural structure: Mozilla becomes more modular over time, while FreeBSD evolves in the opposite direction. Accepting this explanation implies that product modularity is not a necessary condition for a large group to coalesce around a distributed development process (and conversely, that distributed development by a large group does not lead to higher levels of modularity). A problem more fundamental than the inadequacy of this explanation in accounting for increasing group size is posed by the level of analysis itself: examining the relation between group size and modularity from the vantage point of the project as a single organisational entity does not allow for a rigorous analysis, as it leaves out of consideration the fact that the organisational impact of modularity is located at the level of the modules making up the product. Up to now we looked at the FreeBSD project holistically, treating it as an integrated whole, while it is at the level of individual modules that modularity is held to have its strongest effect by allowing for their independent development by autonomous groups. Our inquiry must therefore turn to individual modules as an appropriate unit of analysis.

QUANTITATIVE ANALYSIS

To examine the effect of modularity on group dynamics at the level of individual modules, we carried out a regression analysis of a (panel) dataset consisting of a stratified random sample of twenty-nine FreeBSD modules[82] with observations

82 See section **Sample selection** in chapter 3 for a full description of the procedure employed to

spanning fourteen years of development activity from 1994 to 2008. Table 6.1 below lists the modules included in the analysis.

Small-scale	Medium-scale	Large-scale
cardbus	aac	cd9660
digi	agp	coda
joy	devfs	firewire
netatalk	hpfs	netinet6
netipsec	msdosfs	nfsclient
nfs4client	net80211	nfsserver
pccard	netncp	procsfs
random	ntfs	usb
rpc	nwfs	vm
xe	pseudofs	-

Table 6.1: FreeBSD modules included in regression analysis

For the regression analysis, we used the number of committers as dependent variable and propagation cost as independent variable. Furthermore, the correlation between observations was taken into account: instead of treating each observation as independent of all other observations in the dataset, it was obvious that each module ought to be considered as a separate software project with its own development process. Simply put, observations pertaining to the same module are correlated because the behaviour of developers of the same module is likely to be interrelated. To account for this *intraclass correlation* (Fisher 1925, chapter 7), as distinct from a Pearson correlation which is between two variables, each *module* is used as a group (i.e. cluster) variable for the regression analysis.[83]

The empirical model in Fig. 6.5 illustrates the directionality of effect as hypothesised in *H2*:

draw the sample.

83 The same group variable is used throughout all regression analyses presented in this chapter.

Fig. 6.5: Empirical model H2

Before we proceed to the test results, however, let us elucidate the heuristics used to interpret them. The R-squared of the regression (also known as coefficient of determination) is the fraction of the variation in the dependent variable that is accounted for (or predicted by) the independent variables. In a regression like that below with a single independent variable, it is identical with the square of the correlation between the dependent and independent variable. The R-squared is generally of secondary importance, unless the purpose of the regression is to make accurate predictions. What is more important is (a) the P value for the regression as a whole, which indicates the overall (statistical) significance of the empirical model and (b) the P value of the independent variable, which tells us how confident we can be that it is correlated with the dependent variable (Dallal 2001; DSS 2007). In keeping with the above rules of thumb, our interpretation of regression results is based mainly on the P value for the regression model and the P value of the independent variable.

```
Random-effects GLS regression                   Number of obs      =       280
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0014                         Obs per group: min =         5
       between = 0.0010                                        avg =       9.7
       overall = 0.0004                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      0.28
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.5961

------------------------------------------------------------------------------
  committers |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
propagatio~t |    1.353946   2.554379     0.53   0.596    -3.652544    6.360436
       _cons |    8.276264   1.432741     5.78   0.000     5.468143    11.08438
-------------+----------------------------------------------------------------
     sigma_u |   5.5590533
     sigma_e |   4.3589837
         rho |   .61925276   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.2: Regression results–Effect of Modularity on Contributors

We can now continue with the analysis of results: the regression we ran indicated no model significance (p = *ns*), suggesting that group size is not affected

by the degree of modularity as captured by the propagation cost (see Table 6.2 above). To make sure that potential time-lags were not overlooked, we proceeded to a lag transformation of the predictor, so that we could test the effect of modularity in year=*t* on group size in year=*t+1*. Such a transformation is not arbitrary: it reflects the logical order of the hypothesised causal relationship (i.e. if A causes B, then by definition A precedes B) and so by establishing directionality it allows a refinement of the empirical model. Nevertheless, neither did the test with the lag transformation improve model significance (p = *ns*).

Following this step, we decided to include two additional indicators of modularity in the empirical model: *external_dependencies_lag* (henceforth abbreviated to *ext_dependencies_lag*) and *integrality_index_lag*. Fig. 6.6 illustrates the revised empirical model.

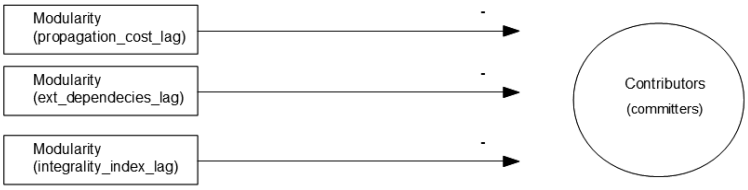


Fig. 6.6: Revised empirical model H2

Compared to the propagation cost, the number of *external dependencies* is a rather simplistic indicator of modularity, for the computation of the former takes account not only of the number of dependencies but also of their pattern of propagation. Given however that a product's degree of modularity is determined by the dependency relations between its components (i.e. modules), using the number of external dependencies as a rough, yet straightforward, index of modularity is not unwarranted. The last indicator of modularity we added to the model, *integrality index*, is the ratio of a module's external dependencies to internal dependencies. As a successful modularisation implies that modules contain most, if not all, of the dependencies internally and the dependencies between separate modules are eliminated or minimised (a practice known as *clustering*)(Parnas 1972; Sharman & Yassine 2004; Simon 1962), this metric captures the extent that dependencies have been effectively encapsulated within modules. Thus, its advantage compared to the other two indicators of modularity included in the model is that the importance of clustering, which eludes an analysis of modularity based either on propagation cost

or external dependencies, is taken into account.[84] Prior to running the regression, we wanted to make sure that our independent variables measure different dimensions of the same construct rather than essentially the same thing. Including in the regression variables that are near perfect linear combinations of one another is a problem when the goal is to understand how the various independent variables affect the dependent variable because the estimates of the coefficients for the regression become unstable. This is known as the multicollinearity problem. Thus, we ran a Variance Inflation Factor (VIF) test, which is commonly used for the purpose of assessing multicollinearity. This showed that the VIF values of all independent variables are smaller than 10, thereby confirming that our predictors are not collinear and therefore can be included in the regression.

Variable	VIF	1/VIF
ext_dependencies_lag	1.21	0.824107
propagation_cost_lag	1.16	0.858636
integrality_index_lag	1.05	0.954472
Mean VIF	1.14	

Table 6.3: VIF test for regression of committers on ext_dependencies_lag, propagation_cost_lag and integrality_index_lag

```
Random-effects GLS regression                   Number of obs      =       242
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0881                         Obs per group: min =         4
       between = 0.5142                                        avg =       8.3
       overall = 0.4255                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(3)       =     44.97
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
  committers |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
propag~t_lag |  -10.46891   3.928068    -2.67   0.008    -18.16779   -2.770041
ext_de~s_lag |   .0865351   .0134067     6.45   0.000     .0602585    .1128118
integrali~ag |  -.1443794    .089027    -1.62   0.105    -.3188692    .0301105
       _cons |   7.020432   1.496355     4.69   0.000     4.087631    9.953234
-------------+----------------------------------------------------------------
     sigma_u |  4.1853288
     sigma_e |  3.7428374
         rho |  .55563926   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.4: Regression results–Effect of Modularity on Contributors

Moving on to the regression itself, as Table 6.4 above shows, adding ext_dependencies_lag and integrality_index_lag to the model yields results that indicate strong model significance ($p < 0.001$), suggesting that modularity indeed

84 For a more extensive description of the metrics, see section **Measuring modularity** in chapter 3.

affects the number of contributors.

Let us look more closely at the test results. While the preceding regression analysis found no significant effect of propagation_cost_lag ($\alpha = 0.05$, $\beta = -1.48$, $p = 0.54$) on committers, it now appears to have a strong effect. Specifically, the coefficient for propagation_cost_lag is negative and significant, suggesting that lower levels of modularity (as signified by an increase of propagation cost) lead to a decrease of contributors. Hence, it supports the hypothesis that higher levels of modularity result in increasing group size.

As for the effect of integrality_index_lag on the number of committers, it *leans* toward significance ($\alpha = 0.05$, $\beta = -0.14$, $p = 0.10$).[85] The coefficient for integrality_index_lag is negative, suggesting that lower levels of modularity (as indicated by an increase of integrality index) result in a decrease of committers. Consistent with theory, this result provides support for the hypothesis that higher levels of modularity result in increasing group size.

A strong effect on committers is also exerted by ext_dependencies_lag. Surprisingly enough, the coefficient for ext_dependencies_lag is positive and significant, indicating that when a module's external dependencies increase, so too do committers working on the module, a result which contrasts sharply with our theoretical assumptions. Modularity theory predicts that modules loaded with external dependencies will attract fewer contributors than modules relatively less encumbered, on account of the coordination costs involved in managing interdependencies between modules. According to Baldwin and Clark, modularity increases the incentives of developers to join and remain engaged in the development of a module by enabling its independent development. Thus, to the extent that modules are independent of one another so that changes in one module do not affect the others, their model of rational choice in FOSS development predicts that 'the more modular...the underlying designs, the larger and more active the user-innovator communities are likely to be' (Baldwin & Clark 2006b, p. 1126). Where however this 'separation of concerns' – to borrow a phrase that Parnas (1972) uses to clarify the criterion of an effective modularisation – is thrown into disarray by the opaqueness of external dependencies, that is clearly no longer the case: by the logic of Baldwin and Clark's modularity theory, the more the external dependencies of a module, the smaller and less active its group of contributors is likely to be. That is only consistent, of course, given the higher learning costs that

85 Abelson (1995, pp. 74-75) suggests that, instead of stating as 'marginally significant' results at the level of $.05 < p < .15$, they can be more precisely stated as *leaning* in significance and as *hinting* about significance for $.15 < p < .25$.

contributors to a module with many external dependencies have to shoulder in order to familiarise themselves with the interactions between that and other modules, and the communication costs forced upon them by the need to coordinate their work with developers working on other modules.[86] Yet, our results reveal the opposite tendency: they show that group size increases when external dependencies increase, so that the more dependent on other parts of the product a module is (i.e., the more it uses functionality contained in other modules), the more contributors are attracted to its development process.

Although at first glance this result is counter-intuitive, it is consistent with two causal mechanisms, which in practice are likely to operate in parallel. The first is that the tendency of dependencies to rise cannot be left unattended: as the proliferation of dependencies has a degrading effect on product structure, their management constitutes a high priority maintenance task. According to the 'law of increasing complexity' (Belady & Lehman 1976; Lehman 1980; Lehman et al. 1997; Lehman & Ramil 2001), the complexity of a software product increases in proportion with the volume of changes made to it so 'that large-program structure must not only be created but must also be maintained if decay is to be avoided or, at least, postponed' (Lehman 1980).[87] Consequently, unless a conscious effort is made to limit the degrading effect on product structure of modifications accumulating over time, the larger a software system becomes the more pervasive shall interdependencies be in its development process, exacerbating coordination problems and encumbering further development. This perspective on software maintenance helps explain why the development of every new major branch (version) of FreeBSD involves an extensive architectural clean-up (Loli-Gueru 2003). In a software development environment characterised by low levels of structure (i.e. a development process of a non-modular product), as Banker and Slaughter (2000, p. 237) point out, 'maintenance effort and errors are higher because of increases in the number of relationships that a maintainer must understand and the difficulties in tracing interdependencies' between modules. Therefore, as product 'structure influences the efficacy of comprehension' (Ibid., p.

86 For a discussion of the learning and coordination costs incurred by contributors to FOSS projects in which the principle of information hiding has not been properly implemented, see Rusovan et al., op. cit.

87 The basic premise is that the chronic accumulation of changes through which software systems evolve and grow larger, has a degrading effect on product structure (i.e. architecture) 'to the point where the system can no longer be...maintained and enhanced unless and until redesign and cleanup or reimplementation is undertaken' (Lehman 1980, pp. 216-217).

236),[88] an elegant coding environment is preferable to one in which the practice of programming is obfuscated by the occurrence of interdependencies between tasks. Such a coding environment, however, takes effort to maintain. From this vantage point, an increase of a module's external dependencies is likely to prompt more contributors to concentrate on checking the growth of dependencies, increasing thus the size of the group working on the module.

A parallel interpretation of this phenomenon is that some modules are continuously upgraded. And so they accumulate changes not only during their early development stage but throughout their entire life-span. Considering that modules characterised by a high frequency of change are bound to accumulate more dependencies than modules relatively less subject to change, it is likely that modules with many dependencies attract more contributors because they manifest a high rate of technical change: their constant change signals to potential contributors that the development potential of the module has not been exhausted. This phenomenon is not new: several attempts have been made to illuminate its underlying causes through the analytical distinction between core and peripheral components. According to this theorisation of technical change, systemic products are composed of core and peripheral components: core components are developed first, then come peripheral components which are dependent upon the former. To illustrate, consider the familiar example of a car: the engine, the steering wheel and the metal body are the core components upon which all other components depend. As the early history of car design illustrates, 'once design converged to a fixed set of core concepts components (gasoline engines, steering wheels, and metal bodies), the design of core components were no longer subject to dispute, and innovations shifted towards low-pleiotropy[89] peripheral components to fine-tune very specific functions (lamps, belt, sets, interior, catalyst, and so on) and to incrementally refine the core technologies underlying the core components (pistons, fuel inlet, and so on)' (Murmann & Frenken 2006, p. 942, footnote 7). This pattern of product evolution is not limited to the car industry, of course: the shift of the locus of development from core to peripheral components has been underlined in studies of personal computers and VCRs, to name but two examples.[90] Viewed in the context

88 Consequently, modularity 'can be seen as a means to facilitate knowledge sharing by making the structure of code explicit and observable' (Capra et al. 2008, p. 769).

89 Although Murmann and Frenken (2006) do not explicitly consider the directionality of dependency relations, *pleiotropy* emphasises *in-degree* external dependencies (i.e. being used by...).

90 For a treatment of technical change in the early history of car design, see Clark (1985). For a description of the development of the IBM 360 computer based on the core-periphery model, see

of software development, the core consists of modules containing functionality that is heavily used by other modules; modules containing functionality that is seldom if ever used by other modules belong to the periphery.[91] In terms of the product's dependency relations, core are modules with many *in-degree* dependencies (i.e. being used by...) and few *out-degree* dependencies (i.e. using...), while peripheral are those with few in-degree and many out-degree dependencies respectively. Core modules form the epicentre of the early development of systemic products, but once they reach a definite level of maturity, development effort turns to new modules, which are adapted to the core ones: plugged, so to speak, into the periphery of the product structure. Reinforcing this interpretation, a recent empirical study of how dependencies relate to the rate and direction of technical change in thirty FOSS (Java) projects,[92] found that development activity gravitates toward modules with many out-degree dependencies, while modules with few out-degree dependencies accumulate changes early in their life-span, thereafter stagnating (von Krogh et al. 2009). From this point of view, the positive effect of (out-degree) external dependencies on committers that the regression analysis highlights is accounted for by the migration of development activity from core modules to new modules, which make extensive use of functionality contained in the former.[93]

Following the regression analysis we have just described, we attempted to further refine the empirical model by controlling for environmental factors such as the modularity and complexity of the broader FreeBSD project. Thus, the project-level variables *propagation_cost_j_lag* and *external_dependencies_per_module_j_lag* (henceforth abbreviated to *ext_dependencies_per_mod_j_lag*) were added to the model. The rationale for including propagation_cost_j_lag as a proxy for project-level modularity is that the development of individual modules is embedded within the development process of the product as a whole, and so (the degree of modularity of) the broader production environment ought to be taken into account when inquiring into the determinants

---

Baldwin and Clark (2000). On the relationship between core-periphery and technical change in VCRs, see Rosenbloom and Cusumano (1987).

91 Conversely, modules heavily reliant on functionality contained in other modules are *peripheral*; while modules that make minimal use of functionality contained in other modules are *core*.

92 Twenty-eight projects were selected from the Sourceforge repository; the remaining two were launched by IBM.

93 It is important to mention that the computation of all metrics used in the present study is based on out-degree (i.e. using...) dependencies; in-degree (i.e. being used by...) dependencies were not considered.

of group size at the module-level. The variable ext_dependencies_per_mod_j_lag was added on similar grounds: its inclusion makes for a more refined examination of the growth of external dependencies than measuring external dependencies independently of the number of modules contained in the product. The third and last variable added to the model, used as an indicator of a module's stage of development, is *maturity_ln*. The purpose of using this variable as a predictor in the regression analysis, therefore, is to capture effects associated with different stages of a module's development, elucidating thus group dynamics over a module's life-span. A *log (Ln)* transformation was applied to it upon the assumption that the gravity of its effect is likely to be stronger during the early years of a module's development.[94] Fig. 6.7 illustrates the expanded empirical model:

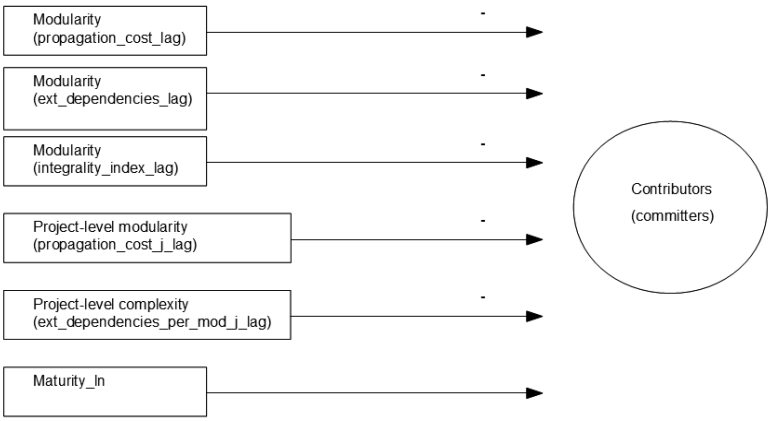


Fig. 6.7: Expanded empirical model H2

Variable	VIF	1/VIF
maturity_ln	2.34	0.426960
ext_dependencies_per_mod_j_lag	1.85	0.539796
propagation_cost_j_lag	1.47	0.680847
propagation_cost_lag	1.25	0.801658
ext_dependencies_lag	1.23	0.815149
integrality_index_lag	1.10	0.913064
Mean VIF	1.54	

Table 6.5: VIF test for regression of committers on propagation_cost_lag, integrality_index_lag, ext_dependencies_lag, propagation_Cost_j_lag, ext_dependencies_per_mod_j_lag, maturity_ln

94 The log transformation of the variable *maturity* is retained throughout the regression analyses presented in this chapter.

Prior to running the regression, we run a VIF test to make sure there is no problem (of multicollinearity caused by) including the six predictors in the same model. Its results, by showing that the VIF values of all variables are smaller than 10, confirm that our predictors are not collinear and can be included in the regression.

Refining the model in this way yields results that indicate strong model significance ($p < 0.001$). Table 6.6 reports the test results:

```
Random-effects GLS regression                   Number of obs      =       242
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.2898                         Obs per group: min =         4
       between = 0.5779                                        avg =       8.3
       overall = 0.5281                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(6)       =    119.07
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
  committers |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
propag~t_lag |  -3.636775   3.833202    -0.95   0.343    -11.14971    3.876163
ext_de~s_lag |   .0977915   .0126436     7.73   0.000     .0730104    .1225726
integrali~ag |  -.2002117   .0871668    -2.30   0.022    -.3710555   -.0293678
propag~j_lag |  -33.46749   6.399284    -5.23   0.000    -46.00986   -20.92513
ext_de~j_lag |   .2268201   .0418048     5.43   0.000     .1448842    .3087559
 maturity_ln |  -815.6295   247.4761    -3.30   0.001    -1300.674   -330.5852
       _cons |   6199.094   1879.782     3.30   0.001     2514.789    9883.399
-------------+----------------------------------------------------------------
     sigma_u |  3.9306219
     sigma_e |  3.3214327
         rho |  .58341369   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.6: Regression results–Effect of Modularity on Contributors

First of all, we observe that the effect of propagation_cost_lag on committers is no longer significant ($\alpha = 0.05$, $\beta = -3.63$, $p = 0.34$), in contradistinction to integrality_index_lag, whose coefficient is negative and significant, suggesting, as theory posits, that lower levels of modularity (as captured by an increase of integrality_index_lag) bring about a decrease of contributors. Ext_dependencies_lag exerts a strong effect on committers too. The coefficient for ext_dependencies_lag is positive and significant, indicating that an increase of external dependencies leads to an increase of committers. As qualified in the context of the previous test, this is accounted for by the shift of development effort from core to peripheral modules with many (out-degree) external dependencies.

As far as environmental factors are concerned, we see that propagation_cost_j_lag (which we use as a proxy for the modularity of the product as a whole) has a strong negative effect on the size of the groups developing individual modules, implying therefore that a decrease of modularity in the broader

FreeBSD production environment (as captured by an increase of propagation_cost_j_lag) has a discouraging effect on contributors, regardless of which module they elect to work on. The complexity of the FreeBSD project considered as a whole (captured through the ext_dependencies_per_mod_j_lag variable) has a strong bearing on committers' choices, indicating that an increase of complexity results in an increase of contributors. At first sight, this proposition is nothing short of absurd: for it implies that not only does increased product complexity (and by implication increased complexity in the product development process) not hinder large-scale collaboration, but that it acts upon it as a catalyst. Yet this result is no longer mystifying once we shun the assumption that the number of external dependencies divided by the number of modules contained in the product is a reliable index of complexity for the entire project. Similarly to the previous statistical test, this test indicates that an increase of a module's (out-degree) external dependencies leads to an increase of its contributors, which result we qualified by introducing the analytical distinction between core and peripheral modules and arguing that over time development effort shifts from core to peripheral modules: peripheral in the sense that they are heavy users of functionality contained in core modules; core because they make limited, if any, use of functionality located in other modules. Hence, the reason why modules with many out-degree external dependencies attract many contributors is because the locus of development shifts to peripheral modules – by definition, those with many out-degree external dependencies – once core modules reach production-readiness (i.e. maturity). Looked at from this perspective, the ratio of external dependencies to modules is an *index of core functionality*: to be precise, an index of the amount of core functionality used by the cumulative number of modules in the product. In the case of FreeBSD in particular, the amount of core functionality contained in the codebase increased rapidly in the first three years of development from 1994 to 1997, at which point it seems to have stabilised as a proportion of the total product, as Fig. 6.8 below illustrates. From 1998 onwards, we see that core functionality increases at a rate similar to the size of the product (proxy-measured here by the number of modules comprising it). That is to say, its growth mirrors the increase in the size of the product as a whole. Apparently, the fraction of the product represented by core functionality has since remained at the same level because the minimum amount of core functionality required for an operating system like FreeBSD was developed in the first three years of the project. Examining the growth of external dependencies relative to other measures of product size than the

number of modules it contains, reinforces this syllogism.[95] As Fig. 6.9 and Fig. 6.10 illustrate, the ratio of external dependencies to MB[96] and KLOC[97] has remained stable after the first three years of development.

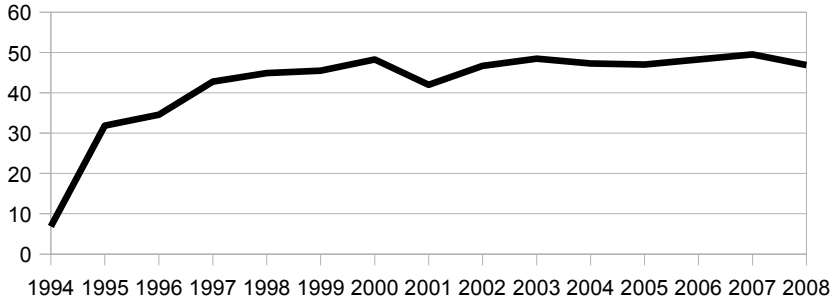


Fig. 6.8: Core functionality (external_dependencies_per_module)

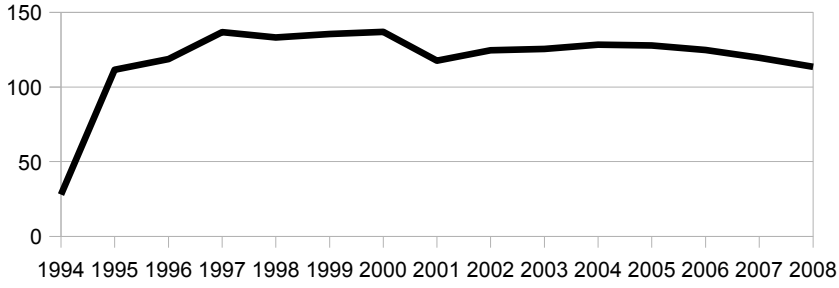


Fig. 6.9: Core functionality (external_dependencies_per_MB)

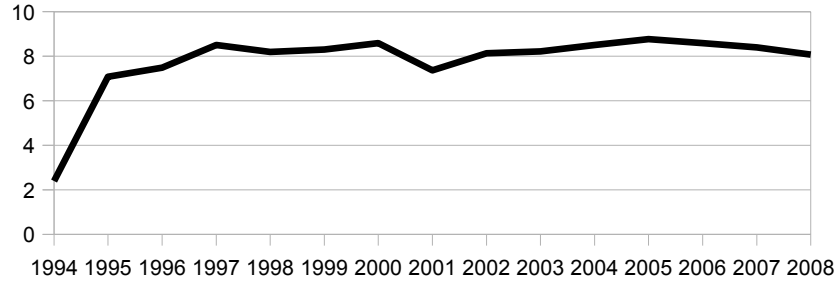


Fig. 6.10: Core functionality (external_dependencies_per_KLOC)

95 This pattern of design evolution is not limited to FreeBSD: a recent empirical study by MacCormack et al. (2010) identified the same pattern in examining the evolution of the size of the core in Linux.

96 1 MB = 1024 KB.

97 1 KLOC = 1000 LOC.

To return to the statistical analysis, an explanation therefore for the positive effect of ext_dependencies_per_mod_j_lag on committers is that an increase of core functionality reflects an expansion of project scale. The reported effect then is not so much accounted for by the growth of dependencies as by the enlargement of project scale, which operates as an attractor of potential contributors. Especially during a project's early stage of development, increasing scale signals to potential contributors that the project is gathering momentum. Nobody wants to contribute to a project that may languish, in fear that this would diminish the perceived value of their contributions or worse still, that the time and programming effort they contributed be rendered useless. From this point of view, core functionality is an index of the utility of the evolving product.

Interestingly enough, maturity_ln has a strong negative effect on group size, indicating a tendency for modules to be developed by increasingly smaller groups over time. That is to say, modules attract more contributors in their early stage of development. How is this explained, considering that a module with a large group of contributors is likely to attract even more contributors? Surely, the influence that large groups exert over potential contributors in choosing on which part of the project to focus should not be underestimated, for potential contributors are more likely to gravitate toward modules developed by large groups than small ones. Is this argument contradicted by our results? We think not. Apparently what accounts for the effect of time on group size is modules' level of production-readiness (i.e. maturity). During a module's early stage of development, the number of production tasks available for potential developers to tackle is much greater than in later development stages, thus signalling to potential developers that their contribution at this point in the module's life-cycle shall be in some way indispensable. As in that phase one's contribution is considered to have a perceptible effect on the development of the module, contributing to a module's early development increases in attractiveness. Conversely, when a module approaches production-readiness and the number of production tasks pending completion is dramatically reduced, fewer contributors are needed. At that point most of the contributors hitherto engaged in the development of the module will be drawn to other parts of the project, leaving behind a committer (or a small team of committers) to serve henceforth as the *maintainer* of the module. In consequence, the number of contributors to a module falls over its development life-cycle, in inverse proportion to the module's level of maturity. Relative to mature modules, more contributors are attracted to modules

in an embryonic stage, for that phase signifies a potential for growth: it is early in a module's life-span that one's contribution is perceived to have a lasting and indispensable effect.

Concluding, a final attempt was made to refine the regression model even further by excluding variables with no significant effect. Thus, propagation_cost_lag was removed from the regression.

```
Random-effects GLS regression                   Number of obs      =       242
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.2869                         Obs per group: min =         4
       between = 0.5774                                        avg =       8.3
       overall = 0.5315                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(5)       =    118.97
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
  committers |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
ext_de~s_lag |   .0940385   .0118353     7.95   0.000     .0708417    .1172352
integrali~ag |  -.2164368   .0853014    -2.54   0.011    -.3836245   -.0492492
propag~j_lag |  -34.13047   6.370398    -5.36   0.000    -46.61622   -21.64472
ext_de~j_lag |   .2265167   .0418768     5.41   0.000     .1444396    .3085938
 maturity_ln |  -881.7319   238.7863    -3.69   0.000    -1349.744   -413.7195
       _cons |   6700.813   1813.885     3.69   0.000     3145.664    10255.96
-------------+----------------------------------------------------------------
     sigma_u |  3.8559278
     sigma_e |   3.320212
         rho |  .57423863   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.7: Regression results–Effect of Modularity on Contributors

As shown in Table 6.7 above, though removing it does not affect the model's significance or explanatory power ($\Delta R^2 = 0.53$, $p < 0.001$), the results now show a perceptibly stronger effect of integrality_index_lag on committers. As the coefficient for integrality_index_lag is negative and significant, it reinforces the conclusion drawn from the previous test that a decrease in modularity results in decreasing group size.

A SUMMING UP

To sum up the results of our statistical tests: *higher levels of modularity result in larger groups*. In greater detail, contributors to a module increase when (a) that module's modularity increases and (b) the modularity of the broader production environment increases (i.e. the complexity of the broader production environment decreases). In addition, we found that (c) modules attract more contributors when the core functionality contained in the product increases and that (d) an increase of

a module's (out-degree) external dependencies results in an increase of contributors to that module. Conversely, contributors to a module decrease when (a) that module's integrality increases and (b) the complexity of the broader production environment increases. Furthermore, we found that (c) contributors to a module decrease over time, as modules attract more contributors in their early development stage. Hence, the size of the group that develops a module is inversely proportional to the module's level of maturity: the more mature a module the smaller the size of the group developing it. In view of these results, *H2 is verified*.

Before we proceed to test *H2R* and *H3*, drawing upon the results of testing *H2* allows us to evaluate the metrics we used in the analysis: of all the metrics used to assess individual modules' degree of modularity, integrality index appears to be the most robust. What propagation cost actually reflects at the component-level is not so much modularity as complexity: assessing the cascading effect of product changes ramifying through a chain of dependencies is not the same as assessing the extent that dependencies have been localised within modules.[98] Last, (out-degree) external dependencies, though unfit for the purpose of assessing modularity, can be used as an index of the locus of development activity. To verify the soundness of what has been conjectured about the metrics used in the analysis, we contrasted the effect of lapse of time on propagation cost with its effect on integrality index. First, we carried out a regression analysis with *propagation cost* as dependent variable and *maturity_ln* as independent variable, which indicated strong model significance ($p < 0.050$):

```
Random-effects GLS regression                   Number of obs      =       280
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0315                         Obs per group: min =         5
       between = 0.0437                                        avg =       9.7
       overall = 0.0010                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      6.98
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0082

------------------------------------------------------------------------------
propagatio~t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
 maturity_ln |   10.27161   3.886545     2.64   0.008     2.654127     17.8891
       _cons |  -77.71154   29.54615    -2.63   0.009    -135.6209   -19.80215
-------------+----------------------------------------------------------------
     sigma_u |  .10005983
     sigma_e |  .10268773
         rho |  .48704077   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.8: Regression results–Effect of lapse of time on Propagation Cost

98 Put another way, the propagation cost reflects the need for coordination among source code files, rather than among modules (i.e. clusters of files) which is the object of inquiry proper.

The positive coefficient for maturity_ln suggests that propagation cost rises over time, that is to say individual components' levels of modularity manifest a declining tendency. This declining tendency is also attested in the project-level descriptive statistics we discussed in the aforementioned qualitative analysis: as we have seen in Fig. 6.3, the propagation cost of the codebase as a whole increases as the development process unfolds. Assuming that propagation cost is a valid indicator of modularity, its tendency to rise can only be interpreted as to mean that with the passage of time FreeBSD is characterised by increasingly lower levels of modularity.

Subsequently to testing the effect of lapse of time on propagation cost, we performed another regression with integrality index as dependent variable and maturity_ln again as independent variable. The test indicated strong model significance ($p < 0.001$):

```
Random-effects GLS regression                   Number of obs      =       271
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0970                         Obs per group: min =         5
       between = 0.0015                                        avg =       9.3
       overall = 0.0236                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(1)       =     25.71
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
integralit~x |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
 maturity_ln |  -470.0172   92.69344    -5.07   0.000     -651.693   -288.3414
       _cons |   3577.526     704.68     5.08   0.000     2196.378    4958.673
-------------+----------------------------------------------------------------
     sigma_u |  5.0776062
     sigma_e |  2.3418861
         rho |  .82459093   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.9: Regression results–Effect of lapse of time on Integrality Index

More importantly, the coefficient for maturity_ln is now negative, suggesting thus that integrality index falls over time, that is to say individual components' degree of modularity manifests an increasing tendency.

How can one reconcile the conflicting results of the last two tests? If it is assumed that propagation cost and integrality index are both robust component-level (i.e. module-level) proxies for modularity, one ends up with logically inconsistent results: basing the analysis on propagation cost leads to the conclusion that individual modules' levels of modularity deteriorate; by contrast, one draws the conclusion that individual modules' levels of modularity improve when using integrality index as a proxy. Given that a module's degree of modularity cannot be rising and falling at the same time, the inconsistency in the results can only mean that one of the two proxies is problematic – that is, either propagation cost or

integrality index does not capture modularity. As we explained earlier, propagation cost assesses the cascading effect of changes propagating through a chain of either direct or indirect dependencies between files, while integrality index reflects the extent that dependencies have been successfully encapsulated within modules. Insofar as the encapsulation of interactions (i.e. dependencies) is considered a more reliable criterion of the efficacy of the modularisation process at the component-level (Parnas 1972; Sharman & Yassine 2004; Simon 1962; Wheeler 2007), it makes more sense to assess a module's degree of modularity through its integrality index than through its propagation cost. On the other hand, calculating the propagation cost at the module-level seems better suited to the task of assessing the impact that changing a file in a module exerts on other files within that module (on account of interdependencies between the focal point of the change and other files contained in the module). Viewed in this way, the propagation cost of a module is an index more akin to its internal complexity than its modularity. In addition, the results are no longer contradictory when interpreted in this light: in fact, it is extremely likely that a decrease of a module's integrality index is accompanied by an increase of its internal complexity. It follows from the operational logic of the *clustering process*[99] that the increase of a module's internal complexity is a concomitant of its encapsulation of interactions: hence, a module's internal complexity increases in proportion to its encapsulation of interactions.

Thanks to this clarification of metrics, we can revisit the results of our statistical tests and further elucidate them. Considering therefore that a module's internal complexity and modularity is reflected in its propagation cost and integrality index respectively, our quantitative analysis indicates that modules evolve toward higher levels of modularity and internal complexity in the course of their development. With the passage of time modules become progressively less dependent on each other, hence more modular. In parallel, as the interdependence between files within modules rises over time, modules become also more internally complex.

REVERSING THE TERMS OF THE PROPOSITION

While *H2* holds that higher levels of modularity result in larger groups, in *H2R* the terms of the proposition are reversed so that the claimed direction of causality is

99 *Clustering* is the process by which most, if not all, of the interactions (i.e. dependencies) are localised within clusters of system elements (i.e. modules) and the interactions or links between separate clusters are eliminated or minimised (Sharman & Yassine 2004, p. 40).

from group dynamics to product structure:

> *An increase of contributors to FreeBSD results in an increase of modularity* (H2R)

Empirical support for the hypothesis that the pattern of interactions between the developers participating in a software project shapes the resulting product's dependency relations comes from the work of MacCormack et al. (2008a), who compared five matched product pairs (of similar size and functionality) developed through different modes of organisation. Importantly, where the product was developed by a large distributed group of the type exemplified by large FOSS projects like Linux, as opposed to a small and co-located group employed by a single firm, the resulting product was markedly more modular, thereby suggesting that 'a product's architecture tends to mirror the structure of the organization within which it is developed' (MacCormack et al. 2008a, p. 20). The crystallisation of the social relations of production into the software artefact, MacCormack et al. argued, is owed to the fact that,

> In closed source projects, dedicated teams employed by a single firm and located at a single site develop the design. Problems are solved by face-to-face interaction, and performance “tweaked” by taking advantage of the access that module developers have to the information and solutions developed in other modules. Even if not an explicit managerial choice, the design naturally becomes more tightly-coupled. By contrast, in open source products, a large and widely distributed team develops the design. Face-to-face communications are rare given most developers never meet, hence fewer connections between the modules are established. The architecture that evolves is more modular as a result of the inherent limitations on communication (MacCormack et al. 2008a, p. 21; also, see MacCormack et al. 2006, p. 1027).

The main thrust of this argument is not foreign to software developers. Better known as *Conway's Law*, it was originally formulated in 1968 by Melvin Conway,

who contended that 'organizations which design systems...are constrained to produce designs which are copies of the communication structures of these organizations'. This has been reported repeatedly. For example, a 1988 study of the development of seventeen large software systems observed that 'the social structure of the project was occasionally factored into architectural decisions...the partitioning [of the product architecture] was based not only on the logical connectivity among components, but also on the social connectivity among the staff' (Curtis et al. 1988, p. 1280). Such a *mirroring effect* was more recently attested in two empirical studies of FOSS development by Capra et al. (2008) and Merlo et al. (2009) based on seventy-five and thirty-seven FOSS (Java) projects respectively. Both studies qualified this result by arguing that modularity is a consequence of the (decentralised, informal and open) governance structure of large FOSS projects. It follows from the predominantly volunteer character of participation in FOSS projects that contributors are not subject to the pressure of deadlines that apply to commercial software development settings. As a result of removing the pressure of deadlines from the development process, developers are given a motive to write clean, elegant code, developing thus software of higher design quality (and therefore more modular) than they would were they working at a commercial software firm. In addition, as programming practice in FOSS projects is essentially a 'public process' founded on the openness of source code, contributors to FOSS projects take for granted that their code shall be exposed to public scrutiny. This serves as an extremely effective mechanism to spur contributors on to producing high quality code. As Capra et al. (2008, p. 778) point out:

> When code is open, all team members take personal pride in writing clean and understandable pieces of code, in polishing the design of their artifacts, and in commenting their work since they feel exposed to the judgement of the whole community of developers and, consequently, pay particular attention to [design] quality. When development is voluntary, with no pressure from managers or customers, time can be more easily allocated to improving the design of the code.

In order to test *H2R* at the module-level, we performed a regression analysis

using the same dataset as before[100] with *integrality index* as dependent variable and the number of *committers* as independent variable. As in testing *H2*, the intraclass correlation between observations pertaining to the same module was taken into account by using the group variable *module* in the regression analysis. Fig. 6.11 illustrates the empirical model:

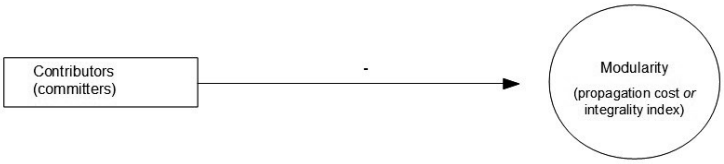


Fig. 6.11: Empirical model H2R

However, the test indicated no model significance (p = *ns*):[101]

```
Random-effects GLS regression                   Number of obs      =       271
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0017                         Obs per group: min =         5
       between = 0.1159                                        avg =       9.3
       overall = 0.0744                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      1.06
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.3032

------------------------------------------------------------------------------
integralit~x |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |  -.0361363   .0350957    -1.03   0.303    -.1049226       .03265
       _cons |   4.651601   .9525754     4.88   0.000     2.784587    6.518614
-------------+----------------------------------------------------------------
     sigma_u |  4.7627606
     sigma_e |  2.4623781
         rho |  .78908183   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.10: Regression results–Effect of Contributors on Modularity

In examining other alternatives, we resorted to testing the model with propagation cost as dependent variable, taking into account however that it is a less robust component-level (module-level) indicator of modularity than integrality index. But neither did this test indicate model significance (p = *ns*).[102]

100 The panel dataset consists of a stratified sample of twenty-nine FreeBSD modules with observations spanning fourteen years of development activity from 1994 to 2008. See section **Sample selection** in chapter 3 for a discussion of the sample selection procedure.

101 To explore the time-structure of processes, we also tested the model with the predictor transformed (i.e. lagged a year) but the test indicated no model significance and found no significant effect of *committers_lag* ($\alpha = 0.05$, $\beta = -0.03$, $p = 0.36$).

102 In experimenting with the regression model, we also tested it with the predictor transformed (i.e.

As none of the tests indicates a significant effect of committers on propagation cost or integrality index, we find so far no empirical support for the hypothesis (*H2R*) that an increase of participants in a distributed software development process leads to higher levels of modularity.

Scale considerations

In order to verify the absence of a perceptible effect of group size on product structure as well as to make sure that the effect of scale has not been overlooked, we attempted to refine the analysis by distinguishing between conditions of large-scale and small-scale development based on the median of committers. That being eight, a small-scale development process is reflected in years that fewer than nine committers participate in the development of a module (i.e. committers < 9), while large-scale development is reflected in years that committers exceed eight (i.e. committers > 8). Distinguishing thus large-scale from small-scale development conditions, to examine the effect of group size on modularity in a large-scale development process, we carried out a regression analysis with *integrality index* as dependent variable and *committers* as independent variable, *excluding years in which committers are fewer than nine*. Contrary to the previous two tests, this one indicated strong model significance ($p < 0.050$):

```
Random-effects GLS regression                   Number of obs      =       123
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0219                         Obs per group: min =         1
       between = 0.1508                                        avg =       5.1
       overall = 0.2228                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      5.01
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0252

------------------------------------------------------------------------------
integralit~x |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |  -.0607349   .0271312    -2.24   0.025    -.1139112   -.0075587
       _cons |   3.890664   .5073959     7.67   0.000     2.896186    4.885142
-------------+----------------------------------------------------------------
     sigma_u |  1.6097449
     sigma_e |  1.3853117
         rho |   .5745161   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.11: Regression results–Effect of Contributors on Modularity in Large-scale conditions (Condition: if committers > 8)

As Table 6.11 shows, the coefficient for committers is significant and negative,

lagged a year), but again the test indicated no model significance and found no significant effect of *committers_lag* ($\alpha = 0.05$, $\beta = 0.00$, $p = 0.25$).

suggesting that an increase of committers leads to a decrease of integrality index. Hence, increasing group size in conditions of large-scale development results in higher levels of modularity.

Subsequently to testing the effect of group size on modularity in large-scale development conditions, we proceeded to test this relationship in conditions characteristic of small-scale development. To do this, we repeated the above regression analysis (with *integrality index* as dependent variable and *committers* as independent variable), but we now *excluded years in which committers exceed eight*. However, the test indicated no model significance (p = *ns*):

```
Random-effects GLS regression                   Number of obs      =       148
Group variable: module                          Number of groups   =        28

R-sq:  within  = 0.0026                         Obs per group: min =         1
       between = 0.0244                                        avg =       5.3
       overall = 0.0153                                        max =        11

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      0.12
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.7248

------------------------------------------------------------------------------
integralit~x |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |   .0553826   .1572942     0.35   0.725    -.2529082    .3636735
       _cons |   4.339793   1.284706     3.38   0.001     1.821816     6.85777
-------------+----------------------------------------------------------------
     sigma_u |  4.9558817
     sigma_e |  3.1572585
         rho |  .71130735   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.12: Regression results–Effect of Contributors on Modularity in Small-scale conditions (Condition: if committers < 9)

In contrast to the test focusing on large-scale development conditions (*Table 6.11*) which suggests that an increase of committers leads to higher levels of modularity, the test centred on small-scale development conditions (*Table 6.12*) found no significant effect. Hence, considered together, they offer empirical support for the hypothesis (*H2R*) that increasing group size leads to higher levels of modularity upon the condition that large-scale development conditions apply.

As we ascertained when testing *H2*, individual modules evolve toward higher levels of modularity and internal complexity in the course of their development, which result we qualified by pointing out that a module's internal complexity rises as a result of its encapsulation of dependencies. We are now in position to conduct an additional test of robustness for this finding from a different angle. We have already seen that increasing group size leads to higher levels of modularity in large-scale development conditions. How does this relate to the effect on modules'

internal complexity? Is the internal complexity of a module rising in parallel with its modularity, as would be expected from the conclusions we have drawn so far? To find out, we looked at the effect of group size on complexity in conditions of large-scale development by conducting a regression analysis with *propagation cost* as dependent variable and *committers* as independent variable, *excluding years in which committers are fewer than nine*. The test indicated strong model significance ($p < 0.005$):

```
Random-effects GLS regression                   Number of obs      =       123
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0921                         Obs per group: min =         1
       between = 0.0079                                        avg =       5.1
       overall = 0.0285                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      8.92
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0028

------------------------------------------------------------------------------
propagatio~t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |   .0038438   .0012867     2.99   0.003     .0013219    .0063658
       _cons |   .3148119   .0231866    13.58   0.000      .269367    .3602568
-------------+----------------------------------------------------------------
     sigma_u |  .06936431
     sigma_e |  .06733187
         rho |  .51486497   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 6.13: Regression results–Effect of Contributors on Complexity in Large-scale conditions (Condition: if committers > 8)

Specifically, the coefficient for committers is significant and positive, suggesting that an increase of group size in a large-scale development process leads to an increase of propagation cost, that is, an increase of complexity. Therefore, insofar as large-scale development conditions apply, increasing group size brings about an increase of modularity (*Table 6.11*) and internal complexity (*Table 6.13*), reinforcing thus the foregoing interpretation of results.

To recap, our statistical tests (*Tables 6.11, 6.13*) show that an increase of group size in a large-scale development process results in higher levels of modularity and internal complexity alike. Consequently, *we find empirical support for H2R that a software product becomes more modular when the number of developers participating in its development process increases upon the condition that large-scale development conditions apply.* Furthermore, consistent with the foregoing analysis, this finding reinforces the conclusion that at the component level (i.e. module-level) an increase of modularity is accompanied by an increase of internal complexity. By contrast, we find no support for *H2R* in conditions characteristic of small-scale development. The absence of a perceptible effect of increasing group

size on product design in small-scale development settings hints that adding more developers to a project – when the overall development group remains small – does not necessitate a radical modification of work patterns. Provided that the group remains small, developers' work process is not significantly affected by an increase of developers working on the project. Insofar as no modification of the existing communication system is required, an increase in the number of developers working on a project is rather unlikely to leave a mark on the architectural structure of the product. In order for an increase of group size to crystallise into the software artefact, the increase must be such that it renders necessary a (radical) modification of communication patterns and, by extension, work patterns.

CONCLUDING REMARKS

The statistical analyses presented in this chapter provide strong empirical support for the hypothesis (*H2*) that modularity increases the potential number of contributors as well as for hypothesis *H2R* which reverses the directionality of the claimed effect so that increasing group size results in an increase of modularity. Thus, *H2 and H2R are verified*.

A concern raised by these results is whether the regression models used to test *H2* and *H2R* reveal correlations instead of causal processes. To delve more deeply into the modularity-committers relation, we tried to test for forms of causality by doing a standard bivariate Granger-causality test on committers and integrality index.

```
      Source |       SS       df       MS              Number of obs =     242
-------------+------------------------------           F(  2,   239) =   15.75
       Model |  1504.12714     2  752.063572            Prob > F      =  0.0000
    Residual |  11411.8067   239  47.7481454           R-squared     =  0.1165
-------------+------------------------------           Adj R-squared =  0.1091
       Total |  12915.9339   241  53.5930867           Root MSE      =    6.91

------------------------------------------------------------------------------
  committers |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
committer~ag |   .1701838   .0599941     2.84   0.005      .051999    .2883686
integrali~ag |   -.449088   .0930598    -4.83   0.000    -.6324103   -.2657658
       _cons |   10.35411   .8169576    12.67   0.000      8.74475    11.96346
------------------------------------------------------------------------------

( 1)  integrality_index_lag = 0

       F(  1,   239) =   23.29
            Prob > F =    0.0000
```

Table 6.14: Granger causality test

The notion of Granger causality (Granger 1969) is simple: If lagged values of *X* predict current values of *Y* in a forecast formed from lagged values of both *X* and *Y*, then *X* is said to Granger-cause *Y*. We implemented the test by regressing committers on lagged committers (committers_lag) and lagged integrality index (integrality_index_lag). As Table 6.14 above shows, the test found statistically significant causality. In specific, we see that the coefficient on lagged integrality index is significant, which suggests that integrality index causes committers. Following this step, we used a symmetric regression to test the reverse causality:

```
      Source |       SS       df       MS              Number of obs =     242
-------------+------------------------------           F(  2,   239) =  608.59
       Model |  4309.55105     2  2154.77552            Prob > F      =  0.0000
    Residual |  846.207936   239  3.54061898            R-squared     =  0.8359
-------------+------------------------------           Adj R-squared =  0.8345
       Total |  5155.75898   241  21.3931908            Root MSE      =  1.8817

------------------------------------------------------------------------------
integralit~x |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
integrali~ag |   .8833187    .025341    34.86   0.000     .8333985    .9332389
committer~ag |  -.0204813   .0163369    -1.25   0.211     -.052664    .0117014
       _cons |   .4898839   .2224646     2.20   0.029     .0516421    .9281256
------------------------------------------------------------------------------

( 1)  committers_lag = 0

       F(  1,   239) =    1.57
            Prob > F =    0.2112
```

Table 6.15: Granger causality test

Here we see that the resulting F-statistic is *1.57* with significance level *0.2112*, indicating thus that committers do *not* Granger-cause integrality index. It is important to note that these results do not necessarily imply that committers have no effect on integrality index. Rather, the interpretation we give to these results is based on the 'temporal ordering interpretation of Granger causality' (Thurman & Fisher 1988), by which we mean that our purpose is to provide an empirical answer to the question *which comes first, modularity or committers?* In this case, based on the results of our Granger test, we conclude that the effect of modularity on committers comes first, preceding the effect that committers exert on the product structure.

The full implications of these results are explored in *Chapter 10: Conclusions,* where a synthesis of the research findings is attempted. In the next chapter, our inquiry turns to the third hypothesis which holds that product modularity has a positive effect on labour productivity.

CHAPTER 7: MODULARITY AND LABOUR PRODUCTIVITY IN FREEBSD

SETTING OF THE PROBLEM

The study of the effect of increasing the number of persons working collectively on group performance has a long history in the social sciences. The branch of social psychology has long underlined the demotivating effects of increasing group size on individual performance (e.g. Ingham et al. 1974; Latané et al. 1979), while economists concerning themselves with the boundaries of the firm, that is, the extent of the division of labour within the firm, have drawn attention to coordination costs as being responsible for decreasing returns to scale (e.g. Coase 1937; Kaldor 1934; Robinson 1934; Walker 1866; Williamson 1967, 1975, 1985). In the realm of software engineering, the negative effect of increasing the number of programmers working on a project on group productivity is known as *Brooks' Law*, after Fred Brooks, project manager for the development of the OS 360 operating system at IBM. Confronted with a project running late, Brooks attempted to step up its development process by assigning more programmers to work on the project. This decision, however, only exacerbated the problem, as adding more programmers occasioned a further decrease of productivity. Brooks (1995) pinpointed the problem in the communication and coordination costs attendant on increasing the scale of the project. According to his diagnosis, pushing the division of tasks beyond a certain point will undoubtedly decrease productivity through the overhead costs it entails: as more programmers are added, the costs of communication and coordination within the group grow exponentially, negatively impacting performance, an effect which empirical studies of software development have since confirmed time and again (Blackburn & Scudder 1996; BlackBurn et al. 2006; Boehm 1981).

The design principle of modularity has been proposed as a solution to this problem. As described by Boehm (1981, p. 194), product modularity is 'one very powerful technique...to reduce diseconomies of scale by reducing scale'. In specific, the reduction of scale is accomplished by breaking down the product into components (i.e. modules) that can be developed independently of one another without undercutting the functionality of the product as a whole. By implication,

developers can work on different components of the product without concerning themselves with what others are doing in the project. As long as they do not need to communicate extensively with developers concentrating on other modules, an increase of group size is no longer subject to the exponential growth of communication and coordination costs that upset Brooks' plans.

This hypothesis is clearly formulated in Narduzo and Rossi's (2005) discussion of the role of modularity in free and open source software (FOSS) development:

> A large number of participants in a project may be not a sufficient condition to generate dysfunctional effects, such as diminishing or negative marginal return of manpower to productivity. The key aspect in this regard is represented by the degree of task interdependency between the various members belonging to the project. Thus, the high productivity...is largely due to the massively modularized structure of the project, enabling the existence of highly independent sub-projects joined by a limited number of developers.

By allowing for modules to be developed independently by autonomous groups of developers, product modularity mitigates the adverse effects of increasing scale, invalidating thus Brooks' Law. As Osterloh and Rota (2007, p. 166) write, 'with a non-modular architecture, having more people involved in a project means higher coordination costs that can in the extreme case, render marginal returns of manpower to productivity negative'. With a modular architecture, on the contrary, 'the costs of the production of the source code are also kept low. A modular architecture invalidates "Brooks' Law" that "adding manpower to a late software project makes it later"' (Osterloh & Rota 2007, p. 166). Concisely, the organisational benefits that this stream of the literature attributes to product modularity are summed up in the following proposition by Boehm (1981):

Product modularity reduces diseconomies of scale

As the mitigation of the adverse effects of increasing scale implies a positive effect on productivity, the proposition can be alternatively stated as:

Product modularity has a positive effect on labour productivity in projects characterised by increasing scale

Given that the FreeBSD project is characterised by an increase of scale over time, the proposition can be reformulated as a hypothesis for empirical testing in FreeBSD:

Product modularity has a positive effect on labour productivity in FreeBSD (H3)

Fig. 7.1 illustrates the hypothesised effect in the broader context of the research model that encapsulates the hypotheses derived from the literature review:

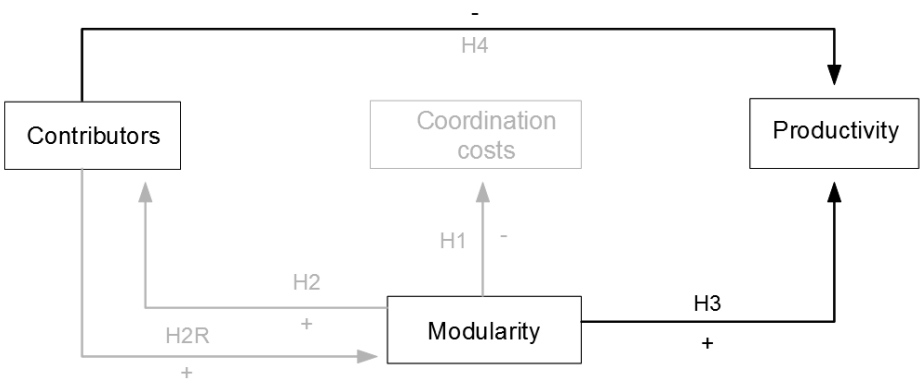


Fig. 7.1: Research model

Despite the fact that this hypothesis is central to the work of such recognised authorities on the economics of software production as Boehm, the empirical confirmation of the claimed benefit is still wanting, as we remarked in our review of the modularity literature in chapter 2. To address the dearth of empirical data on the productivity gains of modularity, increasingly more investigations turn to the analysis of software repositories (e.g. version control systems like CVS or archived mailing lists) for data conducive for quantitative study. But unfortunately, we have not come across a single empirical study that engages with the problem rigorously

enough;[103] as a result, analysis remains inconclusive. Against this background, we analysed fourteen years of development activity as archived in FreeBSD's software repositories with a view to testing this hypothesis.

QUALITATIVE ANALYSIS

Our analysis of descriptive statistics begins with project scale. As Fig. 7.2 illustrates, the scale of the project as reflected in the number of developers checking-in code to the project repository has expanded dramatically over time: committers increased tenfold from 16 committers in 1994 to 198 in 2007.

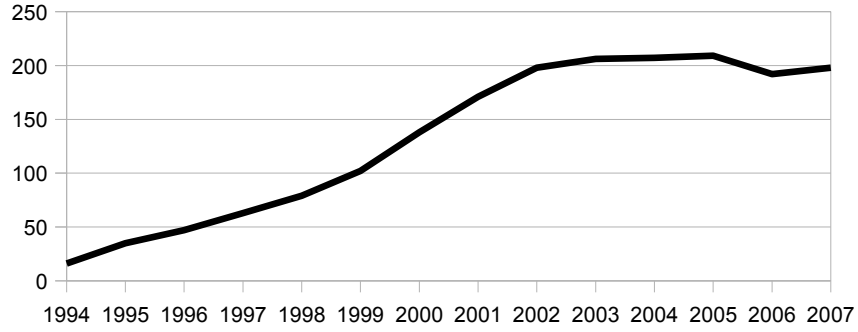


Fig. 7.2: Committers (*src*)

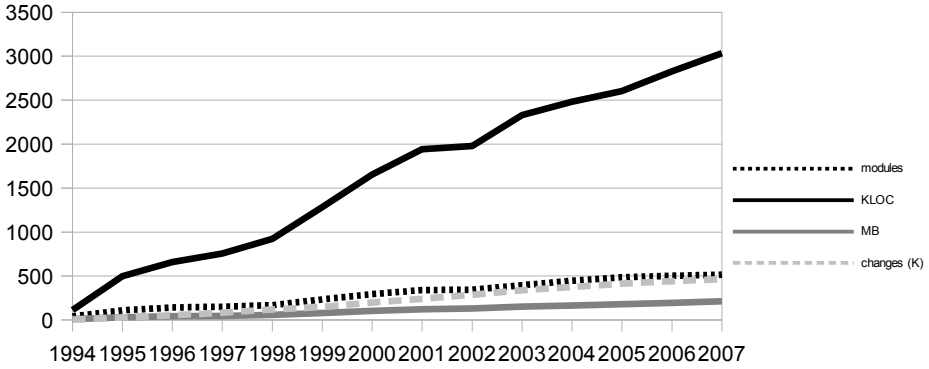


Fig. 7.3: Product evolution (Notes: 1 MB = 1024 KB; 1 KLOC = 1000 LOC)

103 Indicatively, see our criticism of the FOSS studies by Giuri et al. (2008) and Schweik et al. (2008) in chapter 2.

The dramatic enlargement of project scale can be seen not only in the expansion of the base of committers but also in the size of the codebase. As Fig. 7.3 above shows, in the space of thirteen years from 1994 to 2007 the product underwent 463309 changes, as a result of which it grew by 218316 KB, 3034654 source lines of code (LOC) and 515 modules. The sheer number of changes made to the product is indicative of the magnitude of change it undergoes: averaging 33093 changes per year, FreeBSD grows by an average of 22.79% in KB, 39.38% in LOC and 26% in modules per annum.

Having established the remarkable expansion of project scale over time, our inquiry can now turn to the analysis of productivity. Let us look at the volume of code contributions checked into the codebase (i.e. the modifications made to the product) over time in Fig. 7.4:

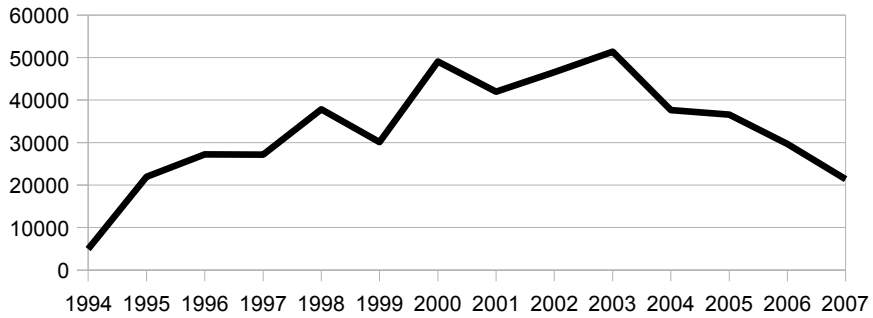


Fig. 7.4: Code contributions

We see that starting in 1994 increasingly more code contributions are checked into the repository per year, peaking at 51384 in 2003. Thereafter contributions manifest a declining tendency, falling down to 21383 in 2007, which, compared to 1994, represents an increase of 436% but a decrease of 59% when compared to 2003.

Interestingly, we do not find such a sharp reduction of production output when looking at the KBs added to the codebase over time (*Fig. 7.5 below*). We observe that from 2004 onwards the cumulative size of new code contributions (in KB) increases steadily, despite the simultaneous fall in the number of code contributions. Substituting LOC for KB leads to similar conclusions (*Fig. 7.6 below*). We see that the LOC added to the codebase after 2005 increase, despite the concurrent decrease of code contributions.

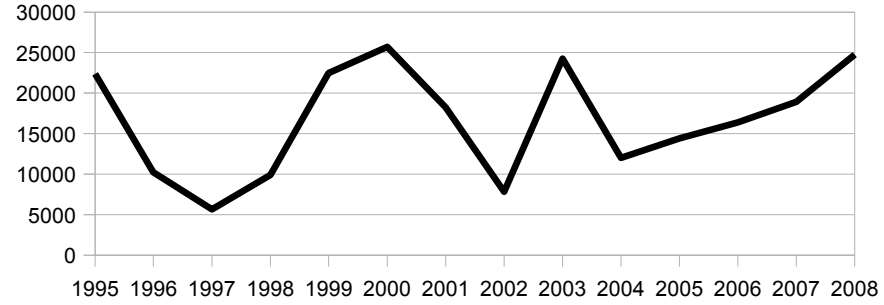


Fig. 7.5: KB added to the codebase

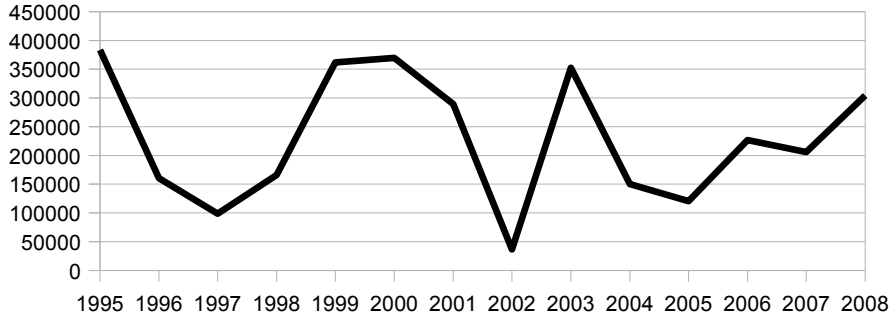


Fig. 7.6: LOC added to the codebase

That after 2004 the KB added to the codebase increase steadily while simultaneously the code contributions decrease can be explained by that code contributions grow bigger in size during this period. This is confirmed by examining the proportion of KB per code contribution. Fig. 7.7 shows that the average size of code contributions (measured in KB) increases after 2004 by 178%.

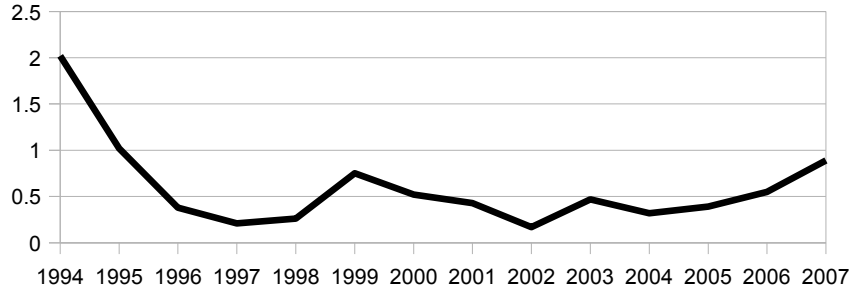


Fig. 7.7: KB per code contribution

Again, substituting LOC for KB leads to the same conclusion. We see that the LOC added to the codebase per code contribution increase after 2004 (by 141.3%), confirming thus that the average size of code contributions increases in this period, as Fig. 7.8 illustrates.

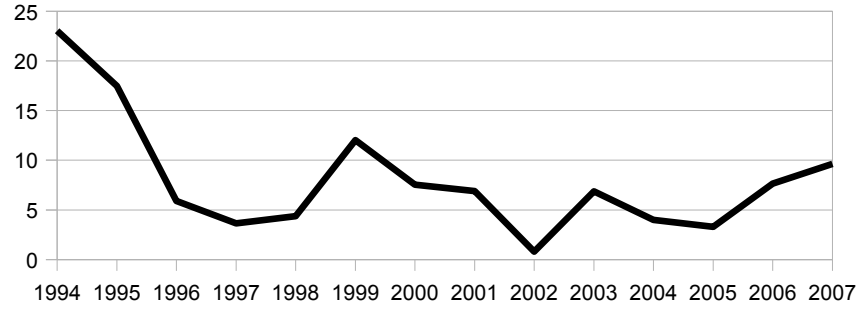


Fig. 7.8: LOC per code contribution

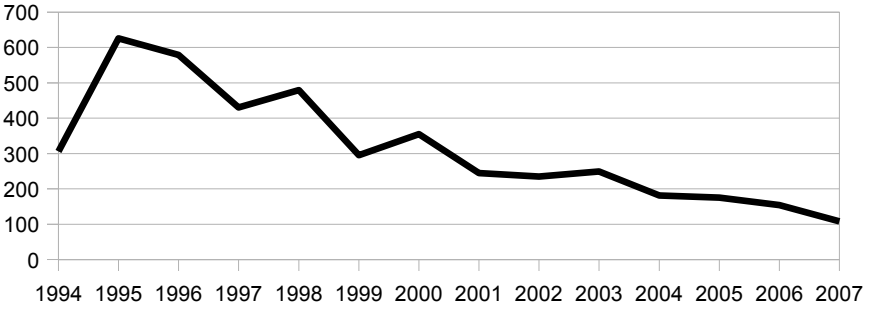


Fig. 7.9: Code contributions per committer

Obviously, an analysis of productivity in absolute terms through the prism of the number and size (measured in KB and LOC) of code contributions independently of the number of committers producing them is incomplete, for it does not account for the effect of the expansion of the base of committers, which is far from irrelevant. As what interests us is the returns to scale exhibited by the production process, we need to look at average productivity as scale increases (Banker 1984; Banker & Slaughter 1997; Banker et al. 1994; Robinson 1934). That is why we looked at the number of code contributions per committer as an indicator of average productivity in the project (*Fig. 7.9 above*). By examining the average number of code contributions per committer over time, we see that productivity

falls relative to the number of contributors: that is, we observe a steady decrease of average productivity.

On the other hand, examining productivity through the size of code contributions (in KB) relative to the number of committers (*Fig. 7.10 below*) shows a steep fall from 1995 until 1997, but from 1997 onwards the KB added to the codebase are proportionate to the number of committers, tending to plateau around an average of 100 KB per committer. Oddly, when looking at average productivity through the prism of the KB added per committer, we do not detect such a gradual productivity fall as when using the number of code contributions per committer as a proxy. By contrast, we observe that average productivity falls abruptly from 1995 to 1997 and fluctuates since. Equally important, its fluctuations, on account of their declining intensity, are flattening out over time.

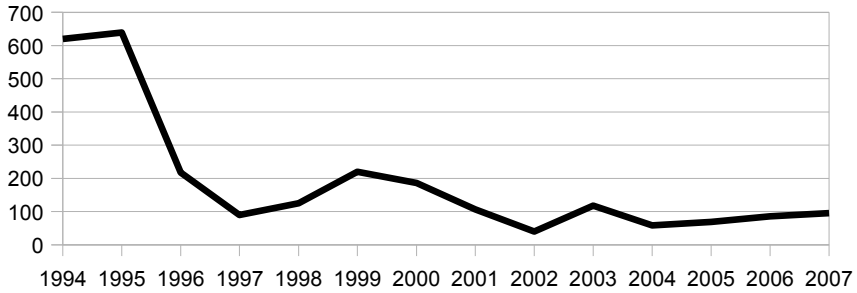


Fig. 7.10: KB added per committer

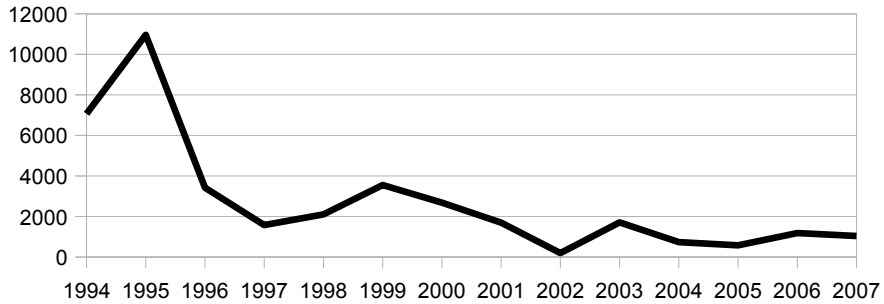


Fig. 7.11: LOC added per committer

Checking for latent inconsistencies by substituting LOC for KB leads to similar conclusions (*Fig. 7.11 above*). We see that, following an initial sharp fall, from 1997 onwards the average number of LOC added per committer fluctuates, yet this

fluctuation nowhere resembles the steady decrease we observed when examining (in Fig. 7.9) the ratio of code contributions to committers over time.

Accounting for this discrepancy is that from 2003 onwards increasingly more development activity in the FreeBSD project migrated from CVS to Perforce and later on to the Subversion revision control environment because of the those environments' superior support for parallel development. By 2006 Perforce had replaced CVS as the development site of experimental features, while the Subversion server is where development work on the *src* tree is currently taking place. However, CVS has not been abandoned as a result of this migration of development activity to other revision control environments: functionality developed in Perforce is merged into CVS when it is considered mature enough; and the *src* tree (that Subversion is used for) is automatically exported to CVS for distribution purposes.[104] That is why the number of code contributions (checked into CVS) is decreasing steadily since 2003, while the LOC and KB added to it do not. Modifications checked into the Perforce repository are not back-ported one by one; instead, a set of interrelated modifications that are considered mature enough, are bundled together and committed to CVS as one big modification. The practice of committing as small changes to the repository as is possible is the technical norm in FOSS projects on account of facilitating trouble-shooting: when changes to the codebase are small, it is easier to pinpoint which one is responsible for a breakdown in product integration ('broken build' in FreeBSD terminology) and reverse it (FreeBSD 2011a; Holck & Jørgensen 2003/2004, p. 46; Kroah-Hartman 2005).[105] For FreeBSD committers, as Holck and Jørgensen (2003/2004, p. 46) explain, 'working with small changes is a consequence of the obligation to integrate one's contributions': for 'the obligation to preserve the development in a working state

104 According to the *FreeBSD Committer's Guide* (2011a), 'as of June 2008, Subversion is used for the *src* tree...the *src* tree is automatically exported to CVS for compatibility reasons only (e.g. CVSup). The "official" *src* repository is not stored in CVS but in Subversion'. As regards the role of Perforce in FreeBSD development, Scott Long (2010), committer since 2000, former Core Team member and former head of the Release Engineering team, explains that it is currently being used 'to manage experimental projects' which are not ready for the main repositories. Also, see the FreeBSD News Flash entry for 3 June 2008 at <http://www.freebsd.org/news/2008/index.html>; Watson (2006); Wemm (2008); and the 'Subversion Primer' at <http://wiki.freebsd.org/SubversionPrimer>.

105 Committers are expected to break up their changes. The *FreeBSD Committer's Guide* advises committers to 'avoid committing several unrelated changes in one go. It makes merging difficult, and also makes it harder to determine which change is the culprit if a bug crops up' (FreeBSD 2011a). This is similar to other large FOSS projects like Linux, for example, where changes must be 'broken up into tiny, individual portions' as that practice makes it easier to verify the correctness of changes and 'debug when something goes wrong' (Kroah-Hartman 2005).

could be seen as implying an implicit rule saying: avoid the introduction of large and complex new features'. But as changes are now being tested in Perforce and Subversion and merged into CVS when they are mature enough, there is no reason for keeping one's changes to CVS small: changes tried out in Perforce or Subversion can be lumped together and committed as one big change in CVS. This explains why the number of contributions logged onto CVS decreases steadily after 2003, while the LOC and KB added to the codebase do not.[106]

Given that our quantitative analysis is based on activity logs collected from CVS alone, the practical implications for the purposes of our analysis of this migration from one development environment to another (i.e. from CVS to Subversion and Perforce) – and the modification of development practice this implies as changes are no longer committed individually to CVS but in bundles – is that the number of code contributions checked into the CVS repository is an inferior indicator of the rate of technical change in the project compared to the LOC or KB added to the codebase. For the same reason, the LOC and KB added per committer make for more robust indicators of average productivity than the number of code contributions per committer. And so it is on these proxies that our analysis shall be based from now on.

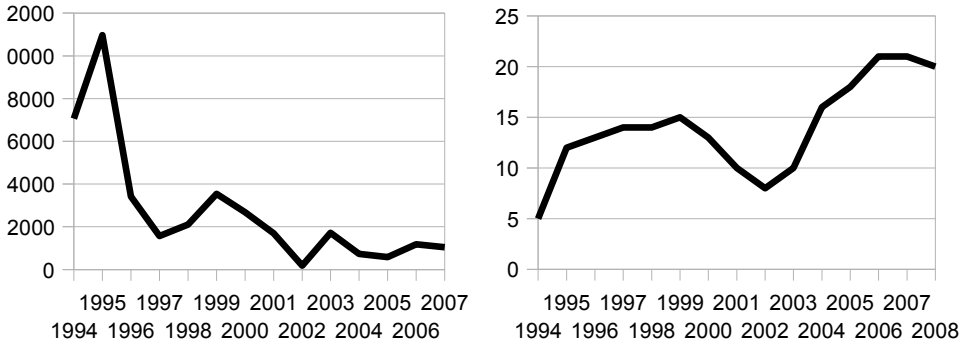


Fig. 7.12: Average productivity versus complexity

All the same, our analysis of descriptive statistics shows that average

106 We are grateful to the FreeBSD developers in attendance at the T-Dose conference in 3 October 2009 in Eindhoven for clarifying the implications of the migration of development activity from CVS to Perforce and Subversion.

productivity falls over time and that its tendential fall is paralleled by higher levels of complexity (as indicated by the tendency for the propagation cost to rise). Contrasting the LOC added per committer to the codebase with its propagation cost in Fig. 7.12 above suggests that higher levels of complexity are correlated with a fall in labour productivity.

It follows from the definition of product modularity as 'a technical and organisational way to manage complexity' (Osterloh & Rota 2007, p. 160) that an increase of complexity amounts to a reduction of modularity.[107] Observing therefore the tendency for the propagation cost to rise hints at a deterioration in the levels of modularity. Consequently, in consideration of the parallel fall in the levels of productivity and modularity over time, our analysis so far supports the hypothesis that modularity has a positive effect on labour productivity. However, as the impact of modularity can be more rigorously examined at the level of individual modules, it is to their quantitative analysis that we turn now.

QUANTITATIVE ANALYSIS

To examine the effect of modularity on productivity at the level of individual modules, we conducted a regression analysis, using the same dataset as in testing *H2* and *H2R*, consisting of a stratified random sample of 29 FreeBSD modules[108] with observations spanning fourteen years of development activity from 1994 to 2008. For the test, we used the number of *KB added per committer* as dependent variable and *integrality_index_lag* as independent variable. As in the tests performed in chapter 6, first, the predictor was lagged a year in order to test the effect of modularity in year=*t* on average labour productivity in year=*t+1* (since this transformation reflects more faithfully the logical order of the hypothesised causal relationship [that is, if A causes B, then A precedes B] and so by establishing directionality it allows a refinement of the empirical model) and second, the

107 The *system-level* definition of modularity emphasises decomposability, that is, independence of components (i.e. modules). As the emphasis is put on reducing, if not eliminating, interactions between components, modularity is equivalent to a reduction of complexity at the system-level. The *component-level* definition of modularity, on the other hand, stresses information hiding, that is, encapsulation (i.e. localisation) of interactions within components. It follows from these two definitions that the reduction of complexity at the system-level is accomplished through the encapsulation of interactions within components, as a result of which components' internal complexity increases.

108 See section **Sample selection** in chapter 3 for a description of the procedure used for sample selection.

intraclass correlation between observations pertaining to the same module was taken into account by using the group variable *module* in the regression analysis.[109] Fig. 7.13 illustrates the empirical model:

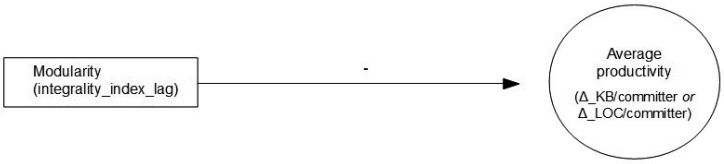


Fig. 7.13: Empirical model H3

The test however indicated no model significance (p = *ns*):

```
Random-effects GLS regression                   Number of obs      =       242
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0058                         Obs per group: min =         4
       between = 0.0560                                        avg =       8.3
       overall = 0.0153                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      0.09
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.7689

------------------------------------------------------------------------------
D_KB_per_c~r |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
integrali~ag |   .0263804   .0897726     0.29   0.769    -.1495707    .2023315
       _cons |   2.035681   .9299735     2.19   0.029     .2129659    3.858395
-------------+----------------------------------------------------------------
     sigma_u |  4.3135107
     sigma_e |  3.7915038
         rho |  .56413937   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 7.1: Regression results–Effect of Modularity on Productivity

We repeated the test, using *LOC added per committer* as dependent variable but again the analysis indicated no model significance (p = *ns*). As an alternative test, we resorted to testing the model with *propagation_cost_lag* as predictor, but neither did that test indicate model significance (p = *ns*). Based on the results of our statistical tests up to this point, we find no empirical support for the hypothesis that modularity has a positive effect on labour productivity.

Scale considerations

To ensure that the effect of scale was not overlooked, we repeated the above tests,

109 The group variable is retained throughout all regression analyses presented in this chapter.

using the median of committers to differentiate between conditions of large-scale and small-scale development. As the median number of committers is eight, a small-scale development process is reflected in years that fewer than nine committers participate in the development process of a module (i.e. committers < 9), while large-scale is reflected in years in which the number of participating committers exceeds eight (i.e. committers > 8). Thus, to examine the effect of modularity on average productivity under conditions of large-scale development, we conducted a regression analysis with *KB added per committer* as dependent variable and *integrality_index_lag* as independent variable, *excluding years in which committers are fewer than nine*. The test indicated strong model significance ($p < 0.050$):

```
Random-effects GLS regression                   Number of obs      =       121
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0227                         Obs per group: min =         1
       between = 0.1901                                        avg =       5.0
       overall = 0.0333                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      4.10
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0428

------------------------------------------------------------------------------
D_KB_per_c~r |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
integrali~ag |  -.2352893   .1161769    -2.03   0.043    -.4629918   -.0075867
       _cons |   2.577659   .4789321     5.38   0.000     1.638969    3.516349
-------------+----------------------------------------------------------------
     sigma_u |          0
     sigma_e |  2.9439646
         rho |          0   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 7.2: Regression results–Effect of Modularity on Productivity in Large-scale conditions (Condition: if committers > 8)

The coefficient for integrality_index_lag is significant and negative, suggesting thus that an increase in integrality_index_lag leads to a decrease in the KB added per committer. Hence, higher levels of modularity result in an increase in average productivity in large-scale development settings.

Substituting *LOC added per committer* for KB added per committer (see Table 7.3 below) yields similar results ($p < 0.005$). Again, the coefficient for integrality_index_lag is negative and significant, suggesting that an increase of integrality_index_lag results in a decrease of the LOC added per committer. That is to say, higher levels of modularity bring about a rise in average productivity in conditions of large-scale development.

```
Random-effects GLS regression                   Number of obs      =       121
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0006                         Obs per group: min =         1
       between = 0.2837                                        avg =       5.0
       overall = 0.0770                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      9.93
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0016

------------------------------------------------------------------------------
D_LOC_per_~r |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
integrali~ag |  -13.85818   4.397171    -3.15   0.002    -22.47647   -5.239882
       _cons |   81.71068   18.12706     4.51   0.000     46.18229    117.2391
-------------+----------------------------------------------------------------
     sigma_u |          0
     sigma_e |  116.41532
         rho |          0   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 7.3: Regression results–Effect of Modularity on Productivity in Large-scale conditions (Condition: if committers > 8)

To summarise our results so far, both tests show that modularity has a positive effect on average labour productivity in the context of large-scale development.

Subsequently to testing the effect of modularity on average productivity in conditions of large-scale development work, we proceeded to examine the effect of modularity on productivity in small-large development environments. To do so, we performed a regression analysis with *KB added per committer* as dependent variable and *integrality_index_lag* as independent variable, but we now *excluded years in which committers exceed eight*. The test however indicated no model significance (p = *ns*):[110]

```
Random-effects GLS regression                   Number of obs      =       121
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0014                         Obs per group: min =         1
       between = 0.0150                                        avg =       5.0
       overall = 0.0134                                        max =        10

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      0.00
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.9480

------------------------------------------------------------------------------
D_KB_per_c~r |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
integrali~ag |   .0071944   .1102143     0.07   0.948    -.2088216    .2232104
       _cons |   2.585224   1.521417     1.70   0.089    -.3966982    5.567146
-------------+----------------------------------------------------------------
     sigma_u |  6.6802182
     sigma_e |  3.8494375
         rho |   .75071836   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 7.4: Regression results–Effect of Modularity on Productivity in Small-scale conditions (Condition: if committers < 9)

110 Nor did testing the model with *LOC added per committer* as dependent variable (p = *ns*).

In contrast to the results of the tests focusing on large-scale development (*Tables 7.2, 7.3*), the last test (*Table 7.4*) suggests that modularity has no significant effect on average productivity in small-scale development settings.

In summary, finding no significant effect of modularity on average productivity in the statistical tests that focus on small-scale development leads us to the conclusion that product modularity has no significant effect on average productivity in conditions of small-scale development. By contrast, the tests centred on large-scale development conditions consistently indicate that modularity has a positive effect on average productivity.

An eventuality that cannot be excluded in attempting to come to grips with the absence of a quantitatively perceptible effect of modularity on labour productivity when the scale of development remains small (i.e. participation is limited to fewer than nine committers) is that the claimed productivity benefits of modularity are being eroded by the increased development costs attendant upon the modularisation process. As an empirical study of seventy-five (Java) FOSS projects concluded, modularity 'represents an important managerial variable to implement the more open governance approach that characterizes OS projects', which, in turn, inflates development costs (Capra et al. 2008, p. 765). In fact, there is evidence to the effect that the development cost of a reusable modular software component could be as much as ten times higher than that of a non-modular one (Garud & Kumaraswamy 1995, p. 97). Nor should coordination costs closely tied to widening the scope of experimentation and variation through modular product structures be ignored: making effective use of the economies of substitution enabled by modular product structures is not devoid of management overhead (Garzarelli & Galoppini 2003). In short, the modularisation process entails significant development and maintenance costs: its benefits, as a result, become *visible* only when the scale of development has been so enlarged that the need to mitigate the adverse effects of increasing scale takes on a pressing character. The benefits of modularity out-weight its costs in conditions of large-scale development because it is only in circumstances where decreasing returns to scale become an issue that the potential of modularity can be fully realised.

In view of these results, we find so far strong empirical support for hypothesis *H3*, according to which *modularity has a positive effect on labour productivity in FreeBSD.*

EFFECT OF MODULARITY ON CORE DEVELOPERS PERFORMANCE

To delve more deeply into the effect of modularity on productivity, our analysis turns to the performance of core developers. It follows directly from the fundamental premise of Brooks' Law (i.e. as more persons are added to the group, the potential interpersonal communication paths rise exponentially) that adding more persons to work on a software project affects negatively the productivity of its core developers by forcing them to channel part of their time into coordinating their tasks with those performed by new members. It is therefore important, as modularity is employed with the aim of mitigating the negative effects of increasing group size on productivity, to probe also its impact on the performance of core developers.

To do this, we performed a regression analysis, using the same dataset as in all previous statistical tests, with the number of code contributions made by the *top two committers* for each module as dependent variable, which we use as our first indicator of core developers' output in individual modules,[111] and *integrality_index_lag* as independent variable. Fig. 7.14 illustrates the empirical model:

Fig. 7.14: Empirical model

However, as can be seen in Table 7.5 below, the test indicated no model significance (p = *ns*).[112]

111 Of course, years in which modules were developed by fewer than three committers were excluded from the regression analysis.

112 Nor did the test with the contributions of the top ten percent of committers as dependent variable (p = *ns*), which we used as an alternative indicator of core developers' output in individual modules.

```
Random-effects GLS regression                   Number of obs      =       229
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0012                         Obs per group: min =         3
       between = 0.1383                                        avg =       7.9
       overall = 0.0894                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(1)       =      0.61
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.4357

------------------------------------------------------------------------------
top_2_comm~s |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
integrali~ag |  -1.336736   1.715052    -0.78   0.436    -4.698177    2.024705
       _cons |   69.98846   17.18699     4.07   0.000     36.30259    103.6743
-------------+----------------------------------------------------------------
     sigma_u |  78.484894
     sigma_e |   64.16151
         rho |   .59940952   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 7.5: Regression results–Effect of Modularity on Core Developers Output (Condition: if committers > 2)

Following this step, we attempted to refine the empirical model by drawing upon the results of testing *H1* and *H2*. So, *propagation_cost_lag* and *ext_dependencies_lag* were added as independent variables. To control for environmental factors, as when testing *H1* and *H2*, the variables *propagation cost_j_lag*, *ext_dependencies_per_mod_j_lag* and *maturity_ln* were included too. Fig. 7.15 illustrates the expanded empirical model:

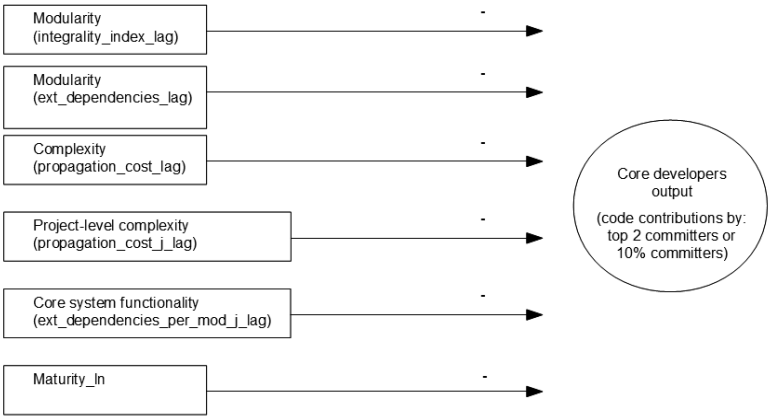


Fig. 7.15: Expanded empirical model

However, neither this regression analysis indicated model significance (p = *ns*). As no indicator of modularity was found to have a significant effect, it suggests that core developers' output is not affected by modularity.

Scale considerations

To come to grips with the absence of a perceptible effect of modularity on core developers' output and to verify the robustness of this finding, we attempted to further refine the analysis by differentiating between conditions of large-scale and small-scale development. Modifying the prism of analysis in this manner ensures that the effect of the scale of development is not ignored, enabling us to contrast the impact of modularity (on core developers' output) under conditions of large-scale development with its impact in conditions of small-scale development. To do this, the median of committers was the criterion used to disaggregate large-scale from small-scale development conditions: that is, as the median number of committers is eight, a small-scale development process is reflected in years with fewer than nine committers, while years in which committers exceed eight reflect a large-scale development process. Thus, to examine the effect of modularity on core developers' output in conditions of large-scale development, we carried out a regression analysis with the contributions of the *top two committers* as dependent variable and *propagation_cost_lag, ext_dependencies_lag, integrality_index_lag, propagation_cost_j_lag, ext_dependencies_per_mod_j_lag* and *maturity_ln* as independent variables, *excluding years in which committers are fewer than nine*.

```
Random-effects GLS regression                   Number of obs      =       121
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0137                         Obs per group: min =         1
       between = 0.4094                                        avg =       5.0
       overall = 0.3348                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(6)       =     23.50
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0006

------------------------------------------------------------------------------
top_2_comm~s |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
propag~t_lag |  -14.98715   140.1516    -0.11   0.915    -289.6793     259.705
ext_de~s_lag |   1.151737   .3652911     3.15   0.002     .4357791    1.867694
integrali~ag |  -15.97723   4.875752    -3.28   0.001    -25.53353   -6.420933
propag~j_lag |  -324.2617   281.7503    -1.15   0.250    -876.4822    227.9589
ext_de~j_lag |    .387209   1.660828     0.23   0.816    -2.867954    3.642372
 maturity_ln |  -24208.13   9746.463    -2.48   0.013    -43310.85   -5105.413
       _cons |   184093.7   74030.14     2.49   0.013      38997.3    329190.1
-------------+----------------------------------------------------------------
     sigma_u |  43.302807
     sigma_e |  85.478453
         rho |  .20422506   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 7.6: Regression results–Effect of Modularity on Core developers output in Large-scale conditions (Condition: if committers > 8)

The test indicated strong model significance ($p < 0.001$). As Table 7.6 above shows, integrality_index_lag has a significant effect on the contributions of the top two committers. Specifically, the coefficient for integrality_index_lag is negative, suggesting that higher levels of modularity result in an increase of core developers' output.

Testing the model with the contributions of the *top ten percent of committers* (which we used as an alternative indicator of core developers' output in individual modules) as dependent variable reinforces the conclusions drawn from the last test ($p < 0.050$):

```
Random-effects GLS regression                   Number of obs      =       121
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0289                         Obs per group: min =         1
       between = 0.3755                                        avg =       5.0
       overall = 0.3437                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(6)       =     18.23
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0057

------------------------------------------------------------------------------
top_10perc~t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
propag~t_lag |  -39.14691   145.3022    -0.27   0.788    -323.9339    245.6401
ext_de~s_lag |   1.255546   .3982866     3.15   0.002      .474919    2.036174
integrali~ag |  -12.15148   5.038609    -2.41   0.016    -22.02697   -2.27599
propag~j_lag |  -522.8809    261.317    -2.00   0.045    -1035.053   -10.70893
ext_de~j_lag |   1.464094    1.53372     0.95   0.340    -1.541941    4.470129
 maturity_ln |  -19593.51   9624.728    -2.04   0.042    -38457.63   -729.3926
       _cons |   148969.9   73107.03     2.04   0.042     5682.793    292257.1
-------------+----------------------------------------------------------------
     sigma_u |  61.568131
     sigma_e |  80.349718
         rho |  .36993669   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 7.7: Regression results–Effect of Modularity on Core developers output in Large-scale conditions (Condition: if committers > 8)

Consistent with the last test, the coefficient for integrality_index_lag is significant and negative, suggesting that higher levels of modularity result in an increase of core developers' output. Crucially enough, testing the effect of modularity on core developers' output in conditions of large-scale development indicates that higher levels of modularity bring about an increase of core developers' output.

Having therefore ascertained that modularity has a positive effect on the output of core developers under conditions of large-scale development, we proceeded to test the relationship between these two variables in conditions of small-scale development. So, we carried out a regression analysis as before with the contributions of the *top two committers* as dependent variable and *propagation_cost_lag, ext_dependencies_lag, integrality_index_lag,*

propagation_cost_j_lag, *ext_dependencies_per_mod_j_lag* and *maturity_ln* as independent variables, except that we now *excluded years in which committers are fewer than two and more than eight*. The test however found no significant effect of any predictor on core developers' output (p = *ns*):[113]

```
Random-effects GLS regression                   Number of obs      =       242
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0272                         Obs per group: min =         4
       between = 0.3087                                        avg =       8.3
       overall = 0.1897                                        max =        13

Random effects u_i ~ Gaussian                   Wald chi2(6)       =     11.98
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0623

------------------------------------------------------------------------------
top_2_comm~s |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
propag~t_lag |  -71.31229   71.93099    -0.99   0.321    -212.2944    69.66986
ext_de~s_lag |   .3505202    .236859     1.48   0.139    -.1137149    .8147553
integrali~ag |  -1.883918   1.632277    -1.15   0.248    -5.083123    1.315286
propag~j_lag |  -120.8887   120.3114    -1.00   0.315    -356.6946    114.9173
ext_de~j_lag |   .1639511   .7860096     0.21   0.835    -1.376599    1.704502
 maturity_ln |  -7281.522   4646.604    -1.57   0.117     -16388.7    1825.655
       _cons |   55436.58   35294.71     1.57   0.116    -13739.77    124612.9
-------------+----------------------------------------------------------------
     sigma_u |  72.022354
     sigma_e |  61.653693
         rho |  .57710171   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 7.8: Regression results–Effect of Modularity on Core developers output in Small-scale conditions (Condition: if 2 < committers < 9)

To sum up, both tests focusing on large-scale development conditions (*Tables 7.6, 7.7*) indicate that modularity has a positive effect on the output of core developers. By contrast, the tests centred on small-scale development conditions (*Tables 7.8*) suggest that modularity has no significant effect on core developers. To understand why, one needs to contrast the work process of core developers of large-scale projects with that of core developers in small-scale projects. The key difference is that core developers specialise far more in large-scale projects than in small-scale ones. The extent of their specialisation is, of course, in the first place determined by the learning costs involved in familiarising themselves with the codebase and grasping all possible interactions. As such, core developers' specialisation is an adaptation to increased scale: it is the strategy they employ to cope with the gigantic learning costs attendant on large codebases.

On the contrary, core developers of small-scale projects, regardless of their degree of decomposability (i.e. the extent that the software product can be

113 Nor did testing the model with the code contributions of the top ten percent of committers as dependent variable (p = *ns*).

decomposed into components, which can be developed independently of each other), are typically familiar with the entire codebase. Their participation is not only extensive but also many-sided, encompassing most, if not all, activities in the project. However, as the scale of the project expands – that is, as more developers are added to the project and the codebase grows bigger – grasping the totality of the development process becomes exceedingly difficult. Consequently, core developers can remain on top of development work only by specialising in some domain(s) of the codebase. Thus, a spontaneous division of labour emerges among core developers out of their choice to focus their participation on that part of the codebase with which they are most familiar. This 'separation of concerns' is reinforced by modular product design: the reduction, if not elimination, of interactions between components (modules) effected through the modularisation process helps ensure that core developers do not need to occupy themselves with activities concerning any areas of the codebase other than those which form the epicentre of their interest.

In small-scale projects, on the other hand, where such a 'separation of concerns' is not considered necessary or desirable, it is immaterial to core developers' work process whether product design is modular or not. Insofar as core developers can keep themselves familiar with the entire codebase, the impetus for specialisation is lacking. Consequently, as long as conditions of small-scale development prevail, core developers' participation in code production is not subject to a division of labour, but spans the entire codebase. That is why the regression analysis found no significant effect of modularity on core developers' output in small-scale, as opposed to large-scale development settings. While understanding and keeping track of all possible component interactions (i.e. dependencies between modules) in a small-scale project may be both desirable and feasible for its core developers, it is by no means possible for the core developers of large FOSS projects, such as FreeBSD or Linux, to do so. Enlarging the scale of the project militates in favour of extending core developers' specialisation. Thus alone can they maintain their code leadership position when confronted with the cognitive difficulties that keeping oneself familiar with a large codebase entails. Modular product design reinforces the already existing tendency of core developers toward specialisation in large-scale development settings by facilitating the independent development of distinct product components (modules), helping ensure thus a scalable self-assignment to tasks.

CONCLUDING REMARKS

All statistical tests performed in this chapter indicate that in *conditions of large-scale development* modularity has a positive effect on average labour productivity as well as on the performance of core developers. In consequence of the strong empirical support these results provide for the claimed effect of modularity on labour productivity in projects characterised by expanding scale, *hypothesis H3 is verified.*

In the next chapter, we put Brooks' Law to the test by examining the effect of expanding group size on labour productivity in the development process of FreeBSD.

CHAPTER 8: DOES BROOKS' LAW HOLD IN FREEBSD?

INTRODUCTION

Observing in Fig. 8.1 that in the course of thirteen years of development, the committers' base has grown from 17 persons to about 200 and average productivity has decreased by eighty-five percent[114] is suggestive of the negative effect of increasing group size on labour productivity known as *Brooks' Law*.

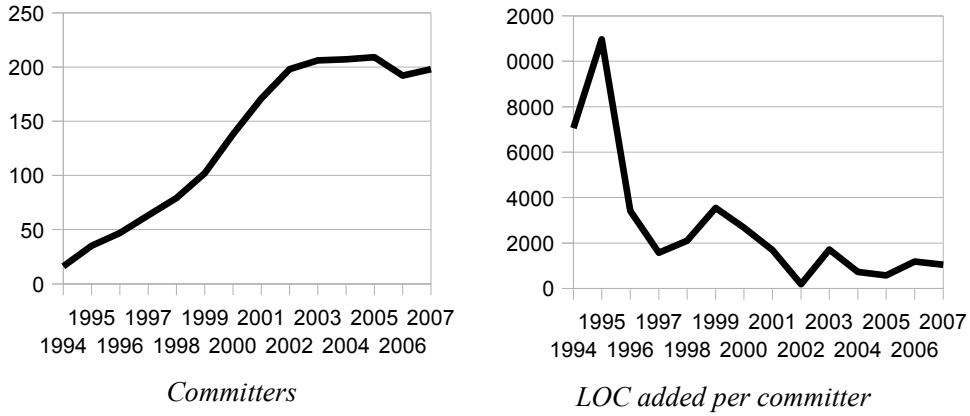


Fig. 8.1: Committers (*src*) versus average productivity

In barest outline, this theory holds that adding more developers to a software development project occasions a fall in group productivity due to the communication and coordination costs that increasing group size entails (Brooks 1995). The cause of the problem is that as more persons are added to the developers' group, 'the potential interpersonal communication paths or interactions, which can lead to diseconomies of scale', grow exponentially. In consequence, 'the more individuals that are added to a team, the more of each individual's time is consumed in communication with other team members about updating common information, handling errors, or resolving the use of shared resources' (Boehm 1981, p. 190; see

114 In specific, by eighty-five percent in LOC and eighty-four percent in KB.

also section **The productivity paradox in software development** in chapter 2). As the increase in inputs (i.e. developers) to the software production process results in a less than proportionate increase in outputs (i.e. code contributions, LOC or KB), the production process exhibits decreasing returns to scale: average labour productivity in the project declines. Stated as a hypothesis to be tested in FreeBSD:

> *An increase of contributors to FreeBSD has a negative effect on labour productivity* (H4)

Fig. 8.2 illustrates how *H4* fits into the research model that sums up the literature review:

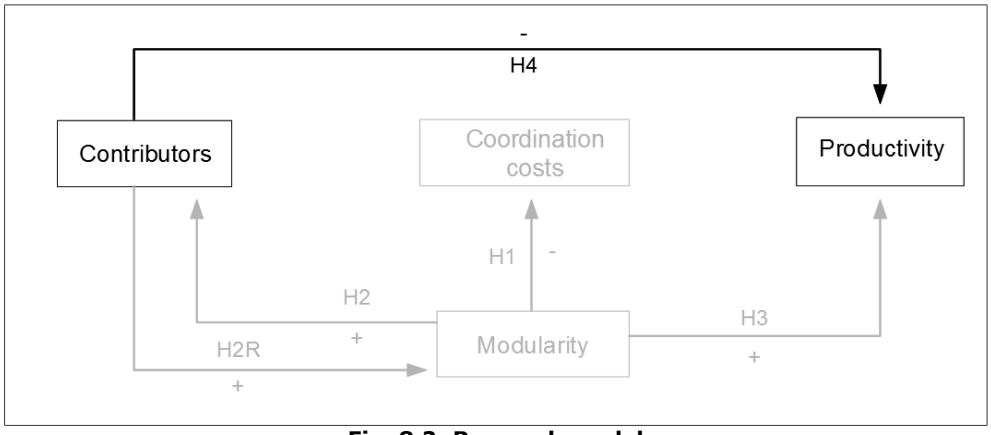


Fig. 8.2: Research model

To put *H4* to the test, we conducted a regression analysis, using the same panel dataset as in all previous statistical tests, with the number of *KB added per committer* as dependent variable and *committers* as independent variable. The intraclass correlation between observations pertaining to the same module was taken into account by using the group variable *module*. To control for the effect of modularity, we included *integrality index* in the regression. To make sure that both independent variables can be included in the regression, we ran a VIF test, whose results confirmed it. Since the VIF value for committers and integrality index (which is 1.08) is smaller than 10, there is no problem including both predictors in the model.

Variable	VIF	1/VIF
committers	1.08	0.925636
integrality_index	1.08	0.925636
Mean VIF	1.08	

Table 8.1: VIF test for regression of KB added per committer on committers and integrality index

Fig. 8.3 illustrates the empirical model:

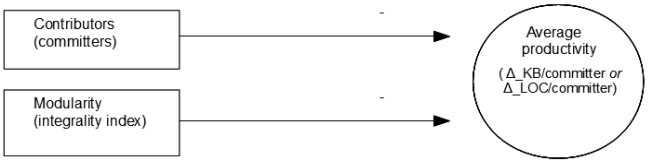


Fig. 8.3: Empirical model H4

The test indicated strong model significance ($p < 0.050$):

```
Random-effects GLS regression                   Number of obs      =       271
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0406                         Obs per group: min =         5
       between = 0.0340                                        avg =       9.3
       overall = 0.0230                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(2)       =      7.78
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0205

------------------------------------------------------------------------------
D_KB_per_c~r |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |  -1.527133   .5712941    -2.67   0.008    -2.646849    -.407417
integralit~x |  -1.165949   .8396305    -1.39   0.165    -2.811594    .4796967
       _cons |   30.69374    8.22837     3.73   0.000     14.56643    46.82105
-------------+----------------------------------------------------------------
     sigma_u |  14.309742
     sigma_e |  55.392992
         rho |  .06256009   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 8.2: Regression results–Effect of Contributors on Productivity

As the coefficient for committers is negative and significant, it indicates that an increase of committers brings about a fall in average productivity. The test found no significant effect of integrality index on the number of KB added per committer, thus indicating that modularity does not affect average productivity.

Substituting *LOC added per committer* for KB added per committer increases the model's significance ($p < 0.001$):

```
Random-effects GLS regression                   Number of obs      =       271
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.1193                         Obs per group: min =         5
       between = 0.0018                                        avg =       9.3
       overall = 0.0355                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(2)       =     16.19
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0003

------------------------------------------------------------------------------
D_LOC_per_~r |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |   -43.7585    11.1319    -3.93   0.000    -65.57661   -21.94038
integralit~x |  -28.14489   16.36132    -1.72   0.085    -60.21249    3.922709
       _cons |   811.6442   160.6655     5.05   0.000     496.7456    1126.543
-------------+----------------------------------------------------------------
     sigma_u |  287.57671
     sigma_e |   1029.994
         rho |  .07231657   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 8.3: Regression results–Effect of Contributors on Productivity

Again, the coefficient for committers is negative and significant, suggesting that an increase of contributors to a module results in a decrease of the LOC added per committer, that is, a fall in productivity, confirming thus Brooks' prognosis. Also, as in the previous test, modularity does not appear to have a significant effect on productivity.

Summing up our results so far, both tests indicate that increasing group size has a negative effect on average labour productivity, thus confirming the hypothesis we referred to in chapter 2 as *Brooks' Law*: the tendency for productivity in a group of software developers to decline when more developers are added to the group.

DISAGGREGATING CORE DEVELOPERS' PRODUCTIVITY

To gain further insight into this phenomenon, we attempted to disaggregate the productivity of core developers from the broader base of committers so as to ascertain whether the fall in average productivity in the development process is due to a fall in the output of core developers[115] or a disproportionate increase of 'lower-contribution' committers. To operationalise this inquiry, we counted the number of code contributions checked into the codebase per year by the ten most productive committers in that year and contrasted it with the total number of code

115 We use the characterisation *core developers* to refer to 'high-contribution' committers, though the FreeBSD project does not use this term on the grounds that it can mislead one to conflate prolific committers with core team members (see FreeBSD committer Greg Lehey's comments in Slashdot 2003).

contributions over time in Fig. 8.4.

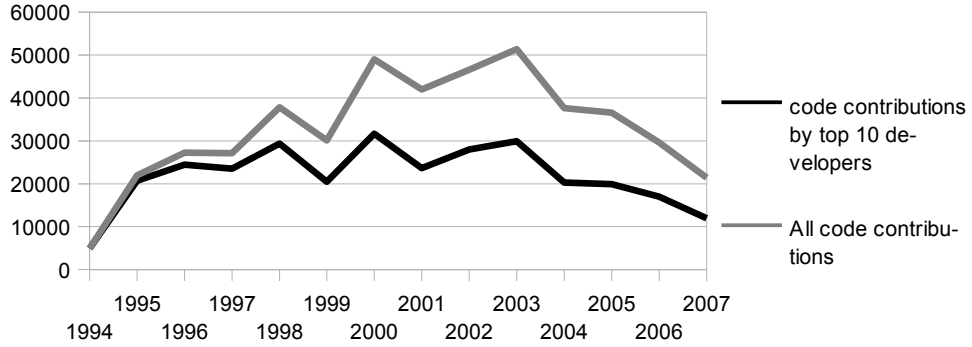


Fig. 8.4: Code contributions by top 10 committers versus all code contributions

Looking at the graph, we observe first that the production output of the ten most productive committers manifests exactly the same fluctuations as that of all committers considered as a group. Second, it manifests no visible sign of a negative effect caused by the historical growth of the group of committers. To illustrate, consider the growth of committers in comparison with the code contributions made by the top ten committers in Fig. 8.5:

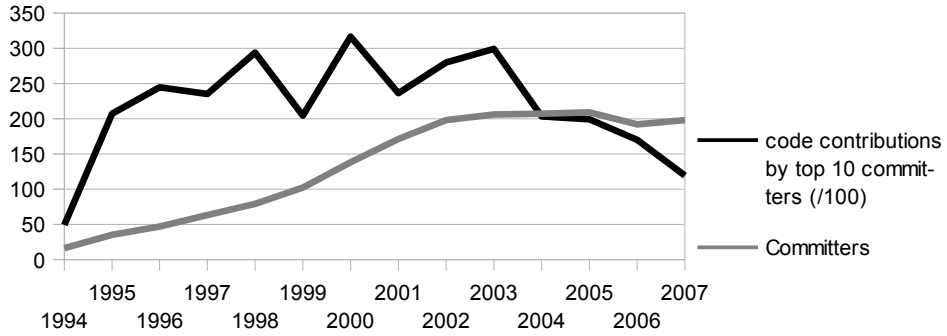


Fig. 8.5: Code contributions by top 10 committers versus all committers

Yet if the dramatic enlargement of project scale that is reflected in the expansion of the committers group has not had a negative effect on the output of core developers, this means that either core developers work on the project for increasingly longer hours over time (so that the time and effort they expend in communicating and coordinating their activities with other project participants

does not come at the expense of the time and effort they devote to producing code) or their work is not burdened with higher coordination costs due to increasing group size, as Brooks' Law predicts.

To find out, we decided to do a survey. First, we identified the 58 individuals that populated the ranks of the top ten committers over time. We found a valid email address for 53 of them and sent them an email questionnaire designed to find out whether the amount of time they spend on the project increases over time and to what extent that is due to non-coding activities (e.g. time spent on coordinating).[116] This attempt generated a 52.8% response rate. Of the 28 committers who replied, more than half (53.5%) remarked the tendency to spend more time over time. With respect to the stretch of non-coding tasks, half of the respondents (50%) noted that their peak activity periods were accompanied by increased non-coding tasks. These answers highlight how time-demanding it is to be a core developer and in parallel how unlikely it is for core developers to abstain from non-coding activities, focusing only on coding. The burden, however, of dealing with non-coding tasks is often seen as a natural consequence of widening one's extent of participation in the project. The coordination costs burdening the work of core developers are not experienced as a result of the historical expansion of the committers group, but as a consequence of the scope of coding tasks they have self-selected. It is not uncommon of developers to start on the project by making changes to relatively self-contained areas of the codebase (such as modules that are not part of the core system), but turn over time to coding tasks of a larger scope, which affect the work of many more developers. Consequently, the need to coordinate changes with other developers increases in proportion to the scope of coding tasks one tackles.[117] That this perspective traces the cause of coordination costs to the scope of one's coding activities should not be construed as implying that coordination costs are independent of the overall scale of the project; rather, their relationship is mediated by the scope of coding tasks committers choose to work on. It is important to note at this point that of the fifty-eight committers who populated the ranks of the ten most productive committers in the space of thirteen years, only three have sustained this level of performance for ten years or more. The average number of years one is part of the group is 3.5. Obviously, being a core developer implies such an expenditure of time that only few committers are in the position to shoulder for extended periods. Given that working long hours has been

116 A more elaborate description of the procedure used to analyse the replies thus collected is given in **Appendix IV: Core developers survey**.

117 We are indebted to FreeBSD committer Nate Lawson for pointing this out.

a constant factor for core developers and their collective output manifests no tendency to fall over time, we are led to the conclusion that the coordination costs brought about by the expansion of the committers group have not so far affected negatively core developers performance.

To recap, as the increase of committers has not brought about a fall in core developers' output, our analysis of descriptive statistics suggests that the marked fall in average productivity is caused by the disproportionate increase of low-contribution committers in the group, rather than by a decrease of core developers' productivity. In order to study in a more rigorous manner the effect of group size on core developers, we proceed now to a quantitative analysis at the level of individual modules.

EFFECT OF GROUP SIZE ON CORE DEVELOPERS PERFORMANCE

To examine the effect of group size on the production output of core developers, we conducted a regression analysis, using the same dataset[118] as in the previous two chapters, with the number of *code contributions of the two most productive committers* specific to each module as dependent variable[119] and the number of *committers* to each module as independent variable. To account for the intraclass correlation between observations pertaining to the same model, we used the group variable *module*. Also, *integrality index* was included in the regression in order to control for modularity. Fig. 8.6 depicts the empirical model:

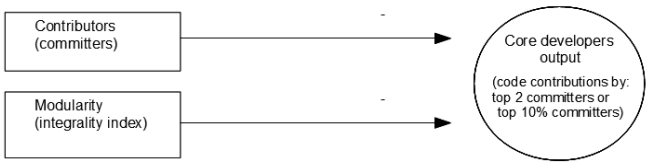


Fig. 8.6: Empirical model

The test indicated strong model significance ($p < 0.001$):[120]

118 See section **Sample selection** in chapter 3 for a full description of the procedure used to draw the sample.

119 Years in which modules were developed by fewer than three committers were excluded from the regression analysis.

120 To explore the time-structure of processes, we also tested the model with the predictor

```
Random-effects GLS regression                   Number of obs      =       255
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0037                         Obs per group: min =         4
       between = 0.6725                                        avg =       8.8
       overall = 0.3495                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(2)       =     16.75
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0002

------------------------------------------------------------------------------
top_2_comm~s |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |   3.282335   .9244487     3.55   0.000     1.470449    5.094221
integralit~x |  -1.920364   1.398781    -1.37   0.170    -4.661925    .8211962
       _cons |   42.50014   14.62092     2.91   0.004     13.84366    71.15662
-------------+----------------------------------------------------------------
     sigma_u |  39.194804
     sigma_e |  62.249584
         rho |  .28389643   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 8.4: Regression results–Effect of Contributors on Core developers output (Condition: if committers > 2)

As before, we see that integrality index has no significant effect on the output of each module's two most prolific committers. However, contrary to our expectations, the coefficient for committers is positive and significant, indicating that an increase of group size at the component-level leads to an increase of core developers' output. Substituting the number of code contributions of *the most productive ten percent of committers* for the code contributions of the two most productive committers, yields the same results ($p < 0.001$):[121]

```
Random-effects GLS regression                   Number of obs      =       271
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.1182                         Obs per group: min =         5
       between = 0.7450                                        avg =       9.3
       overall = 0.4984                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(2)       =     81.95
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
top_10perc~t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |   6.815181   .7714671     8.83   0.000     5.303133    8.327229
integralit~x |  -.8727042   1.199744    -0.73   0.467    -3.224159     1.47875
       _cons |  -6.618627   12.56009    -0.53   0.598    -31.23595     17.9987
-------------+----------------------------------------------------------------
     sigma_u |  39.871689
     sigma_e |   58.05085
         rho |  .32053688   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 8.5: Regression results–Effect of Contributors on Core developers output

transformed (i.e. lagged a year): the test indicated no model significance (p = *ns*) and found no significant effect of *committers_lag* ($\alpha = 0.05$, $\beta = -0.14$, $p = 0.86$).

121 In experimenting with the model, we also tested it with the predictor transformed (i.e. lagged a year): the test indicated no model significance (p = *ns*) and found no significant effect of *committers_lag* ($\alpha = 0.05$, $\beta = 1.29$, $p = 0.07$).

Whereas, as in the last test, modularity does not appear to influence the output of the most productive ten percent of committers, the coefficient for committers is again positive and significant, indicating that an increase of committers results in an increase of the code contributions of the most productive ten percent of them, reinforcing thus the conclusion drawn from the last test that an increase of group size leads to an increase of core developers' performance.

At first sight, this seems to contradict our previous finding that an increase of group size brings about a fall in average productivity (*Tables 8.2, 8.3*). But this contradiction is only a seeming one, for as we have already noted in the analysis of descriptive statistics the output of core developers (as reflected in the output of the top ten committers per year) is not negatively affected by the increase of group size (*Fig. 8.4, 8.5*). The last two statistical tests (*Tables 8.4, 8.5*) reinforce this conclusion, indicating in fact that at the level of individual modules, core developers' output rises when more committers are added to the group. Of course, the positive effect that increasing group size exerts on core developers' output is nothing short of remarkable, for it implies that unless core developers' extent of participation in the project increases over time, that is, unless core developers devote increasingly more time, Brooks' Law does not hold in the project.

Scale considerations

To verify the robustness of this finding, we attempted to further refine the analysis by distinguishing conditions of large-scale development from conditions of small-scale development. This we did by using the median of committers (i.e. eight) as a reflection of the scale of development. By this criterion, large-scale development is reflected in years with more than eight committers, while small-scale development is reflected in years with fewer than nine committers. Thus, to examine the effect of group size on core developers' output in conditions of large-scale development, we conducted a regression analysis with the number of code contributions of the *top two committers* for each module as dependent variable and the number of *committers* as independent variable, *excluding years in which committers are fewer than nine*. As before, *integrality index* was included in the regression as a control variable for the effect of modularity. The test indicated strong model significance ($p < 0.001$):

```
Random-effects GLS regression                   Number of obs      =       123
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0230                         Obs per group: min =         1
       between = 0.7603                                        avg =       5.1
       overall = 0.3724                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(2)       =     71.19
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
top_2_comm~s |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |   7.626893   1.447927     5.27   0.000     4.789009    10.46478
integralit~x |  -15.22334   4.579973    -3.32   0.001    -24.19993    -6.24676
       _cons |   33.44233   32.19719     1.04   0.299      -29.663    96.54766
-------------+----------------------------------------------------------------
     sigma_u |          0
     sigma_e |  85.345932
         rho |          0   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 8.6: Regression results–Effect of Contributors on Code developers output in Large-scale conditions (Condition: if committers > 8)

More specifically, the coefficient for committers is significant and positive, suggesting that an increase of committers to a module results in an increase of the code contributions of the top two committers. That is, the larger the group that develops a module the greater the output of its core developers. As regards the effect of modularity on core developers' output, it now appears to be significant, thus mirroring the results obtained when testing the effect of modularity on core developers' output in large-scale conditions of development in chapter 7 (see *Tables 7.6, 7.7*). The coefficient for integrality index is negative, indicating that decreasing levels of modularity bring about a decrease in core developers' performance.

```
Random-effects GLS regression                   Number of obs      =       123
Group variable: module                          Number of groups   =        24

R-sq:  within  = 0.0077                         Obs per group: min =         1
       between = 0.8052                                        avg =       5.1
       overall = 0.5052                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(2)       =    122.52
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
top_10perc~t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |   10.78388   1.370411     7.87   0.000     8.097921    13.46983
integralit~x |   -13.6444    4.33478    -3.15   0.002    -22.14041   -5.148385
       _cons |  -21.78503   30.47348    -0.71   0.475    -81.51196     37.9419
-------------+----------------------------------------------------------------
     sigma_u |          0
     sigma_e |  80.448329
         rho |          0   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 8.7: Regression results–Effect of Contributors on Code developers output in Large-scale conditions (Condition: if committers > 8)

Testing the model with the *code contributions of the top ten percent of*

committers instead of the contributions of the top two committers (see Table 8.7 above), reinforces the results of the last test ($p < 0.001$). Again, the coefficient for committers is significant and positive, suggesting that an increase of committers to a module leads to an increase of the code contributions of the top ten percent of committers. To recap: both statistical tests indicate that an increase of committers to a large-scale development process results in an increase of core developers' output.

Having established that increasing group size brings about an increase of core developers' output in large-scale development conditions, we proceeded to test this relationship in conditions characteristic of small-scale development. Thus, we repeated the previous two tests *but we now excluded years in which committers are either fewer than three or more than eight*. To control for the effect of modularity, as before, *integrality index* was included in the regression. The test indicated strong model significance ($p < 0.001$):

```
Random-effects GLS regression                   Number of obs      =       271
Group variable: module                          Number of groups   =        29

R-sq:  within  = 0.0037                         Obs per group: min =         5
       between = 0.6823                                        avg =       9.3
       overall = 0.3591                                        max =        14

Random effects u_i ~ Gaussian                   Wald chi2(2)       =     23.73
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
top_2_comm~s |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |   3.796786   .8412546     4.51   0.000     2.147958    5.445615
integralit~x |  -1.498132    1.29801    -1.15   0.248    -4.042186    1.045921
       _cons |   35.05923    13.4501     2.61   0.009     8.697514    61.42096
-------------+----------------------------------------------------------------
     sigma_u |  40.246823
     sigma_e |  62.635585
         rho |  .29222442   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 8.8: Regression results–Effect of Contributors on Code developers output in Small-scale conditions (Condition: if 2 < committers < 9)

We see that the coefficient for committers is significant and positive, suggesting that an increase of committers to a module results in an increase of the contributions of the top two committers. Therefore, even in conditions of small-scale development, an increase of group size causes an increase of core developers' output. As regards the effect of modularity, it appears to be insignificant, in line with the results obtained in chapter 7 when testing the effect of modularity on core developers' output in small-scale development conditions.

Substituting the contributions of the *top ten percent of committers* for the contributions of the top two committers, leads to the same conclusion ($p < 0.001$):

```
Random-effects GLS regression                   Number of obs      =       148
Group variable: module                          Number of groups   =        28

R-sq:  within  = 0.1154                         Obs per group: min =         1
       between = 0.2287                                        avg =       5.3
       overall = 0.1409                                        max =        11

Random effects u_i ~ Gaussian                   Wald chi2(2)       =     19.85
corr(u_i, X)       = 0 (assumed)                Prob > chi2        =    0.0000

------------------------------------------------------------------------------
top_10perc~t |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  committers |    4.26249   .9593469     4.44   0.000     2.382205    6.142776
integralit~x |  -.2059359   .4896469    -0.42   0.674    -1.165626    .7537543
       _cons |   .0740592   7.461875     0.01   0.992    -14.55095    14.69907
-------------+----------------------------------------------------------------
     sigma_u |  24.977892
     sigma_e |  19.427924
         rho |  .62306061   (fraction of variance due to u_i)
------------------------------------------------------------------------------
```

Table 8.9: Regression results–Effect of Contributors on Code developers output in Small-scale conditions (Condition: if committers < 9)

Again, the coefficient for committers is significant and positive, indicating that an increase of committers to a small-scale development process brings about an increase of the code contributions of the top ten percent of committers.

To summarise our results so far, all statistical tests show that an expansion of group size results in an increase of core developers' output, regardless of the modules' scale of development. However, that is not to say that the magnitude of the effect is also the same. As we have seen in Table 8.6, by examining the effect of group size on the output of the top two committers in conditions of large-scale development, we found that the coefficient for committers is *7.62*. By comparison, testing the relationship between these two variables in conditions of small-scale development, as Table 8.8 shows, yields a coefficient for committers of *3.79*. The discrepancy between the two coefficients suggests that the effect of increasing group size on core developers' output is much stronger in conditions of large-scale development: to be precise, the magnitude of the effect is then greater by 101%. This is also attested in the analysis of the effect of increasing group size on the output of the top ten percent of committers: the coefficient for committers is *10.78* in conditions of large-scale development (Table 8.7) versus *4.26* in conditions of small-scale development (Table 8.9), indicating thus that the magnitude of the effect of increasing group size on core developers is greater by 153% in conditions of large-scale development.

As the effect of increasing group size on the output of the top two committers is significantly stronger in large-scale development conditions, it suggests that the productivity of a module's core developers is higher when they are environed by a

group large enough to allow for a finer division of labour within the boundaries of the module. The ability of core developers to effect a division of labour by delegating tasks and responsibilities increases in proportion as the scale of the project is enlarged. Given that tasks can be much more easily delegated within modules than across them, increasing the number of developers working on a module allows its core developers to delegate more responsibilities to other module contributors. In that way, the larger the size of the group developing a module, the greater the ability of its core developers to focus on what they do and like best: to produce code. In addition to allowing a more extensive delegation of tasks within the modules they develop, large development groups have a comparative advantage in characterising and fixing bugs. This constitutes a feedback mechanism that provides core developers with a critical user perspective on how to prioritise tasks in the development process, thereby structuring their work and shaping the direction of development effort.[122] Therefore, large development groups, by enabling an extensive delegation of tasks within the modules they develop and by keeping up a constant flow of bug-reports and bug-fixes, free up core developers' time for coding, while giving structure to their work content.

To sum up, our tests show that core developers' production output is significantly greater in large-scale modules, reinforcing thus our previous finding that an increase of contributors to a module leads to an increase of the output of the module's core developers.

Does modularity negate Brooks' Law?

As in chapter 7, the empirical tests in chapter 8 show that modularity has a significant effect on productivity in large-scale conditions. We previously qualified this finding by arguing that modularity reinforces the tendency of developers to specialise in conditions of increasing scale by facilitating the independent development of distinct components. As developers can concentrate on some one component without having to coordinate their work with others working on different components, their performance is not negatively impacted by increasing group size. We can now put these findings into perspective so as to more directly engage with Brooks' Law. In large-scale conditions, modularity not only offsets but

122 As one of the project founders explains, bug-reports are used in FreeBSD as the criterion by which committers prioritise tasks in the development process (Hubbard 2009). This point is emphasised also in Holck and Jørgensen's (2003/4, p. 45) study of FreeBSD.

actually reverses Brooks' Law so that increasing group size occasions a rise in the performance of core developers. What accounts for the positive effect of adding more developers to the development process of a component (module) on the performance of its core developers is that large groups enable a more extensive division of labour within the components they develop, which allows their core developers to focus on their task of choice, namely new code development. In this sense, modularity creates the conditions in which a project undergoing an expansion of scale can take advantage of the benefits of a more extensive specialisation and division of labour, without incurring a productivity loss (due to the coordination costs attendant upon increasing scale).

CONCLUDING REMARKS

In testing the hypothesis (*H4*) we referred to as Brooks' Law, according to which increasing group size results in a decrease of productivity, we found that though the expansion of the FreeBSD committers group results in a fall in average productivity, seemingly confirming Brooks' prognosis, it also results in a rise in core developers' output, thereby suggesting that the fall in group productivity is not caused by a fall in core developers' productivity, but by the disproportionate increase of lower-contribution committers over time. While these findings provide empirical support for the hypothesis that increasing group size results in a fall in average productivity, they also falsify some of the premises upon which it rests, since the observed fall in average productivity does not spring from a decrease of core developers' performance – whose output rises rather than falls when more committers coalesce around the development of a module – but from the disproportionate increase of lower-contribution committers over time. In the light of these findings, therefore, the hypothesis cannot be wholly accepted, as the causal mechanism underlying the decrease in average productivity is different from that which Brooks' Law postulates.

Before we move on to a synthesis of the empirical results (presented in chapters 6, 7 and 8) in chapter 10, in the next chapter we turn to the historical evolution of FreeBSD's governance structure, looking at how the project attempted to manage expanding scale.

CHAPTER 9: THE EMERGENCE OF GOVERNANCE[123]

INTRODUCTION

As we saw in the literature review in chapter 2, the way Frederick Brooks dealt with the problem of decreasing returns to scale besetting the development of OS/360 at IBM was through a 'fordist' approach (Weber 2004, p. 60). That is not meant to be a denunciation of Brooks' administrative tactics. It is just that the mode of organisation that he devised (adopting a proposal by Harlan Mills) follows the core principle of fordist production: organising in 'surgical teams' means that the software project is split into teams of ten persons, each of which separates the high-level task of architectural design from the low-level task of code implementation and achieves work coordination hierarchically through supervisors, who hold all decision making authority. Surgical teams presuppose a clear distinction between order-givers and order-takers, between those who design the software and those who implement it: a chief programmer assigns tasks to his subordinates, supervises their performance and coordinates their work. Although this organisational configuration, as Brooks himself recognises, is but an imperfect solution to the problems inherent in increased scale, it however succeeds in drastically narrowing the scope for potential interpersonal communication interactions that lead to decreasing returns to scale (Brooks 1995, chapter 3; see also section **The productivity paradox in software development** in chapter 2).

Such surgical teams are hard to come by in free and open source software (FOSS) projects. The voluntary character of participation in them as well as the self-selection of tasks by participants in accordance with their own desires, implies that they must use a different approach to tackle the issue of decreasing returns to scale (Weber 2004, p. 62). In FreeBSD specifically, as shown by the empirical analyses in chapters 7 and 8, modularity not only offsets but actually reverses Brooks' Law so that increasing group size results in boosting core developers performance. We qualified this finding by arguing that large groups enable a more extensive division

123 A modified version of this chapter was published in the first issue (Jul. 2012) of the *Journal of Peer Production* under the title *Authority in Peer Production: The Emergence of Governance in the FreeBSD Project*.

of labour within the modules they develop, by virtue of which core developers can focus on their task of choice, namely, new code development. Although this interpretation throws light on the effect of increasing group size at the component level on core developers performance, it does not go very far in elucidating the effect of expanding scale on the organisational structure of the project as a whole. Modularity is only part of the answer. As Steven Weber (2004, p. 65) says, 'if Brooks is even partially right....then the success of open source [projects]...depends also, and crucially, on how those people are organized'. The question is, how did the FreeBSD project manage expanding scale? Even better: *what modifications did expanding scale cause the organisation of the FreeBSD project?*

INFORMAL GOVERNANCE PHASE (1993-2000)

As we have seen (in chapter 4) FreeBSD evolved for the first seven years (1993-2000) without any formal means of representing its contributors in project governance.[124] During this period, which we refer to as *informal governance phase*, 'those who hacked most became part of the "core group" or "core team"' (Lehey 2002). Jordan Hubbard, one of the three FreeBSD founders, served as the project's president until 1997, which position was 'originally created...to give ISVs and other corporate contacts a more official-sounding person to talk to'. In 1997 he resigned from the position which he also abolished, claiming that it had created 'the illusion of a "super core member"... [and] false expectations of authority' (Hubbard 1997). The growth of the project was continuous throughout this period. Three concurrent and interrelated empirical phenomena – the growth of peripheral contributors without commit rights to the project (by 2000, there were an estimated 1105 individuals in the periphery of the project [FreeBSD 2000a]), the expansion of the (*src*) committers' group from 16 to 138 persons and the growth of the codebase – attest to the project's growth and the dramatic expansion of scale underway. This period of growth was however accompanied by a growing criticism of the project's governance system. Many committers felt that the composition of the core team no longer reflected merit in the project and the core team was censured for abusing its

124 We employ the term *governance* to refer to 'the use of institutions, structures of authority and even collaboration to allocate resources and coordinate or control activity' (Bell 2002) in the project. Our employment is akin to that used in international relations, as 'in that context, "governance" is not government, it is typically not authoritative, and in fact it is not about governing in a traditional sense as much as it is about setting parameters for voluntary relationships among autonomous parties' (Weber 2004, p. 172).

authority for its own interest. In 2000 dissent could no longer be channelled into a manageable form of mediation with the core team. When a prominent committer entered into a confrontation with a core team member, accusing him of trampling on his changes, the situation spiralled out of control, threatening to tear the project apart. In the discussion that ensued on the project mailing lists, Hubbard outlined a number of possible reforms, including the dismantling of the core team, and called for a vote. The proposal was well received by the base of committers, who elected by vote to adopt an elected core team model. In consequence of this decision, core bylaws were drafted, providing thus a regulatory framework for the operation of FreeBSD's democracy.[125]

The course of FreeBSD's institutional evolution is also reflected in a series of documents which the project released with a view to imparting structure to what was until then largely an informal development process. The first version of the *Committer's Guide* (FreeBSD 1999), which laid down guidelines for regulating committers' mode of conduct, was published in 1999 amidst a climate of rising discontent with the project's governance structure. The first version of the *FreeBSD Developers' Handbook* followed in August 2000 – a month before the first core team election – with information geared to new committers about circumnavigating FreeBSD's development model.

In sum, conflicts over the distribution of authority in the project and concerns of a perceived illegitimacy in its exercise by the core team led to the adoption of an elected core team model in 2000. This institutional restructuring along with the bylaws drafted to regulate elections created a democratic basis of legitimacy for the

125 For the *core bylaws*, see *Table 4.1* in chapter 4. Crucially, the *core bylaws* do not make up what is normally understood by the term *constitution*: they specify the mode of elections and the duration of the incumbency, but unlike a constitution they make no reference to the principles on which the core team shall be established, the manner in which it shall be organised or the powers it shall have, save for establishing the right of committers to recall the core team by triggering an early election. Some of those questions are dealt with in other documents released by the project. For example, *The FreeBSD Committers' Big List of Rules* clarifies that the authority of the core team is restricted to the task of managing commit privileges: 'In all other aspects of project operation, core is a subset of committers and is bound by the same rules. Just because someone is in core this does not mean that they have special dispensation to step outside any of the lines painted here; core's "special powers" only kick in when it acts as a group, not on an individual basis. As individuals, the core team members are all committers first and core second' (FreeBSD 2011d). On the whole, questions related to the distribution of authority in the project were – and still are – the epicentre of *conflict*: for instance, the reason why decisions are made by consensus does not lie in some formal rule forbidding the core team from making decisions autocratically, but in the vigorous resistance of committers against core team decisions they regard as conflicting with their own will (Lehey 2002).

authority of the core team. Closely related with this reform was the parallel attempt to more elaborately define the scope of development activities, crystallised in the release of the first version of the Committer's Guide in 1999 which elucidated the process through which changes are integrated in the repository and outlined committers' behavioural code.

DEMOCRATIC GOVERNANCE PHASE (2000-TO DATE)

The first core team election by vote in September 2000 ushered in the next phase in the institutional evolution of the project, that of *democratic governance*. The transition from a self-selected group of veteran committers to an elected one reinforced the already extant tendency toward the systematisation of rules and development procedures.

Indicative of the ongoing systematisation of rules and procedures is that increasingly more instructions and development procedures are being written down as shown by the continuous updates of the FreeBSD Handbook, the Committer's Guide and the Developers' Handbook. More interesting, for the purposes of our analysis, is that this process is closely connected with the exigencies of conflict management. No example illustrates this better than the SMP conflict in 2002 between two developers which led to the formulation of a policy for suspending commit rights. In February 2002 a conflict erupted over changes made by a committer to the SMP[126] module without the permission of John Baldwin, SMP's most active then-developer. The core team intervened immediately asking him to remove his changes from the repository under the threat of revoking his commit privileges. He complied and asked the core team to resolve the issue. The core team, after a month of discussion and consultation with committers on project mailing lists, decided to delegate authority to Baldwin to approve or reject changes to the SMP code as he saw fit. The core team then used the experience to formulate rules for suspending commit rights, thereby creating a standard discipline procedure with set offences and penalties.[127]

Although the SMP conflict in 2002 was particularly difficult to resolve, it was by no means the only one. The transition to the elected core team model, though it appeased concerns of an illegitimacy in the distribution of authority in the project,

126 The goal of the SMP project was to introduce parallelism into the kernel so that FreeBSD could be run on multiprocessor computer hardware architectures.

127 For the rules governing suspension of commit rights, see *Table 4.2* in chapter 4.

did not eradicate conflicts. A case in point is the conflict in 2003 between the core team and Matt Dillon, a prolific committer, which led to the revocation of the latter's commit rights. According to the explanation given by two members of the then-core team, Warner Losh and Greg Lehey, on a popular online discussion forum for hackers, this decision was dictated by social, rather than technological, considerations: Dillon had repeatedly violated FreeBSD's code of conduct: his behaviour clashed with the collective way of doing things. One of the two core members justified the sanction as follows:

> This action was taken due to Matt's interdeveloper relation skills, not due to Matt's technical skills. FreeBSD has a code of conduct between developers that the core team is charged with enforcing. Without going into the details of this mess, at Matt Dillon's prior request, Matt violated the code many times over the years. Core tried to bring him into compliance with this code of conduct. After a recent incident, core felt that his future compliance would not be sufficient. His failure to comply to that code was causing damage to the project in excess of his contribution. Core felt it had no choice but to remove his commit bit for the good of the project (Slashdot 2003).

A few months later, Dillon announced his decision to *fork* FreeBSD – that is, to make a copy of the codebase and start independent development – thus creating an alternative project called DragonFly BSD (Dillon 2003). Dillon, for his part, claimed that the reason to launch DragonFly BSD was not his admittedly strained relations with FreeBSD committers. Instead, he cited reasons of difference of opinion concerning the technical direction of FreeBSD, emphasising its SMP implementation (Biancuzzi 2004). The case of Dillon's 'ostracism' illustrates clearly two things. First, in a community where technical decisions are intimately related to strongly held values and beliefs about effective ways to organise development, it is very difficult, and sometimes impossible, to distinguish personal from technical conflicts (Mateos-Garcia & Steinmueller 2008; Weber 2004, p. 88). In a sense, both dimensions are lurking under a conflict. It is hardly inconceivable that a personal antipathy or rivalry may manifest under the guise of a technical disagreement. And

conversely, a disagreement on a technical issue, if it is not amenable to immediate resolution, is likely to evolve into a personal conflict, given the tendency inherent in volunteer organisations' collective decision-making structures to personalise the ideas that members of the organisation espouse (Rothschild-Whitt 1979, p. 521). But crucially, as Dillon's case demonstrates, the freedom to fork a project (which FOSS licenses ensure) mitigates the potential for conflicts. Organisation theorists know full well that easy access to the exit option dampens the emergence of conflicts: the potential for conflicts in a group is drastically reduced when members can easily walk out, disengaging themselves from it (Hirschman 1970). The practice of forking is nothing but an extreme example of the exit option: in this way, disputes over the direction of technical change in the project that do not admit of resolution are effectively 'translated' into alternative development lines (FreeBSD core team interview by Loli-Gueru 2003).

The phase of democratic governance – just as that before it – is marked by rapid growth. The massive expansion of scale is illustrated from the increase of (*src*) committers from 138 in 2000 to 209 in 2005.[128] Although the expansion of scale brought about a significant increase of coordination costs, the increased need of active coordination within the group did not lead to the introduction of direct supervision, that is, to an internal hierarchy where contributions are processed upstream through 'gatekeepers'. Rather, it prompted changes in the direction of increased *standardisation*: namely, the standardisation of committers' recruitment process and of outputs through frequent building (Holck & Jørgensen 2003/4; Jørgensen 2007).

A standard argument of organisation theory is that work coordination in a small group may well be informal, based on the mutual adjustment of group members. However, as the group gets larger, it becomes less able to coordinate informally. Thus, control of the work passes into a single individual and direct supervision becomes the chief means of coordination (Mintzberg 1993, p. 7; Perrow 1976). Such a transformation in the mode of work coordination occurred in Linux when, in consequence of the dramatic increase of contributors, Linus Torvalds, the project leader, delegated authority to a cadre of subsystem maintainers – his so-called trusted lieutenants – to filter the contributions of the community of Linux developers. Thus, patches have to be reviewed by the trusted lieutenants, who feed the ones they approve of back to Torvalds for inclusion in the official release (Corbet et al. 2010, pp. 15-17; Moody 2001). In the case of FreeBSD, in spite of the

128 By 2010 the total number of committers had increased to 388 (FreeBSD 2010d).

dramatic expansion of the base of committers, the project made no attempt to introduce direct supervision in order to coordinate their work. It did not attempt to supervise their work process. Rather, it resorted to standardising their *skills*.[129] That was done by standardising the process through which outside contributors are inducted into the project. *Table 9.1* outlines the four stages that make up the recruitment process:

1. A committer proposes to core team to grant commit rights to an outside contributor.
2. The core team approves the proposal, investing the new committer with commit privileges.
3. The mentorship period formally begins: the new committer is assigned a mentor to supervise his work, typically the same committer who proposed that he be given commit rights. In parallel, the mentee has to perform several tasks intended to familiarise him with the tools used by committers and the process by which they integrate changes into the repository. Crucially, for as long as the mentorship goes on, the new committer cannot integrate any changes without the approval of his mentor.
4. The new committer is officially 'released' by his mentor.

Table 9.1: FreeBSD committers recruitment process

The process consists of the following stages which are universally applicable to new committers: first, a committer proposes to the core team to grant commit rights to an outside contributor, based on the latter's history of contributions.[130] As Lucas (2002) says, 'by the time you've submitted several dozen PRs, you'll either work well with the FreeBSD team or everyone will understand that you and the team just can't get along. Direct-commit access is either an obvious next step, or an obviously bad move'. Typically, the committer who vouches for a new member becomes his mentor, assuming responsibility for everything his protégé does in the project. The mentor is in a sense his supervisor: he is responsible for reviewing and approving his changes prior to being committed to the repository. Concurrently, the new committer has to perform a series of tasks intended to familiarise him with the tools committers use and the process through which they integrate changes into

129 '*Skills* are *standardized* when the kind of training required to perform the work is specified' (Mintzberg 1993, p. 6).

130 This part of the process has been formalised since 2002: the FreeBSD website outlines the exact steps would-be mentors must follow to propose a new committer (FreeBSD 2011c).

the repository. The mentorship period has no specific duration and ends when the mentor 'releases' officially the new committer. By that time, the new committer is supposed to have developed a strong grasp of project goals and mastered the requisite technical and interpersonal skills (Lucas 2002).

The standardisation of the recruitment process is designed to harmonise the coexistence of highly independent individuals within the developer community and ensure they can work smoothly with each other by reducing the scope of conflicts related to the integration of changes (Watson 2006). To achieve this, FreeBSD has evolved a training procedure akin to the institution of apprenticeship that builds into the committers-to-be the work programs and the bases of coordination. Thus, on the job they appear to be acting autonomously, just as a surgeon and an anaesthesiologist need hardly communicate when they meet in the operating room, knowing through their training exactly what to expect from each other. This procedure clearly cultivates a *homogeneity of values* to facilitate work coordination and ensure community cohesion: it uses homogeneity as a mechanism for social control.[131] Seen from this standpoint, the recruitment process for new committers constitutes an integral component of the project's governance structure. By ensuring that the conduct of new committers is compatible with the collective way of doing things and with the goals and values of the project, the recruitment process effectively reproduces the structural properties of the FreeBSD social system.[132]

To reduce the need for active coordination, FreeBSD resorted not only to standardising the skills of new committers through the recruitment process but also *outputs through frequent building* (Holck & Jørgensen 2003/4; Jørgensen 2005, 2007). Doing a software build refers to the process of converting human-readable source code into executable code that can be run on a computer. A successful build therefore implies that a working version of the software can be 'built' from the evolving codebase. Aside from the obvious benefit of testing whether the evolving

131 Organisations which 'generally refuse to legitimate the use of centralized authority...to achieve social control', commonly resort to such a 'selection for homogeneity', as shown by Rothschild-Whitt's (1979, pp. 513-4) classic study of five collectivist work organisations in California. This homogeneity is, of course, reinforced by the *self-selection* characteristic of participation in collectivist organisations (Mansbridge 1977, p. 336).

132 The same may be said of the recruitment process in other large FOSS projects. Indicatively, Mateos-Garcia and Steinmueller's (2008, p. 337) study of Debian demonstrates the role of the process by which the project selects and trains new members as a control mechanism due to the homogeneity of attitudes it instils into them. In connection with the standardisation of the process by which new maintainers are inducted into Debian, see also Garzarelli and Galoppini (2003).

product is kept in a working state, software companies do frequent builds to facilitate team coordination: 'the key idea is that one large team can work like many small teams if developers synchronize their work through frequent “builds” and periodic “stabilizations” of the product' (Cusumano & Selby 1997, p. 262). FOSS projects are not an exception (Krill 2011). FreeBSD uses three so-called Tinderbox servers that automatically build the most recent version of the software every few hours.[133] The results are posted on the web and on project mailing lists, notifying committers of 'tinderbox failures'. Keeping committers informed of 'broken builds' is focal to the project's use of mailing lists. As FreeBSD committer David Schultz says with regard to the -current mailing list,

> most people who track -CURRENT are subscribed to current@ precisely because they want to know when things break (quoted in Andrews 2008).

The feedback provided by broken builds is extremely important: committers see the effect of the most recent changes and so can pinpoint which change is responsible for breaking the build. In this sense, the practice of doing daily builds makes the development process more visible. It makes it also more predictable by allowing committers to follow closely their progress in developing new features. Thus, frequent building constitutes a code control mechanism that allows committers to stay in sync with the evolving product.

As broken builds result in halting further development until the bug responsible for the breakdown is found and fixed, a key rule for committers is to make no changes that cause the build to fail. According to the *FreeBSD Committers' Big List of Rules*, to make sure that changes checked in do not break the build, committers must test their changes before they commit them:

> If your changes are to the kernel, make sure you can still compile [the kernel]...If your changes are anywhere else, make sure you can still make world[134] (FreeBSD 2011d).

133 The results of the daily build process are accessible online at <http://tinderbox.freebsd.org>. Indicatively, on 21 June 2011, tinderbox machines performed builds of the *-current* version and of six officially released versions of FreeBSD on nine different hardware platforms.

134 'Make world' refers to updating the FreeBSD base system by using a command known by that name.

This rule, as Holck and Jørgensen (2003/2004) correctly remark, by specifying a criterion of performance that the work of committers is required to meet, achieves the standardisation of the results of their work. Compliance with the rule reduces the need for active coordination among committers, as 'with outputs standardized, the coordination among tasks is predetermined, as in the book bindery that knows that the pages it receives from one place will fit perfectly into the covers it receives from another' (Mintzberg 1993, p. 6). Similarly, FreeBSD committers coordinate with each other in terms of certain performance standards. They are expected to commit changes that do not break the build; how they do this is their own business.

In order to facilitate work coordination and more effectively accommodate increased scale, the project proceeded to a series of further measures. First, in 2001 it started using *quarterly status reports* to alleviate problems of information overload caused by increasing group size. As the first of these reports stated, 'the FreeBSD developer community has grown, and the rate of both mailing list traffic and tree modifications has increased, making it difficult even for the most dedicated developer to remain on top of all the work going on in the tree...[The] Status Report attempts to address this problem' (FreeBSD 2001b). Thus, quarterly status reports have since served to give contributors an overview of the various development activities in progress. Second, from 2003 onwards increasingly more development activity migrated from CVS to Perforce and later on to the Subversion revision control environment because of those environments' superior support for parallel development. By 2006 Perforce had replaced CVS as the development site of experimental features, while the Subversion server is where development work on the *src* tree is currently taking place (FreeBSD 2011a; Long 2010; Watson 2006).[135] Third, the work of different (groups of) committers was to a certain extent decoupled by organising the development of important new features as independent sub-projects with their own project manager (Holck & Jørgensen 2003/4, p. 46). In this way, experimental features are developed in a Perforce

135 The migration of development activity in FreeBSD from CVS to Subversion and Perforce parallels the migration of Linux from Bitkeeper to Git in 2005 (see Shankland 2005). Both cases suggest that the larger a project grows, the more it needs tools that allow developers to work in parallel whilst keeping them coordinated. In fact, it is difficult to overemphasise the importance of version control systems like CVS and Subversion for distributed software development: FOSS developers use the logs recorded in them as activity traces to more effectively coordinate their work without having to engage in discursive (direct) communication. In this respect, version control systems constitute a means of 'stigmergic coordination': a medium through which FOSS developers influence the behaviour of each other by leaving traces of their activity in the artefacts they produce and use in their work (den Besten et al. 2008; Bolici et al. 2009).

revision control environment and merged into the main repository only when they are mature enough (Long 2010). Fourth, the project placed a great importance upon developer events, encouraging its contributors to attend them. In fact, one of the activities for which the FreeBSD Foundation was explicitly set up in 2000 is event sponsorship.[136]

In this period roles and responsibilities are increasingly decoupled from individual committers and delegated to teams. In the informal governance phase, to take one example, one person – Satoshi Asami, known as 'Mr Ports' among FreeBSD developers – was responsible for the entire ports collection. In 2001, he was replaced by the Ports Management Team.[137] Similarly, the position of Security Officer expanded into the Security Officer Team in 2002.[138] Whereas FreeBSD machines were administered in the first phase by two or three persons, an admin team was formed for this purpose in the latter phase. In the informal governance phase, public relations were entrusted to one person – the FreeBSD president – who was responsible for interfacing with corporate contacts. Following the abolition of the presidential position in 1997, the task was picked up by the marketing team and, since its founding in 2000, by the FreeBSD Foundation. Every change we have enumerated so far – from the systematisation of rules and procedures to the formation of administrative teams charged with tasks formerly carried out by just one person – attests that there is a contingent relationship between the governance structure and the scale and maturity of a FOSS project (de Laat 2007; O'Mahony & Ferraro 2007, p. 1101; Mateos-Garcia & Steinmueller 2008).

THE IMPERATIVE OF AUTONOMY

Although the adoption of the elective principle altered substantially the mode of project governance, it did not affect the mode of work organisation of committers in the development process. The process by which changes are integrated in the

136 According to one of the project founders, developer events contribute to relationship-building and effective conflict management: 'meeting face to face is almost always a much better way of building bridges since potentially sensitive topics can be discussed without someone going ballistic at a mis-parsed phrase or an attempted joke which fell flat, and you'd be amazed at how conflicts which have burned for months can be suddenly and easily resolved with one short 30 minute talk over a cup of coffee' (Hubbard 1998a). Apparently, developer events help instil a sense of community, even in a group that relies predominantly upon virtual channels of communication for its activities.

137 The Ports Management Team currently numbers eight members.

138 The Security Officer Team currently numbers eleven members.

repository remained the same. Its main feature – the ability of committers to integrate changes directly to the repository – did not change. Given the dramatic increase of participating committers over time, that is counter-intuitive. How is it possible that the expansion of scale did not result in a hierarchical structure where changes are processed upstream through a series of gatekeepers?

What more than anything else explains FreeBSD's structure of organisation of daily work with change integration is contributors' autonomy. Over time, the will to autonomy has become institutionalised in the governance structure of the development process. Admittedly, as 'FreeBSD development is based on initiative' (Saers 2005), FreeBSD developers are highly autonomous. The development of new features is not done at the behest of the core team but springs from the individual initiative of developers.[139] The core team has no authority to tell developers what to do.[140] Their autonomy of action is most clearly seen in their ability to integrate changes directly into the repository (Jørgensen 2007). This contrasts sharply with software engineering models long established in the commercial software industry, which require that changes be documented and cleared through a chain of superordinates prior to being integrated into the repository (Jørgensen 2007, pp. 119-120; Saers 2005).

What accounts for the autonomy FreeBSD committers enjoy? In the first place, developers come to work in FreeBSD because it offers them substantial control over their work. In a survey of seventy-two FreeBSD committers, more than 80% of them said they were encouraged by the freedom to commit code directly to the repository: 'It is frequently easier to make a change to the code base directly than to explain the change so someone else can do it'; 'I don't feel I am under the whim of a single person' or 'I have submitted fixes to other projects and been ignored. That was no fun at all' (Jørgensen 2005, p. 233; see also Jørgensen 2001, 2007).

Bearing in mind FreeBSD's historical background, the significance its developers attribute to their autonomy of action is hardly surprising. Though its roots go back to AT&T's Bell Telephone Labs (BTL), Unix was developed in a consciously informal fashion. When AT&T withdrew from the Multics project – a joint attempt by BTL, General Electrics and MIT to create a multi-user operating system – some

139 The self-selection of tasks is by no means limited to FreeBSD. As one of the founders of the Apache Project remarks, 'the creative energy needed to solve a particular problem, redesign a piece of the system, or fix a given bug is almost always contributed by individual volunteers working on their own, for their own purposes, and not at the behest of the group' (Fielding 1999, p. 42).

140 To quote former core team member Lehey (2002): 'the FreeBSD project is a volunteer organization, so the core team does not have a mandate to tell anybody to do anything'.

BTL employees took it upon themselves to create their own without any support from their employer. And so was Unix born. Its development was from the beginning autonomous from BTL, dispensing with its leadership and supervision altogether. This, however, served to strengthen the feeling of mutual solidarity among the growing number of users at american universities, who contributed a plethora of enhancements, turning thus the development of Unix into a truly collaborative enterprise (Pfaffenberger 1996; Raymond 2003; Ritchie 1984; Salus 1994). With the passage of time, the disdain for bureaucracy evinced in Unix's early years of development was imprinted upon the Unix philosophy's emphasis on rapid prototyping instead of planning (Gancarz 1995). The subsequent development of BSD at the Berkeley campus of the University of California similarly shunned bureaucratic principles of organisation, pioneering a model which revolved around a group of programmers called committers on account of their power to make changes to the codebase:

> The committers were a group of people we trusted to commit stuff...The notion was that you didn't have all these autocratic controls...we didn't need to tell people not to do that; we didn't have to administratively keep them from doing things they shouldn't be doing. We had set up a culture as well as a structure (McKusick quoted in Leonard [2000]).

In addition to animating the development of Unix and BSD, the principle of autonomy is focal to the model of Internet governance evolved by the hacker community. The prototype of this model is the Internet Engineering Task Force (IETF): the closest thing there is to an institution responsible for the development of Internet standards. Formed in 1986, IETF is an association of volunteers organised into more than a hundred working groups, which propose standards through an open publication process. Formal membership is not required: 'Any individual who participates in an IETF mailing list or attends an IETF meeting can be said to be an IETF member...There are no specific criteria for membership other than to note that people and not organizations or companies are members of the IETF' (Bradner 1999). Its founding belief, as put forth by David Clark, sums up the IETF process of developing standards:

> We reject kings, presidents and voting. We believe in rough consensus and running code (Clark 1992).

Deciding whether to adopt or reject a standard through *rough consensus* means that while unanimity is not required, 'strongly held objections must be debated until most people are satisfied that these objections are wrong' (Hoffman 2010). In practice, though there is no fixed percentage, most proposals that are accepted have the support of no less than 90% of the working group (Bradner 1999). Similarly in FreeBSD, as FreeBSD committer Joseph Koshy (2010) says, 'formal specifications and design documents are seldom used...Clear and well-written code and well-written change logs are used in their place. FreeBSD development happens by "rough consensus and running code"'. In an oft-quoted passage, IETF member Brian Carpenter (1996) claims that making decisions by rough consensus and running code is dictated by the evolutionary and decentralised character of Internet development itself: 'Fortunately, nobody owns the Internet, there is no centralized control, and nobody can turn it off. Its evolution depends on rough consensus about technical proposals, and on running code. Engineering feed-back from real implementations is more important than any architectural principles'. In parallel, making decisions in this way is designed to ensure that the actions taken by IETF will not contravene some of the most distinctive values of hacker culture, such as those which emphasise the importance of individual autonomy of action. Autonomy is the overriding ideal of IETF members. As the IETF manual (known as *Tao*) declares:

> IETFers are fiercely independent. It's safe to question opinions and offer alternatives, but don't expect an IETFer to follow orders (Hoffman 2010).

It is in fact not uncommon for the autonomy principle to be enshrined into the 'articles of association' adopted on foundation of a FOSS project. Thus, according to the *Debian Constitution*, 'a person who does not want to do a task which has been delegated or assigned to them does not need to do it' (Debian 1998, 2007). The analysis of the historical and cultural context in which the development of FreeBSD is embedded brings into sharp focus a broader normative standard with reference to which individual hackers act. It shows that the motive of autonomy attributed to the conduct of FreeBSD developers accords with recognised normative patterns.

The freedom they have to commit changes directly to the repository makes sense in terms of accepted norms, as does their imperviousness to taking orders.

According to 80% of the FreeBSD committers surveyed by Jørgensen (2001), the FreeBSD approach to code integration spurs them to contribute, which of course implies that the governance structure of the development process is an important motivating factor (Jørgensen 2005, p. 122).[141] At the same time, this approach to code integration is reckoned to benefit the project. Insofar as developers are encouraged by the ease with which they can add code to the repository, the freedom to commit changes directly allows the project to 'maintain [its] momentum' (Saers 2005). Judging from the volume of changes added to the codebase (see *Fig. 7.4* in chapter 7) and the number of active committers over time, FreeBSD has without doubt managed to keep up its momentum. This effect on the dynamic of development indicates that the manner in which changes are integrated is decisive for project outcome.[142]

The role of autonomy as an organising norm helps to explain why the dramatic increase of committers did not lead to the introduction of direct supervision within the committers group, understood as an internal hierarchy of gatekeepers. A maxim of sociology is that the most stable social structures are those in which the subjective attitudes of participating individuals are directed toward the belief in a legitimate order (Weber 1947, p. 125). The autonomy principle constitutes such a source of legitimacy in the social organisation of FOSS projects (O'Neil 2009, pp. 37-43). The exercise of authority in FOSS projects – as well as its transmutations over time – cannot be understood apart from the influence of the normative standard of autonomy. Under no circumstances is the conduct of bearers of administrative authority – the core team in the case of FreeBSD – allowed to infringe upon developers' autonomy of action, making thus impossible the adoption of organisational configurations which seem to contravene this fundamental principle. Consequently, the fact that the increase of scale in numbers of committers did not result in the hierarchisation of the committers' group is accounted for by committers' motivation with reference to the principle of individual autonomy of action. Had not been for the primacy of autonomy, it remains open to speculation whether the project would have resorted to the standardisation of skills and outputs (through the recruitment process for new

141 The FOSS studies by Ghosh (2005, p. 27), O'Mahony and Ferraro (2007) and Shah (2006, p. 1008) make the same point.

142 Mateos-Garcia and Steinmueller's (2008, p. 335) study of the Debian project arrives at the same conclusion regarding the importance of how contributions are integrated.

committers and frequent building respectively) rather than to a hierarchical reconfiguration of committer relations.

AUTHORITY AND LEGITIMACY

All modes of governance are based on some conception of authority (Harrison 1960). Authority in FreeBSD consists in control of the code repositories: only committers can commit changes and only the core team can grant or revoke commit privileges. However, governance mechanisms cannot be coercive, as this contravenes the autonomy principle animating the work of committers. But if FreeBSD's conception of authority is not based on coercion, then what is it based on?

The answer is *knowledge*. Contributions that reflect extensive knowledge of programming technique and of the goals of the project are rewarded with *reputation*, which gives their authors 'the right to exercise authority over the project and, if not its participants, then at least their contributions' (Mateos-Garcia & Steinmueller 2008, p. 336). The basic criterion for granting commit rights to outside contributors is their technical prowess, acquired and demonstrated through peripheral participation in the project. As core member Robert Watson (2006) notes, committers are recruited on the basis of 'their technical expertise, their history of contribution to the FreeBSD Project, their clear ability to work well in the FreeBSD community'.

The authority of the core team has the same basis: it is based on the recognition of its members' technical competence, acquired and demonstrated through participation in the project. In the first seven years of FreeBSD development, 'those who hacked most became part of the “core group” or “core team”' (Lehey 2002). Although the selection process of core team members was informal, it was ultimately the amount and quality of code one committed which was supposed to serve as the criterion of core team membership. The mode of selection of the core team changed considerably in the subsequent period. The application of the elective principle imparted a democratic legitimacy to the authority of the core team, as its members were now elected by and amongst committers. However, the conception of merit in the project remained anchored in technical competence, proxy-measured by the amount and quality of code one committed.

Max Weber's classic analysis of how authority is legitimised provides a lens through which the historical transformation of FreeBSD's governance structure can

be viewed. According to Weber (1947, pp. 124-125), no authority system is stable unless it is based on the belief of those subject to it in the legitimacy of their subordination.[143] He distinguishes between three types of *legitimate authority (Herrschaft)*. 1. The first type is that of *legal* (or *legal-rational*) authority. In this case, 'obedience is owed to the legally established impersonal order' (Weber 1947, p. 328) so that those subject to legal authority 'owe no personal allegiance to a superordinate and follow his commands only within the restricted sphere in which his jurisdiction is clearly specified' (Giddens 1988, p. 158). Persons in authority typically occupy a 'position' or 'office'. They are not elected to their position but appointed on the basis of their technical qualifications. Their organisation follows the principle of hierarchy: 'each lower office is under the control and supervision of a higher one' (Weber 1947, p. 331). 2. *Traditional* authority is based on the sanctity of age-old rules and powers handed down from the past, such as that which is exercised by village elders in small rural communities. 3. *Charismatic* authority, Weber's third type, is that which is recognised by those subject to it as due to the extraordinary abilities of the leader, 'by virtue of which he is...treated as endowed with supernatural, superhuman, or at least specifically exceptional powers or qualities' (Weber 1947, p. 358). To this type belongs the authority exercised, for example, by prophets and religious leaders over their followers or by heroes in war. The claim to legitimacy in charismatic authority is founded upon the belief in the authenticity of the leader's mission. The charismatic leader supplies proof of his uniqueness through his prodigious feats: the prophet has to perform miracles and the war hero triumphant military exploits. But while these are signs of the validity of the leader's authority, they are not as such the basis upon which it rests, which 'lies rather in the conception that it is the *dut*y of those who have been called to a charismatic mission to recognize its quality and to act accordingly' (Weber 1947, p. 359).[144] Groups subject to charismatic authority are typically based on an 'emotional form of communal relationship' (Weber 1947, p. 360). Their administration is not carried out by 'officials' but by the leader's followers or disciples who share in his charisma. There is no such thing as career or promotion, no salary, no benefice. There is only a 'call', a 'mission' or 'spiritual duty': the leader's administrative staff is summoned to the charismatic mission. There is no hierarchy: the leader merely intervenes when he considers the members of his staff inadequate to the tasks they

143 By formulating this sociological maxim, Weber subscribes to the view – as old as political theory itself – that rule by force, as opposed to rule by persuasion, is illegitimate.

144 As Weber (1947, pp. 359-360) explains, 'no prophet has ever regarded his quality as dependent on the attitudes of the masses toward him'.

have been entrusted with. There is no system of formal rules or precedents handed down from the past: 'the genuine prophet, like the genuine military leader and every true leader in this sense, preaches, creates, or demands *new* obligations' (Weber 1947, p. 361).

As a general rule, FOSS projects 'are created with few traditions to guide them and so do not inherit a traditional basis of authority' (O'Mahony & Ferraro 2007, p. 1081). They do not rely upon a legal-rational basis of authority either, as there is no authoritative division of labour. But when authority cannot be validated through tradition or hierarchy, its justification turns often on the charisma of its bearers. The leadership of Unix had, without doubt, a charismatic character during its early development at AT&T. From its inception in 1969 until the mid-70s, the development of Unix is closely connected with the names of Ken Thompson and Dennis Ritchie. In recognition of their important role in the making of Unix, they both have risen to mythical status in hacker folklore. The FOSS literature has the tendency to present them as individuals endowed with extraordinary abilities (e.g. Raymond 2000). The same charismatic qualities are also attributed to their successor Bill Joy, who spearheaded the subsequent development of Unix at Berkeley from 1977 until 1982. As one of his Berkeley colleagues describes him: 'He had an infectious enthusiasm about him, where he would just get the people around him to do stuff' (McKusick quoted in Leonard [2000]).

The rule of charisma is however ephemeral. Because of its disdain for the routine and the everyday, it is impossible for charisma to survive unless it undergoes a profound modification. Its 'routinisation' therefore implies the devolution of charismatic authority. It is not hard to discern the occurrence of this transformation in the course of BSD development. The project already counted more than five years of development by the time Joy stepped down in 1982. In the wake of his departure, Sam Leffler – Joy's second-in-command – took over the responsibility of completing the release of 4.2BSD. But because 'he was not appointed to Joy's post and felt slighted by this' (Salus 1994), he soon left for Lucasfilm.[145] Following the release of 4.2BSD in August 1983, Leffler was replaced by another member of the team of programmers working on BSD at Berkeley (known since 1980 as the Computer Science Research Group or CSRG for short), Mike Karels, who was joined by Kirk McKusick in December 1984. Under their leadership, the project evolved an organisational structure with a core team at the

145 Currently, Leffler is a FreeBSD committer and a member of the FreeBSD Foundation's board of directors.

centre and a wider base of committers surrounding it (Leonard 2000). The type of authority relationship that emerges from the routinisation of charisma, according to Weber, is determined in large part by how the succession problem is resolved. In the case of BSD, the successor was not nominated by the predecessor. Nor was he self-selected: in spite of his professed willingness to take on the leader's role, Leffler was not appointed to this position by the CSRG and soon stepped down. On the contrary, the fitness of his substitute for the position, Mike Karels, as well as that of Kirk McKusick who joined Karels a year later, was validated through his designation by the CSRG. The issue of succession was not raised again in BSD. With Karels and McKusick as project coordinators, a two-tier organisational structure began to take shape in which leadership, rather than being vested in a single person, was entrusted to a self-selected group of heavily involved developers. This set the stage for an important reinterpretation of the charismatic principle. Instead of being restricted to the person of the project leader, the 'gift of grace' was extended to a leading cadre of hardcore developers.[146] FreeBSD inherited this conception of quasi-charismatic authority from BSD along with its organisational template.

When the FreeBSD project was launched in 1992, the core team included 13 individuals: the last three coordinators of the 'unofficial 386BSD patchkit' plus its most then-active developers. The development of FreeBSD was – and still is – based on a group of programmers who are called committers because of their ability to make changes to the codebase. Committers organised themselves as an *informal meritocracy*: the most active committers were invited by the core team to join its ranks and outside contributors who regularly sent useful patches were offered commit rights. Granting commit rights to an outside contributor amounted to recognition of the technical expertise that his patches demonstrated. In the same way, inviting a committer to join the core team reflected the recognition of his outstanding contribution to FreeBSD and brilliance in coding. Authority was derived from technical competence, acquired and demonstrated through participation in the project.

Although the conception of merit in the project did not change (nor was the notion that legitimate authority derives from technical competence ever questioned), the criticism of the selection process of the core team and of its prerogatives became more virulent over time. Its thrust was, on the one hand, that

146 Weber recognised that 'it is possible for any type of authority to be deprived of its monocratic character, which bonds it to a single person, by the principle of collegiality' (Weber 1947, p. 392).

the core team had degenerated into a gerontocracy of veteran FreeBSD developers which no longer reflected merit in the project and, on the other, that members of the core team abused their power to serve their own ends. When in 2000 a prominent committer announced his intention to quit the project because a core team member was trampling over his work, the criticism of the core team turned to an open conflict that rapidly took on alarming proportions. The intervention of one of the project founders at this point was of decisive importance. He suggested a number of alternative reforms and called on committers to vote. They responded to his call, deciding by vote to adopt an elected core team model. Core bylaws were drafted shortly thereafter to regulate elections.

The transformation of charisma set off by the application of the elective principle to the core team selection was in this case fuelled by the rupture between the group of committers and the core team. The conflict that manifested itself through the growing criticism of the distribution of authority in the project brought about a shift in project governance toward an electoral process for the selection of the core team. As a result, the core team, whose legitimacy rested on its members' charisma, then became the core team thanks to the confidence of committers. The introduction of elected core team members entailed a radical alteration in their position: they became the 'servants' of those under their authority. The passage of leadership from a self-selected group to a freely elected one signified that from now on committers were free to elevate to power as well as depose as they pleased. Whereas the recognition of the charisma of the core team was so far perceived by committers as a consequence of its legitimacy, it now began to be considered as its basis. Legitimacy was in this sense democratised.

The reconfiguration of the governance system brought about by the transformation of charisma limited the authority of the core team in four important ways. First, the sphere of its authority was circumscribed: the role of the core team was restricted to managing commit privileges and mediating in the event there is a serious disagreement between committers. Second, it exerted control on the core team that made it accountable to the community of committers: the core team is required to defer to their wishes, making only decisions that reflect the consensus of the opinions of committers as manifest on mailing lists. Third, its term of office was specified: new elections would be held every two years. Fourth, project leadership became revocable: the core bylaws invested committers with the power to trigger an early election, recalling thus the core team. All these traits correspond to the type of governance Weber calls *direct democracy*: the short term of office,

the liability to recall, the restricted sphere of jurisdiction, the obligation to render an accounting to the general community of committers as well as submit to it every important question (Weber 1947, pp. 412-3). Direct democracy is characteristic of groups which, in order to preserve their members' autonomy, attempt 'to dispense with leadership altogether' by reducing 'to a minimum the control of some men over others' (Weber 1947, p. 389). In that sense, direct-democratic forms of governance are, according to Weber, inherently *anti-authoritarian*. The routinisation of charisma in groups like FreeBSD which champion their members' autonomy is most likely to follow such a line of development.[147]

In FreeBSD, more specifically, the anti-authoritarian transformation of charisma that culminated in the adoption of a direct-democratic mode of governance limited the authority of the core team through the introduction of elements of democratic and legal-rational rule. The principles of consensus-oriented decision making, the limited duration of office and the liability to recall are all institutional safeguards drawing their justification from the sovereignty of the will of committers. The premises for delimiting the authority of the core team by specifying its sphere of jurisdiction are, on the other hand, bureaucratic par excellence. Authority in a bureaucratic organisation is characterised by 'specificity of function'[148]: authority is distributed and legitimised only within the particular sphere of the office.[149] The authority of the core team is likewise restricted to a specific field: it can be exercised only in matters touching commit rights and committer disputes. 'In all other aspects of project operation, core is a subset of committers and is bound by the *same rules*' (FreeBSD 2011d). The use of hats within the project – that is, of assigning clearly circumscribed areas of responsibility to certain committers – is also indicative of a stripped-down, embryonic form of bureaucratisation as is the tendency toward the formation of teams that take on the role formerly held by a single committer (e.g. Ports Management and Security Officer teams).

147 In the Debian project, for example, conflicts between the project leader and the community of maintainers over what was perceived as a lack of legitimacy of the leader's authority led to the drafting of the Debian Constitution and the development of the new maintainer process through which new members are inducted into the project. The former acts as a check upon the authority of the project leader while the latter has the purpose of ensuring that new recruits possess not only the right skills but also views which are consistent with the socio-political goals of the project (Mateos-Garcia & Steinmueller 2008, p. 239; O'Mahony & Ferraro 2007; O'Neil 2009, chapters 7 and 9).

148 Although Parsons (1939, p. 460) employs the term to describe the scope of professional authority, specificity of function is as much characteristic of professional as of bureaucratic authority.

149 A 'specified sphere of competence', as Weber (1947, p. 330) calls it, that involves the obligation to perform definite functions is a fundamental category of bureaucratic authority.

Weber (1947, p. 390) remarked that 'the anti-authoritarian direction of the transformation of charisma normally leads into the path of rationality', as the setting up of an administrative organ that functions reliably invariably involves the systematisation of rules and procedures, fuelling thus the progressive bureaucratisation of the group. Yet the authority of the core team does not belong to the bureaucratic type. If bureaucracy is understood as a 'clearly defined hierarchy of offices', as Weber (1947, p. 333) defines it, then core team members are not bureaucratic types. Since there are no officers on the core team, core team members are not integrated in a hierarchical order: they have no superiors who influence their promotion to the core team or supervise their activity (cf. Weber 1947, p. 387).[150] In contrast to bureaucratic organisations which motivate their members through remunerative incentives, participation in FreeBSD (and in FOSS projects more generally) is voluntary and unwaged. Although a good many committers are professionals working in the IT industry,[151] their involvement in FreeBSD cannot be considered as a career, as conventionally understood. For there is no career advancement in FreeBSD: outside contributors may well become committers and committers core team members, but that is hardly analogous to moving up in a multi-layered hierarchy of ranks.[152] In fact, the aim of FreeBSD's governance system is to eliminate the division of labour that separates decision making labour (administrative tasks) from executive labour (performance tasks). Not only is the core team, in addition to its managerial duties, expected to be producing code, but more crucially decision making rests on a consensus process in which all project members can participate. For decisions to be taken as binding and

150 At the first and only physical meeting of the first elected core team (at the end of BSDCon in Monterey in 2000), it was decided that 'there will be no officers on the core team' (Lehey 2002).

151 Indicatively, in a survey of 72 FreeBSD committers (constituting 35% of all committers) conducted in 2000, '43% said an employer had paid for all or part of their time spent on their latest code contribution' (Jørgensen 2001).

152 If, following Blau (1970, p. 203), we take differentials of status and managerial rank to be the sociological criteria on the basis of which organisational members are differentiated, then we come to the conclusion that FreeBSD consists of two hierarchical levels. The first hierarchical level signifies that the contributions of outside contributors are evaluated by committers, who alone have the right to integrate changes to the codebase; the same applies to new committers whose changes must be approved by their mentor prior to being integrated in the codebase. The second level indicates that the right of committers to integrate changes is subject to the core team's approval. By comparison, Microsoft, taken as an example of software development in a proprietary/commercial context, 'has established “ladder levels” for each specialty, represented by formal numerical rankings (starting from 9 or 10 for college graduates and going to 13, 14, or 15, depending on the area' (Cusumano & Selby 1997, p. 116). The higher echelons of management are not even included on this scale (Ibid., p. 119).

legitimate, they must carry the consensus of the group behind them. To ensure that committers can participate in the process of formulating problems and negotiating decisions, all issues are discussed on project mailing lists. Judging from how decisions are made in the project, FreeBSD is not a bureaucratic but a *collectivist* organisation.[153] The 'consensus rule' applies not only to decisions made by the core team but to all decisions in the project, including the integration of changes into the codebase. It is indicative that, according to the *FreeBSD Committers' Big List of Rules* (FreeBSD 2011d), no change should be committed to the repository unless 'something resembling consensus has been reached'. Consensus, in this case, is reached by asking for community review: committers are advised to announce their proposed changes on the mailing lists and ask for community review before they commit them. Although this process does not generate a large amount of feedback,[154] its significance its clear: as source code modifications, especially non-trivial ones, are equivalent to important decisions, it is crucial that their implementation receives the consensual backing of project members. The criterion of consensus indicates that decision making is not hierarchical but collective: authority resides in the collective as a whole rather than in the 'superordinate'. Such consensus-oriented forms of decision making are obviously incompatible with bureaucratic forms of organisation.

What, according to Weber, differentiates bureaucracy from other forms of organisation is that it allows for regular control of operations over time. That is what, in his view, makes bureaucracy 'rational': through the use of double-entry book keeping, bureaucracy makes continuous capital accounting – the evaluation and assessment of profit-making opportunities – possible over long periods of time. Double-entry book keeping, in a sense, 'stacks' past events and anticipates future

153 In fact, it is not only by virtue of how decisions are made that FreeBSD can be characterised as collectivist. The collective character of decision making is mirrored in the project's ownership structure. Aside from programmers' time and effort, the most important resource in the project – the software produced by its members – is collectively owned by FreeBSD developers. However, FreeBSD differs from prototypical collectivist organisations in that, unlike them, only some of the instruments used in its development process are owned by the collectivity as a whole. FreeBSD owns the IT infrastructure that project contributors use to communicate (public and non-public mailing lists, IRC channels), to integrate changes in the codebase (CVS, Perforce, Subversion), to test the evolving software product (Tinderboxes) and monitor product defects (GNATS), to release and distribute software (CVSup) and to publish information (website). But the tools with which FreeBSD developers write code are their own private property: their own Internet-enabled personal computers.

154 In a survey of 72 FreeBSD committers (constituting 35% of all committers) conducted in 2000, 86% mentioned that they received feedback from two or more reviewers (Jørgensen 2001).

ones, thereby providing the basis for organising collective activities on a stable and continuous footing. Such control of time is integral to bureaucracy: the central role of the 'files' in a bureaucracy lies precisely in enabling a continuous and regular operation. But control of space is equally indispensable to the functioning of bureaucracy: Weber is emphatic that administrative discipline is most effective when the vocational life of the officials is strictly separate from their private life. The type of demarcation of activities thus effected stretches not only across time but also across space: hence the locale of the office must be separate from the domicile of the official. According to Foucault (1975), whose work deals more extensively with the theme of time-space control, the distinguishing feature of bureaucratic organisation – whether in schools, barracks, factories or hospitals – is that the use of an individual's time and space is constantly monitored and controlled. In such organisations, every individual is assigned its 'proper place' and has certain duties to perform at any particular moment. The detailed management of individual activities makes it possible to link every movement of the body with the performance of a specific task. This type of administrative authority, Foucault says, connects discipline directly with utility: its goal is to ensure that the use of an individual's time is channelled solely into those activities that the administrators consider useful. By contrast, participation in the development of FreeBSD is not subject to such forms of control. The project does not keep any record of the time individual committers spend on it. Committers participate in their free time, deciding themselves when they will work and for how long. Moreover, their geographical location is irrelevant: they may work on FreeBSD from the privacy of their homes or from any other place. As seen from the standpoint of time-space control, FreeBSD wholly dispenses with the 'discipline' characteristic of bureaucratic administration: no attempt has ever been made in the project to supervise the individual activities of committers or control with any means the use of their time or space.

The divergence of FreeBSD from the bureaucratic model can also be illustrated from the form of social relationships in the project. While social relations in bureaucratic organisations are based on the formal roles held by their members as laid down by an authoritative division of labour, relationships between FreeBSD developers are far more holistic, affective and personal.[155] For committers, FreeBSD

155 One may wonder how is it possible that developer relations in FreeBSD are personal, given that their interactions occur predominantly in a computer environment. After all, long distance relationships seem rather impoverished, if not shallow, compared to relationships that are based on physical co-presence. It is instructive in this connection to refer to the emphasis Marshall

is a *community*; a fraternity of peers, so to speak. While bureaucratic organisation separates the 'official' from the 'personal', these two dimensions fuse together in the ideal of community that FreeBSD aspires to (O'Neil 2009, p. 175). In Weberian terms, the orientation of social action in FreeBSD is value-rational: that is, social conduct is based on definite moral values. The actions of individuals are directed to an overriding ideal: being part of the hacker community that coalesces around the development of the FreeBSD operating system (cf. Torvalds 1998). That is not to say that their actions are not informed by pragmatic considerations, chiefly that they want the fruits of their labour to be used by as many people as possible (Hubbard 1998b).[156] But relationships between people in FreeBSD – as is typical of collectively-run volunteer organisations (Rothschild-Whitt 1979, p. 514) – are seen as of value in themselves. Arguably, it is not on account of holding some office that core team members are recognised as figures of authority. Although their opinion may well carry more weight in discussions occurring on project mailing lists than that of other committers, this influence is not the result of their 'powers of office' but rather of the respect and trust given them by committers for their substantial contribution to the project. In collectivist organisations, as Rothschild-Whitt (1979, p. 524) remarks, 'because authority resides in the collectivity as a unit, the exercise of influence depends less on positional opportunities and more on the personal attributes of the individual'. Prior studies have shown that collectivist organisations find such inequalities of influence 'acceptable in circumstances in which those who exercise power exercise it in the interests of others (usually because their interests are identical with those of others)' (Mansbridge 1977, p. 326). This interpretation fits FreeBSD nicely: committers accept that some of them exert more influence than others because that influence is reckoned to be aligned with their own interests. Some traces of charismatic authority can still be detected in this type of relationship: the trust of committers in core team members is, to a certain extent, of an emotional type; and the persuasive authority of core team members is legitimised through the recognition of the authenticity of their technical charisma

McLuhan (1964) laid on how the diffusion of electronic telecommunications would transform the globe into a 'global village', signalling the return of humanity to a tribalesque form of sociality. For McLuhan, the effect of telematic technology on social interaction is profound: as its scope is no longer determined by geographical proximity but by affinity, it becomes possible for relations of a more remote kind to be experienced as meaningful and personal.

156 Jordan Hubbard (1998b), one of the project founders, characteristically underlines the role of pragmatic considerations in the development of FreeBSD: 'Our principal objective is to see that our software gets used by anyone who can conceivably find a need for it, and we don't care whether that need is commercial or not'.

by committers.

For Weber, the transition from the autocratic selection of the core team to its democratic election by vote would signal the end of charismatic rule, as its subjection to norms and rules invariably involves the loss of genuine charismatic authority. Charisma abhors permanent forms of organisation and formal rules. Its claim to legitimacy lies in 'the conception that it is the *duty*' of those subject to charismatic authority to recognise its uniqueness and act accordingly (Weber 1947, pp. 359-60). This conception of authority is no longer representative of FreeBSD. The election of the core team by and amongst committers resulted in changing the basis of its legitimacy. The recognition of charisma is no longer treated by committers as a consequence of the legitimacy of authority but as the basis upon which it rests. While legitimacy formerly rested on the 'duty' of committers to recognise the technical charisma of the core team, it became democratic in the latter period with the application of the elective principle: the authority of the core team was no longer validated by the charisma of its members but by the will of committers. Legitimacy was thus 'democratised'.

The routinisation of charisma in FreeBSD resulted in a direct-democratic governance system in which the distribution of authority is validated by the will of committers. Although that form of governance includes elements of bureaucratic authority, as the authority of the core team is delimited by mechanisms that to some extent reinforce bureaucratic values (such as the functional specificity of authority), its source of legitimacy is fundamentally democratic: it is justified by the imperative to preserve the sovereignty of the committers' will rather than by its adherence to an impersonal hierarchical order. It is important to observe that the transformation of charismatic to democratic authority did not modify the conception of merit in the project, which remains anchored in technical competence, acquired and demonstrated through project participation. What changed markedly however is the conception of leadership: leadership is no longer conceptualised as the informal rule of a self-selected group of heavily involved committers, but as a democratically elected group of committers that is revocable and subject to formal rules.

CONCLUDING REMARKS

We analysed FreeBSD's course of institutional evolution by distinguishing two phases, based on their corresponding mode of governance. Whereas from 1993 until

2000 FreeBSD had no formal means of representing its contributors in project governance and leadership consisted in a self-selected group of veteran committers, in 2000 the growing criticism of the distribution of authority in the project brought about a shift toward an elected model, according to which project leadership is exercised by nine persons elected biennially by and amongst committers. Considering the dramatic increase of committers over time, the transformation of the FreeBSD governance system – as well as the systematisation of rules and procedures that runs parallel to it – suggests that a project's governance structure is contingent upon its scale and maturity. The transformation of the governance system, however, did not affect the mode of work organisation of committers in the development process, in spite of the remarkable expansion of scale.

While organisation theory predicts that as a group grows larger it becomes less able to organise informally and so is compelled to turn to supervisory hierarchy as a means of coordination, the expansion of the committers group was not accompanied by changes in that direction. Rather, the project resorted to standardising (a) the recruitment process for new committers and (b) outputs through frequent building. This line of development cannot be understood apart from the influence of the normative standard of individual autonomy of action: it can be accounted for only by bearing into mind that an important reason why hackers are attracted to FreeBSD is the freedom of committers to add changes directly to the repository. The centrality of the autonomy principle elucidates the intervening motivational link between the observed activity – the course of action FreeBSD took to manage increased scale and achieve work coordination within an expanding group – and its meaning to the actors involved. A basic principle of the hacker ethic is to 'mistrust authority – promote decentralization' (Levy 1984). Hackers espouse the view that the ultimate effect of centralised authority is to strangle the creative potential inherent in self-regulating individuals, thus acting as a check upon their free development. As the activities of hackers are driven by an acute sense of independence, it is not conceivable that they would adopt organisational configurations which contravene their autonomy.

The normative significance of individual autonomy explains why authority in FOSS projects cannot be coercive. Authority in this environment, as Benkler says (2006, p. 105), 'is persuasive, not legal or technical, and certainly not determinative'. Naturally, that is not to say that no authority exists. In FreeBSD it specifically consists in control of the ability to make changes to the codebase. Considering that no authority relationship is stable unless it is recognised by those

who submit to it as based on some legitimate order (Weber 1947), we examined how authority is legitimised in FreeBSD, contrasting it with Weber's categories of legitimate authority. We found that legitimacy shifted from the quasi-charismatic authority of a self-selected group of heavily involved committers to the democratic authority of an elected group that is revocable and bound to formal rules.

However, none of Weber's categories captures sufficiently the character of authority in FreeBSD. If, following Weber (1947, p. 152), authority is defined as a relationship in which an actor obeys a specific command issued by another, then FreeBSD is essentially an *organisation without authority*. There is no such thing as giving or following orders in FreeBSD. The administrative organ of the project – the core team – cannot tell committers what to do. When a decision needs to be made, it is made collectively by consensus. If, in the Weberian tradition, we take the basis of authority as the decisive organisational feature, then the mode of organisation of FreeBSD is collectivist, based on direct-democratic procedures of decision making. Seen from the perspective of the division of labour in the project, the mode of organisation of FreeBSD is decentralised and anti-hierarchical: tasks are self-selected by committers as their needs and interests best dictate. The resulting division of labour is spontaneous in the sense that it emerges from the choices of the committers rather than from a central designer. Committers work without supervision, shouldering themselves the ultimate responsibility that the modifications they make to the codebase have been adequately tested and do not clash with the work of other committers. Consequently, FreeBSD illustrates 'a production process that doesn't rely on managers' (Hamel 2007, p. 208). In FreeBSD those who work also manage.

The next chapter sums up the empirical results of the research and reflects on the role of modular product design as a governance mechanism.

CHAPTER 10: CONCLUSIONS

SUMMARY REVIEW OF RESULTS

The results arrived at by testing hypotheses *H1*, *H2*, *H2R*, *H3* and *H4* in chapters 5, 6, 7 and 8 of this dissertation are summarised in the following six tables:

#	Subject	Independent variables	Statistical instrument	N	Test results	Verdict
H1	**Effect of modularity on coordination**	-	Descriptive statistics	N=Raw dataset	No evidence found in 3 instances	**Not confirmed**

Table 10.1: Summary of statistical test results and findings for H1

#	Subject	Independent variables	Statistical instrument	N	Test results *committers*	Verdict
H2	**Effect of modularity on group size**	propagation_cost_lag integrality_index_lag	Regression analysis	N=242	sig = 0.000 p = 0.227 sig = 0.000 p = 0.026	**Accepted**

Table 10.2: Summary of statistical test results and findings for H2

#	Subject	Independent variables	Statistical instrument	N small-scale/ large-scale	Test results small-scale, *integrality_index*	Test results large-scale, *integrality_index*	Verdict
H2R	**Effect of group size on modularity**	committers	Regression analysis	N=148/123	sig = 0.724 p = 0.725	sig = 0.025 p = 0.025	**Accepted**

Table 10.3: Summary of statistical test results and findings for H2R

#	Subject	Independent variables	Statistical instrument	N small-scale/large-scale	Test results small-scale, *Δ_KB_per_committer / Δ_LOC_per_committer / top_2_committers / top_10percent*	Test results large-scale, *Δ_KB_per_committer / Δ_LOC_per_committer / top_2_committers / top_10percent*	Verdict
H3	**Effect of modularity on productivity**	integrality_index_lag	Regression analysis	N=121/121	sig = 0.948 p = 0.948 / sig = 0.767 p = 0.768 / sig = 0.074 p = 0.278 / sig = 0.540 p = 0.628	sig = 0.042 p = 0.043 / sig = 0.001 p = 0.002 / sig = 0.001 p = 0.002 / sig = 0.009 p = 0.023	**Accepted**

Table 10.4: Summary of statistical test results and findings for H3

#	Subject	Independent variables	Statistical instrument	Test results *Δ_KB_per_committer / Δ_LOC_per_committer*	Test results *top_2_committers / top_10percent*	Verdict
H4	**Effect of group size on productivity**	committers	Regression analysis	(N=277) sig = 0.014 p = 0.014 / (N=277) sig = 0.000 p = 0.000	(N=257) sig = 0.000 p = 0.000 / (N=280) sig = 0.000 p = 0.000	**Accepted with qualifications**

Table 10.5a: Summary of statistical test results and findings for H4

#	Subject	Independent variables	Statistical instrument	Test results small-scale, *top_2_committers* / *top_10percent*	Test results large-scale, *top_2_committers* / *top_10percent*	Verdict
H4	**Effect of group size on productivity**	committers	Regression analysis	(N=280) sig = 0.000 p = 0.000 / (N=157) sig = 0.000 p = 0.000	(N=123) sig = 0.000 p = 0.000 / (N=123) sig = 0.000 p = 0.000	**Accepted with qualifications**

Table 10.5b: Summary of statistical test results and findings for H4

Let us now attempt to synthesise the above findings.

EFFECT OF PRODUCT STRUCTURE ON GROUP DYNAMICS

Decentralisation made scalable

As theorised by Sanchez and Mahoney (1996), product modularity imparts scalability to production systems whose key feature is the radical decentralisation of productive activities. That is presumed to be accomplished by decoupling production tasks so they can be tackled independently by autonomous development groups. In short, Sanchez and Mahoney's theory holds that product modularity *makes decentralisation viable on a large scale*. The dimension of size is crucial. While product modularity may not be necessary to a small-scale, though decentralised, software project in which participants are in position to grasp and keep track of the interactions between distinct product components and by extension among the persons or groups working on them, that is by no means the case for large-scale, geographically distributed software projects. With the enlargement of scale comes an emphasis on the design parameters meant to encourage and underpin decentralised production – namely, modular product design.

The findings of our investigation bear this out, highlighting the similarity between the design of technological and organisational systems. As we have seen in chapter 6 when testing *H2*, higher levels of modularity at the component level result in larger development groups. This, of course, implies that relationships between product components are analogous to relationships between developers, providing thus support for Sanchez and Mahoney's (1996, p. 64) contention that '*products design organizations* because the coordination tasks implicit in specific product designs largely determine the feasible organizational designs for developing and producing those products'.

Furthermore, as chapter 6 illustrates, the historical expansion of the FreeBSD committers' base is paralleled by increasing levels of modularity at the component-level. In consideration of the (geographically and functionally) distributed character of the development process (described in chapters 4 and 9), the above finding gives some empirical flesh to the claim that 'the modular architecture of software design enables a decentralized production' by mitigating the need for active coordination between distinct tasks (Osterloh & Rota 2007, p. 159). Although decentralisation may be seen in some quarters as a safe-guard against the arbitrariness of bureaucratic authority (Levy 1984), its more tangible contribution to large free and open source software (FOSS) projects lies in the strategic flexibility with which it invests the production system, thereby enhancing its absorptive capacity: the project can be scaled up (by taking on more tasks or by adding more persons to work on a task, regardless of their geographical whereabouts) while retaining the flexibility typical of smaller organisational configurations. A precondition for the flexibility that decentralisation affords to a production system, however, is the decomposability of the product into loosely-coupled components. Otherwise, an attempt to enlarge the scale of a decentralised production system would necessitate considerable active coordination, owing to the difficulty of managing interdependencies between distinct product components. It is precisely because it mitigates the need for active coordination between product components that modular product design fosters decentralisation.

In the case of FreeBSD, the phenomenal increase of (*src*) committers over time from 16 to about 200, given (a) their geographical dispersal over the world and (b) that 'most of the development work takes place in one-man projects' (Holck & Jørgensen 2003/2004, p. 42; Jørgensen 2001; Jørgensen 2005, pp. 231-232) as developers are working largely by themselves (as noted in chapter 4), is a strong indication that the scope of decentralisation of production has been broadened. The

escalation of decentralisation or what amounts to the same in this setting, the capacity of the project to absorb that many more developers is due, in large part, to the increase in modularity at the component-level. Without the ancillary role of modular product design, the expansion of the base of committers would be attended by such increased coordination costs that the product development process would become bogged down. That helps explain the importance the project lays on periodical architectural re-designs: every new development branch of FreeBSD involves an extensive architectural clean-up intended to remove inter-dependencies (FreeBSD committer Wes Peters interviewed by Loli-Gueru 2003). From this standpoint, to borrow Sanchez and Mahoney's formulation, modular product design is a device by which to enhance a production system's strategic flexibility and absorptive capacity.

Modularity reinforces the emergent division of labour

By dissecting in chapter 7 the results of the regression of core developers'[157] production output[158] on our independent variables (see *Tables 7.6, 7.6, 7.8*), we ascertained that an increase of modularity at the component-level occasions a rise in the output of (that component's) core developers, *provided that conditions of large-scale development prevail*.[159] In order to more fully comprehend the relation of product structure to core developers' performance, it is necessary to take one more factor into account: the learning costs involved in making oneself familiar with the codebase and keeping track of the interactions therein. As these learning costs are determined by the size of the codebase, it is not hard to see that as the scale of the project expands – that is, as more developers join the committers' group and the codebase consequently grows bigger – it becomes increasingly more burdensome for any one of them to grasp the sum total of interactions between the

157 We use the characterisation *core developers* to refer to high-contribution committers, though the FreeBSD project does not use this term on the grounds that it can mislead one to conflate prolific committers with core team members (see FreeBSD committer Greg Lehey's comments in Slashdot 2003).

158 At the level of individual modules, core developers' production output is proxy-measured by the number of code contributions made by the top two committers in each module, and alternatively by the code contributions of the top ten percent of committers in each module.

159 We used the median of committers (*eight*) to distinguish between conditions of large-scale and small-scale development at the component-level. Thus, a small-scale development process is reflected in years that fewer than nine committers participate in the development of a module (i.e. committers < 9), while large-scale development is reflected in years in which more than eight committers are active (i.e. committers > 8).

components comprising the codebase. The only way then that committers can stay on top of development work is by specialising in that part of the codebase with which they are most familiar. Thus, a spontaneous division of labour emerges among them out of their own decision to narrow down the focus of their contributions. This tendency toward specialisation is reinforced by modular product design: enlarging the scale of the project militates in favour of committers' specialisation, to which modular product design conduces by enabling the independent development of distinct product components. Because of that, individual committers need not bother about activities centred on any segment of the codebase other than that which forms the focal point of their work. The reason therefore why an increase in modularity at the component-level results in an increase of core developers' output is because it induces the 'separation of concerns' (Parnas 1972) among committers, reflecting and reinforcing at the same time their own decision to specialise in some one area of the codebase. As such, modular product design is the logical equivalent to the division of labour characteristic of decentralised, large-scale projects: it is the technical expression of the division of labour in the context of decentralised production processes as well as the mechanism through which that division of labour is effected.

Effect of product modularity on labour productivity

In the previous section we examined the positive effect that an increase of modularity at the component-level exerts on the output of core developers when large-scale development conditions prevail. We qualified this result on the basis of the finer division of labour in the project to which modularity conduces, arguing that modularity reinforces the tendency of core developers to specialise in conditions of increasing scale. But what about the effect of modularity on the performance of the committers' group as a whole? The statistical tests we performed focusing on small-scale development conditions (*Table 7.4*) showed no significant effect. On the contrary, by dissecting the results of the tests centred on large-scale development conditions (*Tables 7.2, 7.3*), we found that an increase of modularity causes an increase in average group performance, providing thus empirical support for proposition *H3*, which holds that modularity has a positive effect on labour productivity in projects such as FreeBSD which are characterised by increasing scale.

This result (viz. the positive effect of modularity on group performance) is

explained by modularity theory as follows: a modular product design allows developers of software projects, which are undergoing an expansion of scale, to focus on some one component of the product without having to coordinate their work with that of others working on different components – that is, it allows developers to work independently of each other. Consequently, their individual performance remains as high as if they actually worked by themselves. The statistical analysis (presented in chapter 7) verifies the claimed benefit of modularity, showing that – in large-scale conditions of development – higher levels of modularity at the component-level bring about an increase in average group performance. The reason why the statistical analysis, on the other hand, finds no significant effect of modularity on group performance in small-scale development conditions appears to lie in the increased development costs attendant upon the modularisation process, which erode the claimed productivity benefits of modularity (Capra et al. 2008, p.765; Garud & Kumaraswamy 1995, p. 97; Garzarelli & Galoppini 2003). Viewed this way, that higher levels of modularity do *not* result in an increase in average group performance in small-scale development conditions, as opposed to large-scale ones, implies that in order for the benefits of modularity to exceed its costs, the scale of development has to be so enlarged that the need to mitigate the adverse effects of increasing scale takes on a pressing character – for it is only then that the potential of modularity can be fully exploited.

EFFECT OF GROUP DYNAMICS ON PRODUCT STRUCTURE

Product structure mirrors organisational structure

The notion that the architectural structure of a product mirrors the structure of the social organisation that produced it is not new. As early as 1968, Conway argued that the social relations of production of software artefacts crystallise into their architectural structure, which phenomenon is also attested in the results of the statistical tests conducted in chapter 6. Our findings bear this out in part, indicating that – *to the extent that large-scale development conditions prevail* – an increase in the number of participants in a distributed software development process leads to higher levels of code modularity.

As qualified in prior empirical work, this effect is the corollary of the very mode of production exemplified by large FOSS projects (Capra et al. 2008; MacCormack

et al. 2006; Merlo et al. 2009; Weber 2004). Because FOSS projects are: (a) devoid of the pressure of deadlines characteristic of commercial software development settings and (b) paradigmatic of software production as a public process founded on the openness of source code, it follows that FOSS developers are incentivised to produce software of higher design quality (which, as structure is considered an aspect of design quality, is therefore more modular) than their counterparts in the commercial software industry. From this vantage point, removing the pressure of deadlines from the product development process and exposing one's code to the scrutinising gaze of a multitude of programmers creates a powerful inducement to develop modular code (Capra et al. 2008, p. 778).

Furthermore, as large FOSS projects are (c) typified by a large and geographically distributed base of developers, in consequence of which the scope for face-to-face communications is drastically narrowed, it follows that fewer communication paths between developers – and by implication, fewer connections between components (modules) – are established. Owing to the inherent limitations on communication, therefore, the product architecture that evolves is more modular (MacCormack et al. 2006, p. 1027; MacCormack et al. 2008, pp. 20-21). That the logical equivalent to the decentralisation of production processes is the architectural modularity of the resulting product is also attested in the pattern of industrial growth that the microcomputer industry (better known today as the personal computer industry) has come to epitomise. As chronicled by Langlois (1992) among others, a decisive role in edging personal computers onto a modular path was played by the hobbyist community that bootstrapped the industry in its early days. As established firms of the likes of IBM initially failed to appreciate the market potential for small computers for individual end-users, the early stages in the history of the PC industry are largely the story of enterprising hobbyists who fed on the capabilities of a large network of external sources to develop their own computers (Anderson 1984; Gray 1984; Hauben 1991; Stern 1981). Lacking the technical capabilities for producing in-house all the components they needed to build a personal computer, hobbyists banded together in user-groups (such as the legendary *Homebrew Computer Club* out of which emerged the distinctive culture of high-tech entrepreneurship that Silicon Valley is acclaimed for) and resorted to specialising in some components while outsourcing the rest. Had these hobbyists – and the start-ups they founded – not drawn upon a globally distributed network of capabilities, it would have been impossible to give flesh to their vision of

'computers for the masses'.[160] As Langlois says, 'the rapid growth and development of the microcomputer industry is largely a story of *external economies*. It is a story of the development of capabilities within the context of a decentralized market rather than within large vertically integrated firms' (Langlois 1992, p. 38, emphasis ours; see also Langlois & Robertson 1992, p. 311). The architecture that evolved for personal computers was therefore modular not because of any inherent technological necessity, but because the personal computer was a systemic product made up of components that were developed independently of one another by different firms, with little, if any, active coordination between them (Langlois 1992, p. 39).

The above analysis furnishes ample information in order to qualify the results of the statistical tests pertaining to conditions of small-scale development. If an increase of group size up to eight committers prompts no changes in the product structure in the direction of increased modularity, that is because of the extent they coordinate their work with each other; because the pattern of work in the group – when participation is limited to fewer than nine persons – is essentially that of a close-knit group. The product structure that evolves is non-modular, therefore, because it reflects the work patterns of a tightly-coupled group of developers.

Product structure as coordination mechanism

Considering that FOSS projects are (d) destitute of recourse to an authority structure by which to effect coordination, it appears that product structure constitutes an essential coordination mechanism in this setting (Merlo et al. 2009, p. 35). In contradistinction to commercial software development environments where coordination is effected through the organisational hierarchy, as decisions made 'at the higher hierarchical levels are addressed and structured through the middle levels and implemented by the low-level development teams', FOSS projects have

> no explicit and formal organizational structure...the network is highly dynamic and team members are likely to change, even within the core. As a consequence, the

160 The Apple II (1982) illustrates this well: its stuffed boards were developed by GTC; its floppy-drives from Shugart and Alps; its hard-drives from Seagate; its RAM and ROM chips from Mostek, Synertek and NEC; its monitor from Sanyo. The only components that Apple developed in-house were floppy and hard-drive controllers, the power-supply and the case. See Langlois (1992, pp. 14-15, footnote 44).

> only coordination mechanism that can be effectively exploited is the software product [structure] itself, which becomes the only way to share design decisions over time and coordinate tasks among the community members (Merlo et al. 2009, p. 16).

It is a well known principle of organisation theory that 'organizations, through the authority mechanism, provide a means for *coordinating* the activities of groups of individuals' (Simon 1991, p. 38). By contrast, the predominantly voluntary character of participation in FOSS projects dispossesses project managers (who, in the context of FOSS projects, are better known through a variety of names such as project leaders [like Linus Torvalds in Linux or Stefano Zacchiroli in Debian], administrators [in Sourceforge-hosted projects], module owners [in Mozilla], subsystem maintainers aka 'trusted lieutenants' [in Linux] or the core team [in Apache and FreeBSD]) of the means by which to command-and-control. At the same time, FOSS developers are keenly aware that their informal and voluntary division of labour is incomplete, if not fragile. The coordination issues raised by this problem are at the forefront of their discussions and product architecture is designed accordingly (Weber 2004, p. 175). That is to say, FOSS projects have no alternative but to use the product structure as a coordination mechanism. Linus Torvalds' experience with version 2.0 of Linux (which was released in 1996) drove home this lesson (Torvalds 1999), which has since become part of community practice. It is for that reason that software structure is arguably perceived by FOSS developers as a variable that FOSS projects can and must manipulate in order to induce a definite division of labour by reducing the need for active coordination between the product's constituent elements. Admittedly, FOSS developers leverage design structure to induce what David Parnas (1972) calls a definite 'assignment of responsibility': in this sense, product architecture can be said to drive the organisation of FOSS projects (Weber 2004, pp. 174-175).

The above conclusion, however, by emphasising the impact that modifications of the product structure exert on the development organisation, may be seen as contradicting the foregoing analysis, according to which product structure evolves to reflect the production environment embedding it. If product structure can be moulded so as to shape group dynamics, this implies that the modularisation process cannot be conceived apart from the political will that enacts it. From this viewpoint, changing the product structure constitutes an active managerial

intervention. To use an expression of Henry Mintzberg, it is an action taken on the basis of 'deliberate strategic intent': as such, product modularity cannot be said to arise out of the actions of individual developers pursuing their own interests independently of what others are doing on the project, but is imposed from the top. Yet this contradiction is but a seeming one: for as we have seen, in the context of FreeBSD development, modularity is a design parameter meant to reinforce core developers' decision to specialise in some one area of the codebase, which decision is a strategy they deliberately employ to cope with the increased learning costs that a growing codebase implies. The initiative enacting the modularisation process, therefore, seldom emanates from a lead architect or a command centre cut off from the actual site of development activity. Rather, the growth in the size of the codebase impels the realisation that core developers, should they want to stay on top of development work, have to concentrate on that part of the codebase in which they are most experienced. And, of course, with this decision comes an emphasis on the design parameters aimed at encouraging and facilitating core developers' specialisation – namely, modular product design.

Why not in small-scale development conditions?

Contrary to the tests centred on large-scale development conditions, the results obtained by testing the effect of group size on product structure in conditions of small-scale development (see *Table 6.12*) suggest that increasing the number of developers who work on a software project – as long as the overall group of developers remains essentially small (viz. does not exceed eight developers) – exerts no significant effect on product structure. The reason why the enlargement of the group left no mark on the product structure is because the increase in the number of developers was such that no extensive modification of communication patterns was rendered necessary. To the extent that the increase of developers working on the project is moderate, it does not interfere with individual developers' work process: hence, insofar as no modification of the existing communication system is required, an increase in the number of developers is rather unlikely to prompt any changes in the product structure in the direction of increased modularity. Modifications in the product design structure aimed at higher levels of modularity are called forth when the increase in group size is large enough for existing channels of communication to accommodate without at the same time clashing with work patterns. In order for an increase of developers to be imprinted onto the

software artefact, therefore, the group must be so enlarged that it necessitates a radical modification of communication patterns, and by extension, work patterns – which exigency modular redesign meets.

EFFECT OF GROUP SIZE ON LABOUR PRODUCTIVITY

Brooks' Law revisited

The hypothesis known as Brooks' Law holds that adding more developers to a software project brings about a fall in group productivity because of the increased communication and coordination costs attendant upon group expansion. Our analysis of descriptive statistics provides ample support for this proposition, for as we have seen in chapter 8 the historical growth of the committers' base is paralleled by a tendential fall in average labour productivity (measured in both LOC added per committer and KB added per committer). At first glance, observing that average productivity drops concurrently with the rise in the number of participating committers is suggestive of the negative effect of group expansion on productivity that Brooks' Law predicts.

Yet this conjecture may be somewhat premature in light of the effect that increasing group size exerts on the output of core developers. By contrasting the code contributions of core developers (as reflected in the code contributions of the top fifteen committers for every year of development activity) with the total volume of code contributions to the project over time, we ascertained that the output of core developers is not negatively affected by the expansion of the committers' group. From this premise it follows that *the drop in average productivity is due to the disproportionate increase in 'low-contribution' committers* over time. This of course implies that either core developers' work process, in spite of the expansion of the committers' base, is not subject to increased coordination costs or they invest increasingly more time in the project so that the time they channel in communicating and coordinating their activities with one another does not eat away at the time they put in producing code. By showing that the amount of time which their majority (55.5%) spends on the project is steadily increasing, the results of our survey of core developers (discussed in chapter 8) lend support to the latter hypothesis, suggesting that their high performance is not due to the absence of coordination costs but to the temporally increased scope of their participation.

The statistical tests discussed in chapter 8 reinforce the syllogism that the drop

in average productivity is caused by the disproportionate increase of low-contribution committers over time. They show that an increase in group size at the component-level (i.e. at the level of individual modules) occasions a rise in core developers' output. All the more so, they furnish proof that this effect is augmented in large-scale development conditions so that the larger the group that develops a component (module) the larger the output of that component's core developers. This result, though counter-intuitive at first sight, is nonetheless not unaccounted for. On the one hand, increasing the number of persons engaged in the development of a module allows its core developers to introduce a finer division of labour within the boundaries of the module. The larger the group that develops a module the greater the room for task delegation within it. Thus, the ability of a module's core developers to delegate tasks and responsibilities (on a voluntary basis of course) increases in proportion with the number of committers attached to the module, thereby freeing up time for core developers to more fully concentrate on churning out code. Moreover, large development groups excel in generating problem-reports and fixes. This gives structure to the work content of core developers, as it is on the basis of that feedback that they prioritise tasks in the development process. Hence, if increasing the number of committers working on a module results in boosting the productivity of its core developers, that is because large groups enable a more extensive division of labour within the modules they develop, by virtue of which core developers can focus on their task of choice, namely, new code development.

It is important to note, however, that these results do not falsify the basic premise of the Brooks' Law hypothesis: there is no doubt that interpersonal communication paths or interactions, which can lead to decreasing returns to scale, grow exponentially as more developers join the development process of a module. It is therefore unavoidable that the need for active coordination becomes the more pressing as more individuals are added to the development group. Hence, the increased productivity of core developers in modules with large groups is by no means accounted for by a mitigation of coordination costs within the boundaries of the module. Quite the contrary, the increased performance of core developers in modules jointly developed by large groups needs to be explained on the basis of increased coordination costs. It begs the question, how is it possible that core developers produce *more* in the face of increased coordination costs within the boundaries of the module? Apparently, the cognitive difficulties represented by the increased coordination costs are not beyond the ability of those modules' core

developers to handle. Keeping up a high level of individual performance requires that they understand and keep track of interactions between tasks within the module. Of course, the ability of humans to process information is limited, and so is the ability of core developers to manage an ever-growing number of interactions. That is to say, the limit to core developers' productivity is the number of task interactions they can manage. Insofar as the group that develops a module does not grow large enough for interactions to spiral out of core developers' control, they may well sustain a high performance. Conversely, an increase of interactions beyond core developers' capacity of apprehension would most certainly disrupt their work process, lowering their productivity. In the case of FreeBSD, we have not been able to trace such a threshold. Without exception, the statistical tests we performed reveal that the larger the group that develops a module the greater the output of its core developers. Yet, that we found no evidence to the effect that increasing group size drags core developers' productivity down should not be taken as proof that such a threshold does not exist *in potentia*. What we know with certitude is that the threshold has not so far been reached: at the level of individual modules an increase of group size up to forty-five committers – which no FreeBSD module exceeds on any one year of their development[161] – is demonstrably shown to raise core developers' output. However, should the increase of group size at the component-level be greater, it remains an open question whether core developers could sustain their high performance. Crucially, this eventuality is tempered by modules' development dynamic over time: for as our statistical tests in chapter 6 indicate, the size of each module's development group is inversely proportional to the module's production-readiness (i.e. maturity). Put simply, the need for manpower declines in proportion as modules approach maturity. Modules attract more contributors in their early development stages because at that point the number of production tasks to be worked on is much greater – hence, more developers are needed. Conversely, the closer a module approaches production-readiness the fewer the production tasks pending completion – hence, the fewer the developers that are needed. This implies that modules evolve through the successive stages of growth and stagnation, which effectively regulate the relative size of a module's development group. By ensuring that development groups grow

161 It is worth noting that in other FOSS projects some of the modules attract significantly larger development groups. Take Linux for example: in version 2.5.25 of Linux, 3.55% of all modules were developed by groups numbering 51 to 100 developers, while 4.74% of modules had groups working on them that exceeded 100 developers (Ghosh & David 2003). Similarly, modules with more than a hundred developers are quite common in Mozilla (Mockus et al. 2002, p. 334).

as large as modules' technical production requirements allow for, as well as that they grow smaller in proportion as these requirements are fulfilled, this pattern of component evolution suggests that modular product design functions (within the boundaries of the modules) as a mechanism, which modulates the number of developers that can be assigned to work on a module *according to its development stage*.

It is worth repeating that the positive effect of increasing group size on the output of core developers is not accounted for by a mitigation of coordination costs *within* the boundaries of the module. The results of our component-level analysis do not falsify the core premise of the Brooks' Law hypothesis: interpersonal communication paths within the boundaries of a module grow exponentially as more committers join the module's development group. It follows that what moderates the potential for decreasing returns to scale at the component-level is not a mitigation of coordination costs within the boundaries of modules, but the more extensive division of labour that larger development groups make possible. That is to say, the increased output of core developers is explained by the fact that larger development groups enable a finer division of labour within the module they develop, thanks to which core developers can delegate more tasks to other module-developers, thus being able to more fully preoccupy themselves with the development of new code. However, it is not only the finer division of labour within the boundaries of modules that prevents the negative consequences of increasing group size from asserting themselves. A strong moderating effect on the potential for decreasing returns to scale is exerted by the motivational forces at work.

As the discussion in chapter 1 clarifies, besides the exponential growth of interpersonal communication paths, which results from adding more persons to the development group (e.g. Boehm 1981; Brooks 1995), decreasing returns to scale are often the result of *reduced individual motivation*: group performance falls because people tend to expend less effort when working as part of a group (e.g. Ingham et al. 1974; Latané et al. 1979). That is not however an unavoidable consequence of collective work. The tendency for people to expend less effort when working collectively is reduced or eliminated when individual outputs can be evaluated collectively; when one is working on tasks perceived as meaningful and engaging; when a group-level comparison standard exists; when working with friends or in groups one highly values; and when inputs to the collective outcome are (or are perceived as being) indispensable (Karau & Williams 1993; Kerr & Brunn 1983).

That is without doubt the case with FreeBSD: high levels of individual motivation are sustained thanks to the psychological frame of developers. In line with the volunteer character of participation in FOSS projects, the ability to self-select the tasks one is going to work on (according to one's own interests) ensures that tasks are perceived as meaningful and engaging. Furthermore, a common motivation for contributing to a FOSS project is the value placed on being part of the hacker (i.e. FOSS) community. In addition to furthering a sense of community and belonging, the adoption of the hacker identity functions as a motivation for action: the surest way to share in this cultural identity is by affiliating oneself with a FOSS project. As Linus Torvalds (1998) says, 'the act of making Linux freely available wasn't some agonizing decision that I took from thinking long and hard on it: it was a natural decision within the community that I felt I wanted to be a part of'. For FOSS developers, therefore, launching a project or joining an existing one constitutes a core part of what defines them as individuals. Although it is not without reason that the work ethic of FOSS developers has been frequently portrayed as highly individualistic (e.g. Shah 2005, p. 12), this description fails to grasp a fundamental element of shared belief within the hacker community: engaging in cooperative relationships with other hackers is not a constraint upon one's freedom of action, but an enabler for the full development of one's potentialities.[162] As one of the Apache Project founders puts it, 'we collaborate on producing and supporting the Apache server out of enlightened self-interest: by pooling our efforts, the resulting product is much more functional and robust than anything we could have produced alone' (Fielding 1999, p. 42). That is what FreeBSD core team member Robert Watson (2006) alludes to when he says that 'FreeBSD developers are generally characterised by independence [and] a good sense of cooperation'. The fact that in the development of a hacker project (a) tasks are perceived as inherently meaningful and interesting and (b) collaboration is intrinsically motivated as participation is valued in-itself for reasons of cultural identity, ensures that increasing group size does not result in diminishing individual motivation.

As far as the potential for *performance measurement* is concerned, though conventional yardsticks of labour productivity are shunned,[163] FOSS projects are not

162 As Steven Weber (2004, p. 145) explains, 'personal efficacy not only benefits from, but positively requires, a set of cooperative relationships with others. The popular image of an open source hacker as a lone ranger emphasizes the self-reliant attitude that is certainly present but misses the deep way in which that self-reliance is known to be made possible through its embedding in a community. The belief is that the community empowers the individual to help himself'.

163 Whereas labour productivity is typically measured in relation to labour time expended (i.e. number of working hours) or money-wages advanced (in which a definite number of working

destitute of ways by which to evaluate the extent of the contribution of any given individual committer. First of all, the openness and free availability of source code makes it possible for anyone interested to study it and hence evaluate the quality of the work of its author(s). Likewise, in projects giving free (read-)access to their code repositories (e.g. CVS and Subversion in the case of FreeBSD), this can be done by looking at logs of activity traces: one can identify the committers responsible for any piece of code checked into the repository and so evaluate their contribution to the collective output. In the spirit of fostering friendly rivalry between committers, FreeBSD also keeps a record of the number of changes every committer has checked in.[164] The performance of the group as a whole is also amenable to evaluation, as a group-level comparison exists in the form of the other projects descended from the original BSD operating system (i.e. OpenBSD, NetBSD) and the Linux kernel project. In much the same way that one can compare the end-user functionality (i.e. features) of these operating systems with that provided by FreeBSD, their development status may well serve as a benchmark by which the development progress of FreeBSD can be evaluated. Taken together, the ability to evaluate (a) individual committers' contributions to the collective outcome (i.e. any one release of FreeBSD) and (b) the performance of the committers' group as a whole against other FOSS projects' group performance, serves to reinforce the already present high levels of individual motivation.

The final, though no less important, factor that restrains the manifestation of decreasing returns to scale in this production setting consists in the radical departure of FreeBSD from the pattern of scale expansion inherent in conventional organisations. A cause of decreasing returns to scale as prominent as reduced individual motivation springs from the *communication distortions* attendant upon expansions of the scale of production. As Williamson (1967, 1985) elucidates (see chapter 1), this problem is inherent in that form of expansion of scale, universally characteristic of hierarchical organisations, according to which the span of control principle must be strictly adhered to. Because an extra manager must be installed

---

hours are crystallised), that is by no means possible in the realm of FOSS development where participants are volunteers who contribute in their free time. The fact that they receive no remuneration for their contributions, and there is no record of the time they dedicate, dictates an alternative method for the measurement of productivity. For an extensive discussion of alternative methodological approaches to the analysis of economic activity in FOSS projects, see section **Measuring labour productivity** in chapter 3 and Ghosh (2003).

164 The so-called activity tables are accessible online at <http://people.freebsd.org/~peter/commits.html>.

for every *X* number of persons added to the working group,[165] attempts to expand the scale of operations under this principle are afflicted by the ills of increased bureaucracy: the more the successive layers of hierarchy that information has to pass through the greater the potential for serial reproduction loss. In consequence, information (such as reports) sent upward is fragmentary or erroneous, while information (such as instructions) passed downward becomes exceedingly harder to operationalise. The historical growth of the FreeBSD committers' group contrasts sharply with this form of organisational expansion. Not only is maintaining a definite ratio of subordinates per supervisor not a precondition for (nor an after-effect of) expanding the scale of the project, but this logic of organising is totally done away with. The self-selection of tasks by committers – the fact that they define both the process and content of their work – takes the place of hierarchical organisation. A hierarchical structure, especially one with many layers, is evidently impossible when there is no distinction between those who make decisions and those who execute them. In this respect, FreeBSD represents a radical departure from the growth pattern characteristic of organisations modelled on the hierarchical separation of decision-makers from decision-executants. The overlap of decision making and executive labour in the FreeBSD development process obviates the need for a layered hierarchy – which amounts to saying that it creates the conditions under which the span of control principle can be completely disregarded – thereby negating the negative consequences of scale expansion that clinging to this principle entails. The only sense in which a discussion of the span of control principle is meaningful in the context of FreeBSD is with respect to the learning costs attendant upon a growing codebase and the coordination costs involved in integrating an increasing stream of code contributions from the periphery of the project, that is, from outside contributors without commit rights.[166] The growth of the (*src*) committers' group over time can be seen as an adaptation to the increase of outside contributors. For it is this influx of peripheral contributors, largely accounted for by the explosive growth of Internet connectivity in the 1990s (Lehey 2002; Saers 2005),[167] that was accommodated by the expansion of the committers' group.

165 'If any one manager can deal directly with only a limited number of subordinates, then increasing firm size necessarily entails adding hierarchical levels' (Williamson 1985, p. 134).

166 Mateos-Garcia and Steinmueller's (2008, p. 337) study of Debian makes the same point.

167 The massive diffusion of the Internet revolutionised the scope of geographically distributed software development by enabling a far greater number of people than ever before to participate in such projects.

By considering it in this light, the core team's delegation of authority (in the form of commit privileges) to increasingly more outside contributors represents essentially an attempt to address limitations in the 'span of control'. As committers are responsible for integrating the contributions of those without commit rights, the increase of outside contributors in the periphery of the project made it necessary to bolster the ranks of the committers with more persons.

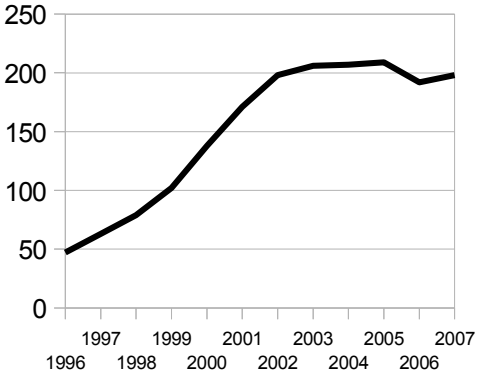


Fig. 10.1a: Committers (*src*)

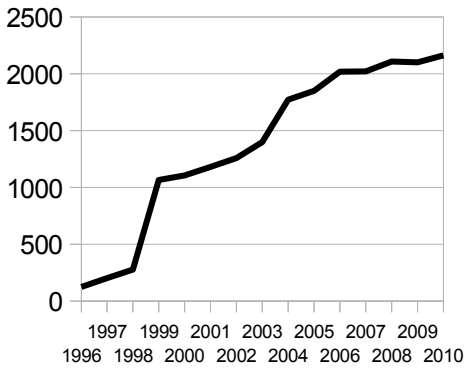


Fig. 10.1b : Peripheral contributors

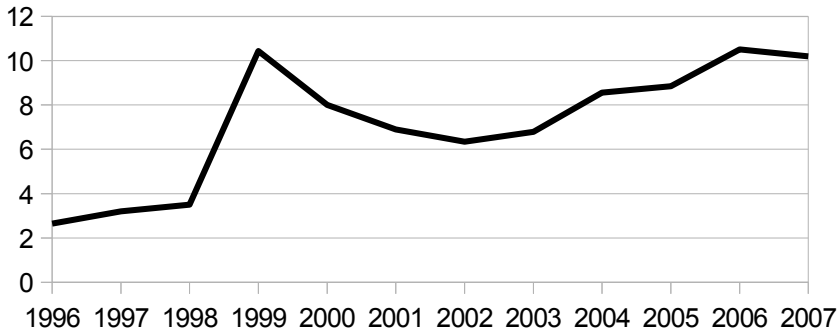


Fig. 10.2: Number of peripheral contributors per committer

In the same way, the growth of the codebase entailed such learning costs that it was no longer possible for the dozen or so[168] committers comprising the core team to stay on top of development work across all areas of the project. This perspective on how the learning costs attendant upon a growing codebase and the coordination

168 The core team had thirteen members in 1993; since 2000 it has nine.

costs involved in integrating an increasing stream of peripheral contributions trigger changes in the size and composition of the committers' group sheds light on the emergence of the FreeBSD governance structure. It suggests that 'strategies for integrating [peripheral] contributions play an essential part in determining the outcomes of a project' (Mateos-Garcia & Steinmueller 2008, p. 335) and that a principle of organisation, seemingly analogous in function to that of the span of control in hierarchical organisations, is operant, regulating the number of committers in relation to that of contributors without commit rights.

There is however a crucial difference. The span of control principle refers to the number of subordinates per supervisor. In FreeBSD there is no such thing as a structure of subordination. The core team can suspend committers' privileges or expel them from the project, but cannot tell them what to do, how or when. Similarly, committers may choose not to integrate into the repository what outside contributors send them, but that is as far as their authority over them goes.

To recap, the historical expansion of the FreeBSD group of committers is accompanied by a fall in average group productivity, seemingly confirming Brooks' prognosis. However, this decrease in productivity, as our analyses show, is owed to the disproportionate increase of low-contribution committers, rather than a fall in the output of core developers. From the perspective of core developers' performance, therefore, the FreeBSD project manages to elude the negative consequences of increasing group size. The reason for this, however, does not lie in the mitigation of coordination costs within the boundaries of the modules but in the finer division of labour that larger groups enable within the modules they develop, thereby allowing their core developers to more fully concentrate upon producing code. In parallel, the potential for decreasing returns to scale is blunted by the motivational forces at play in this setting: when one works in a group one highly values, as is the case in FreeBSD (and the general rule with FOSS projects), performing only such tasks as one's own interests dictate, increasing group size has no longer a demotivating effect. Similarly, the ability to evaluate individual contributions to the collective work product as well as the performance of the group as a whole by comparing it against the activity of groups working on projects of similar scope and functionality, reinforces the motive to contribute. Furthermore, the pattern of scale expansion encountered in the context of FreeBSD development has contributed to averting the manifestation of decreasing returns to scale. In contrast to conventional organisations in which expanding the scale of operations presupposes a proportional increase of layers of hierarchy, FreeBSD has

so far faced no such constraints: it has been possible to expand the scale of the project without necessitating a taller hierarchy. Flat hierarchies, such as that encountered in FreeBSD, limit the potential for serial reproduction loss, ensuring thus that there is no loss in the quality of information communicated among project contributors.

GENERALISABILITY

Across community of FOSS projects

At this point it makes sense to ask whether the conclusions we drew from our study of FreeBSD are valid for other FOSS projects. Compared to other FOSS projects, FreeBSD differs in several respects:

(1) It is a large project with hundreds of contributors, as opposed to the majority of FOSS projects whose development is carried out by small groups (Krishnamurthy 2002).
(2) It is one of the oldest and most mature FOSS projects, being developed since 1993. Its longevity implies that the pattern of group interactions has crystallised into an organisational structure. By contrast, young projects are driven by groups that have not yet settled into a definite set of organising principles, goals and processes to perform their activities.

So, compared to FreeBSD, FOSS projects that are either small or at an early stage of development are arguably less likely to try to use the product structure (modularity) as a means of coordination but more open to experimentation with alternative organisational configurations. However, the characteristics common to all FOSS projects far exceed in importance their individual differences:

(1) Their development is not collocated but distributed, thriving on the contributions of a loosely-coupled community of programmers scattered around the world.
(2) Their mode of organisation is non-hierarchical. Participation is open, volunteer and based on the self-selection of tasks, as participants choose tasks to perform in accordance with their own desires.
(3) The hacker ethic constitutes the common normative standard for FOSS developers.

In consequence of the geographical diaspora of their developers and their non-hierarchical mode of organisation, FOSS projects, regardless of their scale or maturity, are receptive to the idea of using the product structure as a means of coordination. But of course, the need to do so becomes acutely felt and pressing only when the scale of the project is expanding. With that caveat in mind, the case of FreeBSD is representative of the function of modularity in FOSS projects undergoing an expansion of scale. It is also very unlikely that FreeBSD is an outlier with respect to the effect of increased scale on project governance. To the extent that the hacker ethic constitutes the normative standard to which the conduct of FOSS developers conforms, an increase of project scale is unlikely to prompt changes in an authoritarian direction, as FOSS developers are averse to heavy-handed control and organisational configurations that seem to contravene their individual autonomy of action. Equally important, the environment in which FOSS projects operate does not provide the objective conditions for the emergence of hierarchy: a structure of subordination cannot develop in an environment characterised chiefly by the massive participation of volunteer contributors who (a) are geographically dispersed, (b) can easily exit and (c) are not bound by relations of economic dependency. Given the commonalities we have just remarked, there are legitimate grounds to believe that the results we obtained from our study of FreeBSD apply to FOSS projects in general. But is it possible to generalise from the case of FreeBSD to draw conclusions that hold in organisations other than FOSS projects?

Beyond the realm of FOSS

That should only be attempted with great caution, considering the distance separating FOSS and conventional organisations. In asking whether the results we obtained by testing the effect (a) of modularity on group dynamics and (b) of scale on project governance in FreeBSD are likely to hold in other organisational contexts, we are faced with several difficulties, of which the most important is that the form of organisation and management of FOSS projects is fundamentally different from that encountered in other work environments.

As we have seen in FreeBSD, FOSS project administrators do not have the authority to give orders and tasks are not allocated through an authoritative division of labour but self-selected by participants in accordance with their own

wishes. The resulting division of labour is spontaneous in the sense that it emerges from the choices of the contributors rather than from a central coordinator. It is not so in other product development settings. Companies in the commercial software industry, for example, typically decouple the high-level task of architectural design from the low-level task of code implementation, assigning the former to a central designer and the latter to a group of programmers (Brooks 1995). Such an organisational configuration implies that the modularisation process may only be enacted through a top-down managerial intervention and its implementation is centrally coordinated. In such hierarchical contexts, the application of code modularity might be geared to enhancing the control of managers over their subordinates, as has actually been observed (Reinstaller 2007). Modularity in FOSS projects, by contrast, is emergent rather than imposed from the top. It is not a decision made by a lead architect or in a command centre cut off from the actual site of development, but a mechanism to which project developers are driven by their own decision to focus their work on some area of the codebase, as a result of the learning costs attendant upon its growth. For much the same reason, we expect structural changes such as increasing size to trigger radically different responses in hierarchical organisations and FOSS communities. To effect coordination within an expanding group of developers, the FreeBSD project resorted to tightening its control over the inputs and outputs of the development process (through the induction process for new members and the practice of frequent building, respectively). But a hierarchical organisation would arguably be more inclined to manage coordination issues attendant upon increasing size much differently: for instance, by intensifying supervision and central coordination or by introducing additional layers of hierarchy (e.g. Brusoni & Prencipe 2006; Staudenmayer et al. 2005).

At the same time, we should not overlook the importance of some factors:

(1) The phenomenon of distributed product development is not limited to FOSS projects: the drive to distribute production requirements across the network pervades all industries.
(2) Nor is the model of open/user innovation (Chesbrough 2003; von Hippel 2005) confined to FOSS: increasingly more organisations try to tap into the creative potential of actors outside their boundaries and involve them in the product development process. FOSS development is but an extreme example of such innovation.

(3) Increasingly more organisations experiment with self-managing teams and regimes of peer control (Barley 1993; Sewell 1998), empowering their employees to participate in the managerial process.

Distributed work organisations characterised by horizontal control processes and extensive participation of external actors look very much similar to the organisational configuration exemplified by FOSS projects. It is quite likely therefore that such organisations will manifest a similar response to increased scale as FreeBSD and look upon product structure as a variable that can be manipulated to better support distributed development.

The epilogue which follows reflects on the effect of expanding organisational size on organisational structure, collating the results of our study against a long tradition in the social sciences, according to which the separation of order-givers from order-takers is the inevitable concomitant of increasing size.

EPILOGUE

Social scientists have for a long time laboured under the assumption that as a group grows larger, it becomes less able to self-organise. This view has been so influential that it has left an indelible mark on the social sciences. A few examples will suffice to demonstrate this point. In what has come to be regarded as the founding tract of crowd theory, Gustave Le Bon's *The Crowd*, first published in 1895, the 'crowd' is portrayed as mentally retarded and destitute of the faculty of judgement. For Le Bon, the crowd, no matter how competent or intelligent in their own right are the individuals who comprise it, is incapable of displaying even the slightest common sense: as he exclaims, 'in crowds, it is stupidity and not mother-wit that is accumulated' (Le Bon 2002, p. 6). That is so, Le Bon contends, because the behaviour of individuals, once they are part of a crowd, becomes totally dominated by its 'collective mind'. And this collective mind, on account of its 'infantile state', needs a strong leader to guide it. In consequence of the psychological transformation that individuals undergo in groups, crowds are susceptible to demagoguery. Le Bon's message was clear: the notion of groups without leaders is chimerical. His vitriolic statements left no room for any other interpretation: 'The crowd demands a god before anything else' (Ibid., p. 40); 'a crowd is a hostile flock that is incapable of ever doing without a master' (Ibid., p. 72); 'it is the need not of liberty but of servitude that is always predominant in the soul of crowds' (Ibid., p. 75). Le Bon's provocative views were reprised in later seminal works. Characteristically, in his 1921 venture into the field of crowd psychology, Sigmund Freud dismissed as erroneous the notion that crowds, be they transient group formations or stable associations, can exist without leaders: 'man is...a horde animal, an individual creature in a horde led by a chief' (Freud 1975, p. 68).[169] According to Freud, what unites individuals in groups is not so much their affinity as their emotional tie with the leader. It is the tie with the leader, rather than the ties between themselves, that is the ruling factor.

In the ninety years that elapsed since Freud's diatribe, it is true, classic crowd theorists have been castigated for masquerading their own anti-democratic views as the results of scientific inquiry. Nevertheless, the notion that crowds are unfit to

169 Freud's critique was levelled against Wilfred Trotter's (1916) argument that the role of leaders in groups had been unduly overemphasised.

govern themselves has proven to be remarkably persistent.[170] In the few instances in the literature where mention is made of leaderless and anti-hierarchical groups, it is always in the shape of an exceptional case that has little bearing on the subject at hand. Thus Elias Canetti, in his monumental study of crowds, praises the spontaneity, responsibility and dignity evinced in the bottom-up organisation of worker-occupied factories, underlining however that such structures have a characteristic shortness of life, being unsuitable for organisation on a stable basis (Canetti 1984, pp. 56-58).[171]

The thesis that an increase of scale in numbers undermines a group's ability to self-organise and self-govern penetrated academic sociology through the work of Max Weber. According to Weber, direct-democratic forms of administration are possible only in circumstances where (a) group members can gather together in a single spot and (b) administrative tasks can be carried out by any group member:

> In addition to the small scale of the group in numbers or territorial extent, or still better in both, as essential conditions of immediate democracy, is the absence of qualitative functions which can only be adequately handled by professional specialists (Weber 1947, p. 413).

More specifically, the ability of a group to dispense with leaders collapses when

> the group grows beyond a certain size or where the administrative function becomes too difficult to be satisfactorily taken care of by anyone whom rotation, the lot, or election may happen to designate. The conditions of administration of mass structures are radically different from those obtaining in small associations...The growing complexity of the administrative tasks and the sheer expansion of their scope increasingly result in the

170 For a recent literature review, see Mazzarella (2010).

171 The most notable exception to this general trend is Hardt and Negri's (2000, 2004) conceptualisation of the *multitude* as inherently opposed to intermediation and representation. Yet, even they feel compelled to draw a distinction between their use of the concept of the multitude and that of the crowd (perhaps in order to distance their work from the field of crowd theory), contending that the crowd 'is fundamentally passive in the sense that it needs to be led and cannot act of its own accord, autonomously', whereas 'the multitude, in contrast, must be...capable of acting autonomous' (Hardt 2006).

> technical superiority of those who have had training and experience, and will thus inevitably favor the continuity of at least some of the functionaries. Hence...the rise of a special, perennial structure for administrative purposes, which of necessity means for the exercise of rule (Weber 1978, pp. 951-952).

As large scale militates in favour of centralised administration and the necessity of specialisation to fulfil specific administrative tasks creates a stratum of experts who, by gradually appropriating their functions, come to concentrate in their hands all actual power, Weber concluded that this process eventuates in bureaucratic administration. Robert Michels' well-known study of socialist parties and trade unions in pre-World War I Europe made this point even more forcefully. Michels argued that large-scale 'organization implies the tendency to oligarchy'. As a result, 'every party or professional union becomes divided into a minority of directors and a majority of directed' (Michels 1915, p. 32). Michels' argument rested on the same points raised by Weber: 'the technical specialization that inevitably results from all extensive organization renders necessary what is called expert leadership' (Ibid., p. 31). The necessity of large-scale organisation then, according to Michels, forces even those political organisations that aspire to egalitarian ideals to adopt bureaucratic structures. Their large size makes them dependent upon professional specialists for all sorts of administrative tasks and so power passes from the rank and file to the experts who run the organisation. Thirty five years later, Philip Selznick's study of the Tennessee Valley Authority reiterated Michel's conclusions. By focusing on an organisation known for its commitment to democratic ideals, Selznick's study sought to emphasise that the end-product of increased size and administrative complexity in organisations, 'whether formally democratic or not...is a split between the leader and the led, between the agent and the initiator' (Selznick 1949, p. 9). Although theories explaining the organisational split of a group into a class that commands and another that obeys as a 'constraint' imposed by the administrative requisites of large scale organisation have been criticised as unwarrantably pessimistic and fatalistic (e.g. Gouldner 1955), the tendency of modern theorists of group organisation to insist that expanding organisational size leads by necessity to centralised authority is as alive today as ever. According to social anthropologist Robin Dunbar (1993), for instance, a group's ability to dispense with hierarchy depends on its size. If it numbers less than 150 members,

group activities may well be coordinated informally, based on the mutual adjustment of participants. But once it exceeds 150 members, its ability to self-organise without formal rules wanes. After crossing this threshold, coordination can only be achieved by the erection of formal hierarchical structures.

Against the backdrop of the foregoing discussion, our findings seem quite unnatural. One would expect that a project like FreeBSD, whose (*src*) developer base has increased more than tenfold from 16 to 198 members in the space of thirteen years, would have resorted to some type of hierarchical arrangement to combat the increased coordination costs attendant upon the expansion of scale. However, the expansion of the FreeBSD committers group, though it increased the need for active coordination, did not result in its social stratification, that is, to an internal hierarchy where contributions are processed upstream through gatekeepers. The structure of organisation of daily work with respect to change integration remained much the same. How is that possible? Here we might return to our discussion of modularity by reformulating the question in terms of modularity theory: does modular product design make hierarchical organisation unnecessary by mitigating the need for active coordination within the committers group? As we saw, the historical increase of scale in numbers of participating committers is accompanied by higher levels of component modularity. To a certain extent, the tendency of FreeBSD modules to become less tightly coupled over time has helped to moderate the need for active coordination between groups of committers who focus on distinct modules, thus enabling them to work independently of one another. However, product modularity cannot by itself account for the eschewal of hierarchical coordination. An interpretation of this outcome that rests solely on the moderating effect of product modularity on coordination costs is, in Weberian terms, not 'casually adequate'.[172] Modularity would furnish such a casually adequate explanation for the mode of organisation of committers if it could be shown empirically that the adoption of a modular product design is a sufficient condition for the forestalling of hierarchical coordination. But this is not possible: modular product development does not preclude hierarchical organisation. Quite the contrary, the development of a modular product may very well be organised in a hierarchical fashion, as shown by a plethora of studies which document the development of modular products inside conventional organisations. In the software industry in particular where modularity has been established as the

172'Causal explanation depends on being able to determine that there is a probability, which...is always in some sense calculable, that a given observable event (overt or subjective) will be followed or accompanied by another event' (Weber 1978, pp. 11-12).

dominant design principle, one would have to look very hard to find a company that does not lay emphasis on modular product design. The problem then with an interpretation that holds product modularity to be the ruling factor for the non-hierarchical organisation of FreeBSD committers is obvious: modular product development can be shown to be equally compatible with hierarchical structures of organisation and non-hierarchical ones.[173] Modular product design might be a key enabler for the mode of organisation of committers but is certainly not sufficient in itself to bring it about.

What throws light not only on the mode of work organisation in FreeBSD but more generally on the mode of project governance is the normative principle of individual autonomy of action. Hackers are driven by an acute sense of independence: they do not like taking orders by others. It is telling that programmers have a notorious reputation in the software industry for disobeying their project managers and defying their authority. In the words of a manager:

> The technologists more closely identified with the digital computer have been the most arrogant in their wilful disregard of the nature of the manager's job. These technicians have clothed themselves in the garb of the arcane wherever they could do so, thus alienating those whom they would serve (quoted in Ensemenger & Aspray 2000; see also Barley 1996, pp. 429-434).

Hackers' disdain for bureaucratic authority is nowhere more pronounced than in the realm of hacker projects. That is only logically consistent, of course, considering that hacker projects are explicitly set up as anti-bureaucratic spaces of collaboration. Their raison d'être is not only to produce software but to exemplify the common conviction of hackers that software development can be organised without the hierarchical controls inherent in bureaucratic organisations. In rejecting bureaucratic hierarchies, hackers attune their actions to a value system which is as old as hacker culture itself and which exalts the autonomy of the individual as a cardinal value. This moral tradition is better-known as the hacker ethic and emphasises individual autonomy and self-determination as the principles by which the conduct of hackers should abide (Himanen 2001; Levy 1984; Turner

173 As it has been remarked, the introduction of product modularity in some companies is aimed at enhancing the control of managers over their subordinates (Reinstaller 2006, 2007).

2006). From the perspective of hacker morality, forms of collective organisation which contravene the autonomy and self-determination of the individual are abhorrent and harmful. In consequence of this normative standard, hackers tend to adopt structures which are reckoned to maximise their individual autonomy. As Thomas Paine (1791) has said, 'forms grow out of principles and operate to continue the principles they grow from'. That is certainly true of FreeBSD: stripping committers of the right to commit changes directly to the codebase would have amounted to the delegitimisation of the FreeBSD governance system. It is the subjective reason why FreeBSD committers did not opt for a hierarchical solution to the problem of increased coordination costs, but instead resorted to standardising (a) the induction process for new committers and (b) outputs via frequent building. In order to manage increased scale, instead of resorting to direct supervision as a means of coordination, FreeBSD tried to reduce the need for active coordination within the committers group. To achieve this, the project (a) focused on building into the committers-to-be the work programs as well as the bases of coordination and (b) established a performance standard for the code checked in by committers. This line of development cannot be understood apart from the normative principle of individual autonomy of action. The significance that committers attribute to their autonomy elucidates the course of action that was taken to manage increased scale as their conscious choice.

By elucidating the intervening motivational link between the conduct of committers and the observed outcome, the above interpretation is suggestive of the level of control that committers, by reflexively regulating the overall conditions of reproduction of the FreeBSD social system, are characteristically able to sustain over their conduct. However, it can be reformulated so as to engage more critically with Weber's analysis. It will be remembered that for Weber – and even more so for some of his students like Michels and Selznick – the separation of the directors from the executants is essentially an organisational constraint triggered by increased size and complexity of administrative functions; the outcome of organic necessity regardless of the feelings of organisational members. The objective conditions that Weber considered necessary for the functioning of a direct-democratic form of governance in a group are, on the one hand, the small size of the group, and the absence of administrative functions whose fulfilment necessitates specialisation, on the other. In order that decisions can be made collectively, the size of the group must be small enough so that group members can assemble in a single spot. But in FreeBSD, as mailing lists are the project's main

communication fora, there is no need for project members to be physically present in one place. The project's 'general assembly' is in a sense convoked whenever a committer posts a message to one of the project's mailing lists. Thus, purposive discussion is distributed across space: a message posted by a committer from America may generate replies from committers logging on to the Internet from as many as thirty-four different countries. Equally important is that online discussions are distributed across time: not everyone has to participate at the same time. Mailing lists permit asynchronous communication, thereby imparting flexibility to group decision making, as committers can participate in their own time frame. It is on account of this flexibility that a characteristic problem of collectivist organisations – interminable meetings – is overcome in FreeBSD.

As far as the role of administrative expertise in FreeBSD is concerned, the project has tried to ensure that administrative tasks can be handled by any committer. To become a core team member, one has to be a committer first. Thus it is made clear that project administrators must be thoroughly involved in code development; it is not required of them to be management experts. The administrative tasks they are called upon to fulfil – managing commit rights and mediating in developer conflicts that are not self-revolving – are obviously not of the type which calls for such specialised skills as, for instance, a professional accountant possesses. On the contrary, those administrative functions are reckoned to be in the ability of every committer to discharge. However, though it admits no management specialists into its ranks, FreeBSD itself is an organisation of experts; it is made up of highly skilled programmers. As one of them says: 'By and large, most of the committers are better programmers than the people I interview and hire in Silicon Valley' (quoted in Jørgensen 2001). A long tradition in the social sciences has it that experts are an instrument of bureaucratic domination. According to Weber, for example, bureaucracy is nothing but 'expert officialdom' (Weber 1994): a mode of organisation of experts characterised by their circumscribed sphere of activity and their subordination to an impersonal hierarchical order. For Foucault (1975), likewise, the spread of 'disciplinary institutions' – encompassing modern schools, hospitals, prisons, barracks and factories – from the 18th century onwards is inseparable from the emergence of a new type of administrative authority exercised through experts such as teachers, doctors, social workers or business managers. However, as works in organisation theory have shown, the mode of work organisation of experts may well be anti-bureaucratic. Indicatively, Mintzberg and McHugh's study of the National Film Board of Canada remarked that 'the obsession

with control found in...bureaucracy is anathema to the exercise of expertise' as it contravenes the organisational flexibility required for the performance of non-routine tasks (Mintzberg & McHugh 1985, p. 192). Unfamiliar problems require not only the extensive involvement of experts, but also a considerable degree of flexibility in dealing with them, which bureaucratic structures – catering for control, not flexibility – cannot provide. That is why 'conventional administration', as Mintzberg and McHugh write, 'is so fruitless in an organization of experts'. Their work demands 'structures [which] are designed largely to leave these people free to work as they know how'. Such structures are often referred to in the literature as *adhocracies* (Mintzberg & McHugh 1985) or *organised anarchies* (Cohen et al. 1972). FOSS projects could indeed be seen as examples of organised anarchies or adhocracies. Hackers' rejection of supervisory hierarchy is analogous to the autonomy from managerial control other professionals enjoy by virtue of being expected to exercise judgement and discretion in the course of performing their daily work. But while professionals working in organisations, even in the most 'adhocratic' ones, are invariably subject to some measure of bureaucratic control (Bendor et al. 2001, p. 173), hackers have completely ousted bureaucratic authority from their organisational frame. The way the Internet was developed by the original 'Internet tribe' – a globally dispersed network of parsimoniously linked, self-regulating groups of computer hackers – became a template for articulating authority on the development process of FOSS projects. In the making of the Internet, as Internet researcher Mathieu O'Neil writes,

> Quasi-scientific expertise became independent from hierarchical institutions: hackers recognised the judgement only of their peers. The authority of *experts* is traditionally subordinated to the authority of *leaders*. However when the Internet was developed *learned authority* to a great extent determined *administrative authority* for the simple reason that only computer hackers knew how to run the systems (O'Neil 2009, p. 2).

As hackers were in position to understand the problems – both technological and organisational – that the development of the Internet entailed better than anyone else, they became the 'experts' entrusted not only with the development of the technological infrastructure but also with the management of the enterprise.

Hackers formed, as it were, a network of autonomous expertise: in building the Internet, they evolved new models and procedures for the production, evaluation and dissemination of information and software independently of state and corporate authorities. The story of the development of the Internet illustrates clearly that expert labour can be harnessed in organisational models quite different from those Weber considered germane to professional work. The same applies to the case of FreeBSD. In fact, FreeBSD demonstrates even more succinctly that there is no 'iron law of oligarchy' set in motion by increased scale. While Weber held that the setting up of bureaucratic hierarchy is the inevitable outcome of expanding organisational size, that is not what happened in FreeBSD. We must free ourselves from the view that one can deduce the organisational configuration of a group, as a historically necessary development, from such structural changes as increased size. As FreeBSD shows, the conditions governing the emergence and development of hierarchy in a group are not independent of the values held by its members. There is nothing in the nature of our data that lends credence to such a mechanical conformity to the organisational exigencies of increased scale as posited by the Weberian school. Though the expansion of scale catalysed some changes in project governance, those were not in the direction that Weber would have anticipated. In order to cope with increased scale, instead of attempting to control the work process of individual committers, FreeBSD sought to control its inputs – by specifying the kind of training required to perform the work – and outputs – by specifying a performance standard for the code checked in by committers. Such an organisational response is of course not entirely original: control of the inputs to a process or of its outputs is an organisational device commonly employed when the content of the process itself does not admit of control (Coleman 1993; Mintzberg 1993; Simon 2002). In this respect, FreeBSD is not unique. Rather, its analytical significance lies in demonstrating that the organisational devices employed by a group are contingent upon the mode of orientation of social action in the group. It is to the hacker ethic that we may trace not only the qualities which distinguish the (attitudes underlying the) orientation of social action in FOSS projects from that obtaining in bureaucratic organisations but also the subjective conditions which determine the organisational devices employed by FOSS projects in response to increased scale.

Hackers do not consider themselves to be workers but autonomous creators who find the impulse for what they do in the joy it gives them. For them, programming is an end in itself. Their participation in FOSS projects can be more aptly described

as a labour of love or hobby than as work. As Linus Torvalds (2001, p. xvii) says, the reason that 'hackers do something is that they find it to be very interesting, and they like to share this interesting thing with others...Hackers [are] working together because they enjoy what they do'. Torvalds' account of what makes hackers tick encapsulates a unified view of ethics, which exalts the joy and autonomy inherent in intrinsically-motivated activities as well as the practice of free sharing of software that lays at the heart of the hacker community. This moral code, as aforementioned, is known as the hacker ethic. It is no coincidence that it is expounded by those who study it as the direct antithesis of bureaucratic authority and the protestant work ethic (i.e. the value system which, according to Weber's famous thesis, was a catalyst for the rapid development of modern capitalist activity). For Himanen (2001), for example, the protestant ethic's main feature – the disciplined obligation of work as a duty – contrasts sharply with the hacker ethic's celebration of creative activity as an end in itself: its eulogy of joy and creative spontaneity is diametrically opposed to the protestant ethic's emphasis on 'the earning of more and more money, combined with the strict avoidance of all spontaneous enjoyment' (Weber 2005, p. 18). For Levy (1984) and Turner (2006), similarly, the hacker ethic was forged out of hackers' disdain for corporate bureaucracies. Given that the hacker ethic serves as an organising norm for the activities of hackers, it should not be surprising that the organisational devices employed by FOSS projects differ so radically from those that bureaucratic organisations tend to resort to.

It is to be expected that readers familiar with Michels' and Selznick's studies will be sceptical of interpretations that explain organisational outcomes by reference to moral factors, regarding them as idealistic and ignorant of the basic conclusion one must draw from the work of Michels and Selznick, namely that even organisations that are strongly committed to egalitarian ideals are forced to evolve into bureaucratic hierarchies when they have grown big enough. The fact that this line of development contravenes the founding principles of those organisations just goes to show that the moral values espoused by their members have very little, if anything, to do with edging them onto that direction; at the very least, it shows that objective factors take precedence over subjective ones. That is no doubt a crucial perspective on the dynamics of organisational evolution, which, by highlighting the primacy of objective conditions in shaping organisational outcomes, implies that the governance structure of FreeBSD is not the product of the ideas animating its members, as our analysis so far seems to suggest. There is

more than a grain of truth in this syllogism. Ideas and values do not in themselves suffice to forestall the development of hierarchy in a group: the objective conditions in which the group operates must be incompatible with hierarchical organisation as well. That is certainly true of FOSS projects: the objective environment embedding them does not provide the preconditions required for the emergence of oligarchy. Our analysis has already touched upon some of the objective parameters of the FreeBSD organisational system that obviate the need for hierarchical bureaucratisation: (a) there is no need for project members to assemble in a single spot in order to make decisions, as purposive discussion occurs on Internet mailing lists and so is distributed across space and time; (b) there are no administrative tasks in the project that require specialised training; and (c) modularity reduces the need for active coordination between developers of different components. These factors go a long way to explain the *radical autonomy* of FreeBSD developers. But they are not the only ones. Three equally important ones come readily to mind: (d) FOSS developers are not bound by relations of economic dependency; (e) the composition of FOSS projects is highly dynamic due to the mobility of their members, that is, the ease with which they can enter or exit them; and (f) FOSS development takes place in a distributed environment, as developers are dispersed all over the world. Taken together, these characteristics of FOSS projects suggest that their members cannot be managed in the traditional sense of the word (see also van Wendel de Joode 2005). In a sense, developers' autonomy is built into the very parameters of the distributed work environment of FOSS projects, being a function of the objective conditions of FOSS development. To put it more simply, nobody is in position to take away their autonomy: there is no process in FOSS projects that their administrators could latch on in order to dominate the other members. Thus, while the subjective conditions underlying the specifically non-hierarchical response of FreeBSD to its historical expansion of scale can be traced to the hacker ethic, the objective conditions that precluded the transformation of the project in an authoritarian direction spring from the incompatibility of the distributed environment in which FreeBSD operates with coercive authority.

To close this study, a comment on the cultural significance of FOSS seems fitting. Let it be allowed me to formulate it in the context of Weber's reflections on the tension between the demand for technical efficiency and productivity, on the one hand, and the human values of spontaneity and autonomy, on the other, that manifests itself as a result of the advance of bureaucratisation. For Weber, modern

capitalism is by far the most advanced economic system that ever existed in terms of the substantive values of efficiency and productivity. But the very rationalisation of social life which has made it possible to achieve this level of administrative efficiency and labour productivity has consequences that contravene some of the most distinctive values of western civilisation such as creativity and autonomy of action. Weber was cognizant of the harrowing possibility that the domination of bureaucratic institutions over social life, which he regarded as inescapable, might thrust humanity into an 'iron cage':

> No one knows who will live in this cage in the future, or whether at the end of this tremendous development entirely new prophets will arise, or there will be a great rebirth of old ideas and ideals, or, if neither, mechanized petrification, embellished with a sort of convulsive self-importance. For of the last stage of this cultural development, it might well be truly said: "Specialists without spirit, sensualists without heart; this nullity imagines that it has attained a level of civilization never before achieved" (Weber 2005, p. 124).

The 'cage' of which Weber speaks is the degeneration of humans to cogs in an administrative machine: their transformation into the underlings of an impersonal apparatus, hierarchically organised, which has in its hands the management of collective activities. The very working existence of hacker projects like FreeBSD, by constituting autonomous spaces where individual autonomy of action and self-determination are given free play, dispels to a great extent such pessimism. By furnishing a concrete example of organisation without coercive authority, perhaps the most important contribution of FOSS projects consists in sketching the general outlines of a form of collective organisation in which under no circumstances is the suppression of individual autonomy of action ever justified.

SUMMARY

The central question driving this PhD research is whether modularity, by mitigating the need for active coordination between distinct components, increases the potential number of contributors to a free/open source software (FOSS) project and has a positive effect on their labour productivity, allowing them to work independently of each other.

Chapter 1 situates the emergence of the design principle of modularity as a response mechanism to the organisational problem of decreasing returns to scale.

Chapter 2 reviews the literature on modularity as a design principle for complex product development, drawing the following hypotheses for subsequent empirical testing:

- *Product modularity reduces coordination costs in FOSS projects (H1)*
- *Product modularity increases the potential number of contributors to a FOSS project (H2)*
- *An increase of contributors to a FOSS project results in an increase of modularity (H2R)*
- *Product modularity has a positive effect on labour productivity in FOSS projects (H3)*
- *An increase of contributors to a FOSS project has a negative effect on labour productivity (H4)*

Chapter 3 describes the research methodology: it explains our indicators of modularity, coordination costs, group size and labour productivity and the manner in which we use panel data (a.k.a. longitudinal or time-series data) collected from FreeBSD's software repositories to put the hypotheses to the test. In specific, our analysis takes place on two levels: we examine the relationship between modularity, coordination costs, group size and productivity (a) at the project-level (that is, for the FreeBSD project as a whole) through a qualitative analysis of descriptive statistics and (b) at the component-level (i.e. at the level of the individual modules making up the FreeBSD operating system) through a quantitative analysis of a dataset that includes thirty modules selected through stratified random sampling: modules were categorised into three strata based on

their scale, as reflected in the number of developers contributing to them (*N=387, H=3*), and ten modules were randomly selected from each stratum. The statistical instrument used for the quantitative analysis is random-effects GLS regression.

Chapter 4 introduces the empirical setting – the FreeBSD project – and discusses its historical and organisational background.

Chapter 5 examines the extent to which modularity reduces coordination costs in FreeBSD (*H1*) but finds no empirical support for the hypothesis that higher levels of modularity correlate with lower levels of coordination costs.

The first part of chapter 6 examines whether modularity increases the potential number of contributors to FreeBSD (*H2*) and provides strong empirical support to the hypothesis. The second part of chapter 6 tests *H2R*, which reverses the directionality of the effect so that increasing group size is claimed to result in an increase of modularity. The statistical tests we performed verify the hypothesised effect, provided that conditions of large-scale development (i.e. committers > 8) apply.

Chapter 7 examines the effect of modularity on group performance and finds that – to the extent that conditions of large-scale development prevail – modularity has a positive effect on both average group performance and core developers' performance.

Chapter 8 examines the effect of increasing group size on labour productivity. Our analysis of descriptive statistics shows that the historical expansion of the FreeBSD committers' group brought about a fall in average group productivity, seemingly confirming *H4,* but it also resulted in a rise in core developers' output. This finding is qualified by arguing that large groups enable a more extensive division of labour (on a voluntary basis, of course) within the modules they develop, thanks to which core developers can focus on their task of choice, namely new code development, thereby suggesting that the fall in group productivity is not caused by a fall in core developers' performance, but by the disproportionate increase of 'lower-contribution' committers over time. In the light of these results, *H4* cannot be wholly accepted, as the causal mechanism underlying the decrease of average productivity differs markedly from that which *H4* postulates (i.e. that the fall in group productivity is due equally to the low performance of new members and the fall in core developers' performance that is caused by the communication and coordination costs attendant upon increasing group size).

Chapter 9 examines the transformation of governance to which FreeBSD resorted in order to accommodate itself to expanding scale. Catalysed by the

growing criticism of the distribution of authority in the project, the adoption of the elective principle for the selection of the FreeBSD administrative team brought about a shift in the conception of leadership from the informal rule of a self-selected group of veteran developers to the democratic authority of an elected group that is revocable and bound to formal rules. Since, FreeBSD has evolved a collectivist governance system, based on a direct-democratic, consensus-oriented process of decision making. Furthermore, in keeping with the normative standard of individual autonomy of action, FreeBSD did not attempt to manage increased scale by supervising developers' work process but rather tried to achieve coordination through the standardisation of the induction process for new developers and of outputs through frequent building. Interestingly, the transformation of FreeBSD's governance structure contrasts sharply with how other large FOSS projects have attempted to manage increased scale. Characteristically, to facilitate coordination in an expanding group of developers, the Linux project introduced an additional layer of managerial hierarchy, as Linus Torvalds, the project leader, delegated authority to a cadre of subsystem maintainers – the so-called 'trusted lieutenants' – to filter the contributions of the wider base of Linux developers (Corbet et al. 2010, pp. 15-17; Moody 2001). Such a hierarchical response to increased scale points not only to the presence of important differences in the distribution of authority between FOSS projects but also to a strong element of trial-and-error experimentation with varying degrees of control over the process of integrating changes in the project code repository (Holck & Jørgensen 2004; Weber 2004).

Chapter 10 sums up the empirical findings and reflects on the role of modularity as a governance mechanism. As regards the effect of product structure on group dynamics:

(a) Modularity makes decentralisation scalable by mitigating the need for active coordination between distinct modules.
(b) Modularity reinforces the emergent division of labour: enlarging the scale of the project militates in favour of committers' specialisation (because of the learning costs involved in familiarising oneself with the codebase) to which modularity conduces by enabling the independent development of distinct product components.
(c) Modularity has a positive effect on average group productivity in large-scale conditions, for it allows developers to work independently of each

other. But in small-scale conditions (i.e. committers < 9), its effect is insignificant: this implies that the potential of modularity can be fully exploited only in settings in which the need to mitigate the adverse effects of increasing scale takes on a pressing character.

With respect to the effect of group dynamics on product structure:

(a) Product structure mirrors organisational structure: the distributed character of the development process of large FOSS projects such as FreeBSD implies that the scope for face-to-face communications is drastically narrowed. Because of the inherent limitations on communication, therefore, the product architecture that evolves is more modular.
(b) Product structure constitutes a coordination mechanism. As FreeBSD (and FOSS projects in general) is devoid of an authority structure by which to effect coordination, FOSS developers are induced to use software structure as a variable that can be fine-tuned to reduce the need for active coordination between product components.
(c) However, as long as the overall group of developers working on a module remains small (i.e. it does not exceed eight developers), adding more developers to the group prompts no changes in the direction of increased modularity. The product structure that evolves is then non-modular because it reflects the work patterns of a tightly-coupled group.

With respect to the effect of group size on labour productivity:

(a) The high performance of core developers is not due to the absence of coordination costs but to the temporally increased scope of their participation.
(b) Large groups enable a more extensive division of labour (on a voluntary basis) within the modules they develop, thanks to which core developers can focus on their task of choice, namely new code development. Hence, increasing group size results in boosting core developers' performance.
(c) Crucially, the potential for decreasing returns to scale is blunted by the motivational forces at work: when one is working on tasks perceived as meaningful and engaging as well as in groups one highly values, then

adding more persons to the group no longer has a demotivating effect. The ability to evaluate (a) individual committers' contributions to the collective outcome and (b) the performance of the committers' group as a whole against other FOSS projects' group performance also reinforces the motive to contribute. Furthermore, the pattern of scale expansion encountered in FreeBSD has been equally important in averting the manifestation of decreasing returns to scale: as the expansion of scale in FreeBSD was not accompanied by a taller hierarchy, it did not result in the communication distortions commonly besetting hierarchical structures.

In the *Epilogue*, the findings of our study are collated against a long tradition in the social sciences which holds that an increase of scale in numbers undermines a group's ability to self-organise and self-govern. We challenge this theory by arguing that, on the one hand, the conditions governing the emergence and development of hierarchy in a group are not independent of the values held by its members and, on the other, that the characteristics of the distributed environment in which FOSS projects operate render it incompatible with the exercise of coercive authority.

SAMENVATTING (SUMMARY IN DUTCH)

De centrale vraag die ten grondslag ligt aan dit promotieonderzoek is of modulariteit – door het verminderen van een behoefte aan actieve coordinatie tussen gescheiden software modules – het potentiële aantal ontwikkelaars die hun bijdrage leveren aan een FOSS project vergroot en hun arbeidsproductiviteit verhoogt doordat zij onafhankelijk van elkaar kunnen werken.

Hoofdstuk 1 schetst een beeld van de opkomst van het concept modulariteit als ontwerpprincipe, in reactie op het organisationele vraagstuk van afnemende productiviteit bij toenemende schaalgrootte.
Hoofdstuk 2 geeft een overzicht van de literatuur over modulariteit als ontwerpprincipe voor de ontwikkeling van complexe producten. Op basis van deze literatuur zijn de volgende hypotheses opgesteld, welke worden getoetst in het empirisch deel van dit onderzoek:

- *Productmodulariteit leidt tot een afname in coordinatie kosten in FOSS projects (H1)*
- *Productmodulariteit leidt tot een toename in het potentiële aantal ontwikkelaars die bijdragen aan een FOSS project (H2)*
- *Toename in het aantal ontwikkelaars die bijdragen aan een FOSS project resulteert in een toename van de productmodulariteit (H2R)*
- *Productmodulariteit heeft een positief effect op arbeidsproductiviteit in FOSS projects (H3)*
- *Toename in het aantal ontwikkelaars die bijdragen aan een FOSS project heeft een negatief effect op arbeidsproductiviteit (H4)*

Hoofdstuk 3 beschrijft de onderzoeksmethodologie: het geeft uitleg over de indicatoren van modulariteit, coordinatie kosten, groepsgrootte en arbeidsproductiviteit, die in dit onderzoek worden gebruikt. Hiernaast beschrijft dit hoofdstuk de wijze waarop panel data (i.e. longitudinale of time-series data afkomstig van de FreeBSD repositories) wordt gebruikt om de onderzoekshypotheses te testen. In de onderzoeksanalyse onderscheiden wij twee niveaus: We analyseren de relatie tussen modulariteit, coordinatie kosten, groepsgrootte en productiviteit (a) op project-niveau (dwz voor het gehele FreeBSD

project) door middel van een kwalitatieve analyse van beschrijvende statistiek en (b) op de module-niveau door een kwantitieve analyse van een gestratificeerde a-selecte steekproef van 30 modules uit het FreeBSD project. Op basis van het aantal ontwikkelaars dat een bijdrage levert zijn de modules ingedeeld in drie categorien van schaalgrootte (klein, midden, groot). Voor dit onderzoek zijn uit elk van deze drie categorieen, op a-selecte wijze, tien modules geincludeerd in de steekproef. Voor de kwantitatieve analyse van de steekproef data is gebruikt gemaakt van random-effects GLS regressie-analyse.

Hoofdstuk 4 gaat in op de historische en organisatorische achtergrond van het FreeBSD project waarop het empirisch deel van dit onderzoek op is gericht.

Hoofdstuk 5 gaat in op de vraag in hoeverre modulariteit de coordinatie kosten vermindert (H1). Dit hoofdstuk concludeert dat deze hypothesis niet kan worden geverifieerd.

Het eerste deel van hoofdstuk 6 richt zich op de vraag of sterkere mate van modulariteit leidt tot een toename in het aantal ontwikkelaars dat een bijdrage levert aan een FOSS project (H2). Dit hoofdstuk concludeert dat de empirische bevindingen deze hypothese ondersteunen. Het tweede deel van hoofdstuk 6 beantwoordt de omgekeerde vraag, namelijk of een toename in de groepsgrootte van de ontwikkelaars resulteert in een toenemende mate van modulariteit (H2R). Dit veronderstelde effect wordt bevestigd door de statische analyses, maar dit geldt alleen voor modules die zijn gecategoriseerd als modules met een grote schaalgrootte (i.e. met meer dan acht ontwikkelaars).

Hoofdstuk 7 gaat in op het effect van modulariteit op de productiviteit van de groep, en vind dat – zolang de voorwaarden voor grote schaal ontwikkeling van kracht zijn – modulariteit een positief effect heeft op zowel de gemiddelde productiviteit als op de productiviteit van core-ontwikkelaars.

Hoofdstuk 8 richt zich op het effect van toename van het aantal betrokken ontwikkelaars op arbeidsproductiviteit. Onze analyse toont aan dat de groei van het FreeBSD groep over de jaren heeft geleid tot een afname in gemiddelde arbeidsproductiviteit. Op het eerste gezicht lijkt de analyse hiermee H4 te bevestigen. Echter de toename in het aantal betrokken ontwikkelaars is ook gepaard gegaan met een toenemende productiviteit van de core-ontwikkelaars. Deze bevinding kan worden worden verklaard door het feit dat grotere groepen van ontwikkelaars het mogelijk maken om werkzaamheden binnen een module beter te verdelen. Hierdoor kunnen core-ontwikkelaars zich meer toeleggen op hun primaire interessegebied, namelijk het programmeren. De afname in productiviteit

komt dus niet doordat de core-ontwikkelaars minder produceren, maar doordat het aantal minder productieve ontwikkelaars die betrokken zijn bij de module toeneemt. In het licht van deze bevindingen kan H4 daarom niet zonder meer worden aangenomen omdat de gevonden verklaring voor de afname van de gemiddelde productiviteit afwijkt van het standpunt dat is weergegeven in H4. Deze hypothese veronderstelt namelijk dat zowel nieuwe ontwikkelaars als de core-ontwikkelaars minder productief worden naarmate de groepsgrootte van het aantal betrokken ontwikkelaars en door toenemende coordinatie kosten.

Hoofdstuk 9 reflecteert op de wijze waarop FreeBSD zelf heeft gereageerd op de toenemende schaalvergroting door wijzigingen in aansturing en governance door te voeren. Onder invloed van groeiende kritiek op de verdeling en toekenning van autoriteit binnen het project werd een gekozen FreeBSD bestuursteam ingesteld. Deze transformatie heeft een verschuiving teweeggebracht van informeel leiderschap in handen van een selecte groep veteranen naar een democratischer vorm van leiderschap in handen van een gekozen bestuursteam dat is gebonden aan formele regels. Met deze verschuiving in leiderschap heeft FreeBSD een collectivistisch governance model ontwikkeld dat is gebaseerd op een direct-democratisch en consensus-georienteerd besluitvormingsproces. Door de handhaving van de normatieve standaard van individuele autonomie heeft FreeBSD met de ontwikkeling van het nieuwe governance model niet geprobeerd om op schaalvergroting te sturen door over de schouders van ontwikkelaars mee te kijken. In plaats van hiervan is geprobeerd de coordinatie te verbeteren door het standardiseren van het inductie proces van nieuwe ontwikkelaars en door het standaardiseren van de output, door regelmatig te compileren. Interessant is dat de transformatie van de FreeBSD governance structuur in scherp contrast staat met de wijze waarop andere grote FOSS projecten invulling hebben gegeven aan sturing op (effecten van) schaalvergroting. Bijvoorbeeld, het Linux project een hierarchische managementlaag geintroduceerd om coordinatie binnen een groeiende groep betrokken ontwikkelaars te faciliteren. Hiertoe heeft Linus Torvalds, de leider van het Linux project, autoriteit toegekend aan een kader van subsysteem-beheerders – de zogenoemde 'trusted lieutenants'- om de bijdragen van de ontwikkelaars te filteren (Corbet et al. 2010, pp. 15-17; Moody 2001). Deze hierarchische benadering van sturing op schaalvergroting wijst niet alleen op significante verschillen in de distributie van autoriteit tussen FOSS projecten, maar ook op trial-and-error experimenteren met varieerende gradaties van controle over het proces van het integreren van wijzingen in de project code repository (Holck & Jørgensen 2004;

Weber 2004).

Hoofdstuk 10 vat de emperische bevindingen van dit onderzoek samen en reflecteert op de rol van modulariteit binnen governance structuren. De belangrijkste bevindingen met betrekking tot de effecten van productstructuur op groepsdynamiek zijn de volgende:

(a) Modulariteit maakt decentralisatie schaalbaar doordat de behoefte aan coordinatie tussen gescheiden modules afneemt.
(b) Modulariteit versterkt een emergente arbeidsverdeling: toenemende schaalgrootte van projecten werkt specialisatie van ontwikkelaars in de hand, door de leerkosten die het doorgronden van de codebase met zich meebrengen. Modulariteit stimuleert specialisatie door onafhankelijke ontwikkeling van gescheiden product componenten.
(c) Modulariteit heeft ook een positief effect op de gemiddelde productiviteit binnen projecten met grotere schaalgrootte (committers > 8) omdat het ontwikkelaars in staat stelt om onafhankelijk van elkaar te werken. Echter, binnen projecten met een kleinere schaalgrootte (committers < 9), is dit effect van modulariteit niet significant. Dit betekent dat modulariteit alleen ten volle kan worden benut in een context waar de behoefte aan het afzwakken van negatieve effecten van schaalvergroting urgent is.

De belangrijkste bevindingen met betrekking tot de effecten van groepsdynamiek op productstructuur:

(a) De productstructuur weerspiegeld de organisatiestructuur: het gedecentraliseerde karakter van het ontwikkelproces in grote FOSS projecten – zoals FreeBSD – impliceert dat mogelijkheden voor face-to-face communicatie drastisch afnemen. Deze inherente beperkingen voor communicatie zorgen ervoor dat de product structuur een sterker modulair karakter ontwikkeld.
(b) De productstructuur vormt een coordinatiemechanisme. Omdat FreeBSD (en FOSS projecten in het algemeen) geen coordinerende autoriteit kennen, gebruiken FOSS developers de productstructuur als een variable die gefinetuned kan worden, om een behoefte aan coordinatie tussen de product componenten tot een minimum te beperken.
(c) Echter, zolang als de totale groep ontwikkelaars die betrokken zijn bij een

module klein blijft (committers < 9), leidt het toevoegen van meer ontwikkelaars aan de groep niet voor meer modulariteit. De productstructuur die eruit voortkomt is dan niet modulair omdat het de manier van werken van een hechte groep weerspiegeld.

De belangrijkste bevindingen met betrekking tot de effecten van groepsgrootte op arbeidsproductiviteit:

(a) De hoge productiviteit van core-ontwikkelaars is niet toe te schrijven aan lage coordinatie kosten, maar eerder door een tijdsgebonden toename in hun participatie in het project.
(b) Grote groepen betrokken ontwikkelaars bieden mogelijkheid tot een omvangrijker arbeidsverdeling (op vrijwillige basis) binnen de modules die ze ontwikkelen. Daardoor kunnen core-ontwikkelaars zich beter focusen op de dingen ze leuk vinden, namelijk programmeren. Daarom resulteert een toenemende groepsgrootte in een toename van de productiviteit van de core-ontwikkelaars.
(c) Door de motivatie van ontwikkelaars kunnen afnemende schaalopbrengsten worden opgeheven. Wanneer men werkt aan taken die als betekenisvol en boeiend worden ervaren binnen een groep waar men zich mee identificeert, dan heeft uitbreiding van deze groep niet langer een demotiverend effect. De zichtbaarheid van individuele bijdragen aan het collectieve resultaat en de herkenbaarheid van de groepsprestatie in relatie tot andere FOSS projecten vormen een prikkel om bij te dragen. Bovendien, heeft de wijze waarop de governance structuur van FreeBSD zich heeft ontwikkeld on der omstandigheden van schaalvergroting een belangrijke rol gespeeld in het afwenden van de negatieve effecten van schaalvergroting. Omdat de toenemende schaalgrootte van FreeBSD niet gepaard ging met een toenemend hierarchische organisatie van het project, kwamen de communicatie verstoringen die kenmerkend zijn hierarchische organisatiesturcturen niet voor.

In de Epiloog, worden de bevindingen van onze studie vergeleken met een lange traditie in de sociale wetenschappen welke voorstaat dat schaalvergroting het vermogen van een een groep om zichzelf te organiseren en besturen ondermijnt. Wij weerleggen deze stellingname, enerzijds door te beargumenteren dat de

condities die leiden tot de ontwikkeling van hierarchische organisatiestructuren in groepen niet los gezien kunnen worden van de normen en waarden van de groepsleden. Anderzijds, beargumenteren wij dat de kenmerken van de gedecentraliseerde context waarbinnen FOSS projecten opereren onverenigbaar zijn met een dwingende en directieve invulling van autoriteit binnen hierarchische organisatiestructuren.

APPENDICES

APPENDIX I: THE FREEBSD LICENSE

The FreeBSD License, also known as Simplified BSD License, is the variant of the original BSD software license[174] used by the FreeBSD Project for the distribution of its software. Its verbatim text is as follows:

The FreeBSD Copyright[175]

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE FREEBSD PROJECT ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FREEBSD PROJECT OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

174 The text of the original BSD software license (known also as the *4-clause BSD license*) is accessible online at <http://www.xfree86.org/3.3.6/COPYRIGHT2.html#6>

175 Accessible online at <http://www.freebsd.org/copyright/freebsd-license.html>

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the FreeBSD Project.

APPENDIX II: RELEASE RATE (1993-2003)

Version	Release date	Days since last release
1.0	1.11.1993	-
1.1	6.5.1994	186
1.1.5	29.6.1994	54
1.1.5.1	5.7.1994	6
2.0	22.11.1994	140
2.0.5	10.7.1995	230
2.1	19.11.1995	132
2.1.5	15.7.1996	239
2.1.6	16.11.1996	124
2.1.6.1	26.11.1996	10
2.1.7	20.2.1997	86
2.2	16.3.1997	24
2.1.7.1	19.3.1997	3
2.2.1	25.4.1997	37
2.2.2	16.5.1997	21
2.2.5	22.8.1997	98
2.2.6	25.3.1998	215
2.2.7	22.7.1998	119
3.0	16.10.1998	86
2.2.8	29.11.1998	44
3.1	15.2.1999	78

3.2	17.5.1999	91
3.3	17.9.1999	123
3.4	20.12.1999	94
4.0	14.3.2000	85
3.5	24.6.2000	102
4.1	27.7.2000	33
4.1.1	27.9.2000	62
4.2	22.11.2000	55
4.3	20.4.2001	150
4.4	20.9.2001	153
4.5	29.1.2002	131
4.6	15.7.2002	167
4.6.2	15.8.2002	31
4.7	10.10.2002	56
5.0	19.1.2003	101

Source: FreeBSD Project
<http://www.freebsd.org/releases/>

APPENDIX III: COMMITTERS ADDED AND REMOVED PER MONTH (2000-2003)

Month	# committers added	# committers removed
Jan 2003	4	2
Dec 2002	0	1
Nov 2002	1	0
Oct 2002	8	0
Sep 2002	2	3
Aug 2002	6	0
Jul 2002	2	0
Jun 2002	4	2
May 2002	2	1
Apr 2002	8	5
Mar 2002	5	0
Feb 2002	3	1
Jan 2002	0	0
Dec 2001	3	0
Nov 2001	10	2
Oct 2001	3	0
Sep 2001	0	0
Aug 2001	5	2
Jul 2001	5	0
Jun 2001	7	0

May 2001	1	0
Apr 2001	4	0
Mar 2001	6	0
Feb 2001	2	0
Jan 2001	3	0
Dec 2000	4	0
Nov 2000	7	0
Oct 2000	6	2
Sep 2000	1	0
Aug 2000	3	0
Jul 2000	10	3
Jun 2000	5	0
May 2000	2	0
Apr 2000	1	0
Mar 2000	2	0
Feb 2000	4	0
Jan 2000	4	0
Jan 2000 – Jan 2003	*142*	*24*

Source: FreeBSD Project <http://www.freebsd.org/cgi/cvsweb.cgi/CVSROOT-src/access>

APPENDIX IV: CORE DEVELOPERS SURVEY

As our analysis of descriptive statistics in chapter 8 shows, the historical expansion of the committers group did not have a negative effect on the performance of core developers, as reflected in the number of commits made by the ten most active committers per year. This of course implies that either core developers spend more time on the project over time or their work is not subject to such increased coordination costs as Brooks' Law suggests. To find out, we did a survey: we identified the 58 committers that populated the ranks of the top ten committers over time and, except for five of them for whom we could not find a valid email address, sent them the below email questionnaire, designed to find out whether the amount of time they spend on the project increases over time and to what extent that is due to non-coding tasks (e.g. time spent on coordinating):

Email Questionnaire

Dear [name of FreeBSD committer],

I contact you in the context of a research project at Delft University of Technology which examines the organisation of FreeBSD development. By analysing CVS logs from 1994 until 2007, we have identified you as one of the 58 most prolific committers in that period and would like to ask you (a) whether the amount of time you spent on the project increased over the years until 2007 and (b) if yes, whether that was due to non-coding activities (e.g. time spent on coordinating).

A simple yes or no suffices for our needs; however, should you feel the urge to elaborate on your answer, please feel free to do so.

Thank you in advance,
george

The questionnaires were sent from 9 January 2012 until 16 January 2012 and 28 replies were collected by 26 January 2012, amounting to a 52.8% response rate.

Analysis

To quantify the survey results, we classified the replies we collected into five categories, which encompass the range of possible answers to the questionnaire:

(1) Yes, time spent on the project increased over time due to non-coding tasks
(2) Yes, time spent on the project increased over time but not due to non-coding tasks
(3) Yes, time spent on the project increased over time but not due to non-coding tasks, though these increased as well
(4) No, time spent on the project decreased over time
(5) Varied: activity tends to ebb and flow over time

Results

This classification method gives the following results:[176]

Answer	*Replies*
(1) Yes, time spent on the project increased over time due to non-coding tasks	7
(2) Yes, time spent on the project increased over time but not due to non-coding tasks	5
(3) Yes, time spent on the project increased over time but not due to non-coding tasks, though these increased as well	3
(4) No, time spent on the project decreased over time	7
(5) Varied: activity tends to ebb and flow over time	6

Of the 28 core developers who responded to the survey, 15 said that the time they spend on the project has increased over the years; 7 said the opposite; while 6 maintained that the extent of their participation tends to ebb and flow over time. Of the 15 core developers who answered that they spend more time on the project

176 As a test of robustness, we asked two colleagues to repeat this procedure in order to check whether different researchers would obtain the same results when applying the above classification schema to the survey replies. The results of their classifications were similar to the ones we obtained.

over time, two-thirds claim that this is due – wholly or partly – to increased non-coding tasks. Of the 13 core developers whose extent of participation manifests a tendency to fluctuate or decline over time, 4 remarked that their peak activity periods were/are associated with increased non-coding tasks. Hence, according to half of the core developers surveyed, periods of peak activity are directly or indirectly related to increased non-coding tasks.

Collected replies

#	Committer	Reply	Result
1	jkh	I effectively left the project (after co-founding it in 1992) in 2002, so anything after that period would have been fairly minimal. The reason I left was due to joining Apple in 2001. I simply don't have time for external projects with so much to do for my employer anymore! :)	4
2	rgrimes	My time spent on the project was pretty well constant so neither yes or no is a proper answer. However the amount of my time spent coding vs coordinating in general increased as the project moved forward. Do realize that I was the ``committer'' that created the repository, and spear headed the core team for the projects first years, so my non coding activies where already adnormally high.	4
3	nate	Like anything in life, there is no simple answer. As one of the three founders, I was very involved in the early years. However, from 2000 to 2007 my involvement increased and decreased as my interest and availability increased and decreased. However, in the latter years, I was less and less involved as family and life made it more difficult to be as involved as I was in the early days.	5
4	ache	a) No, it was slowly decreased. As the time passes I become more and more dissatisfied with the quality of inter-project communucations, first of all from programmer's point of view and, in some extent, from human's point of view too. Now I treat the start of the project as the best time.	4
5	bde	>I contact you in the context of a research project at Delft >University of Technology which examines the organisation of >FreeBSD development. By analysing CVS logs, we have identified >you as one of the 58 most prolific committers over time That was long ago. I haven't committed a single thing for over 3 years, but still spend too much time on this.	4

		> and would like to ask you (a) whether the amount of time you >spend on the project has increased over time and (b) if yes, >whether that is due to non-coding activities (e.g. time spent on >coordinating). I now try to spend less than 1 hour a day on FreeBSD (takes half an hour just to read mail on a quiet day), but sometimes spend 8-20. I only manage to always ignore anything related to management and most non-techical things. > A simple yes or no suffices for our needs; however, should you >feel the urge to elaborate on your answer, please feel free to do so. One grows old and should do something different :-).	
6	phk	I need to qualify my answer a little bit: If I take your question as of my "semi-retirement" point, the answer is a resounding "YES" and "YES". For the period after my "semi-retirement" the answer would be "no" and "no". One of the main drivers for my "semi-retirement" was that I spent far too much time on non-coding activities, and far too much on what you charitably calls "coordinating". So depending on what you are trying to find out, you may need to use one or the other reply, but it's probably the first you're looking for. >By the way, which year was the first of your 'semi-retirement'? I don't think there was a sharp cut-off (that's the cause for the "semi-"), but 2007 i certainly in my active period.	1
7	csgr	The main time during which I was active on the FreeBSD project was during 1993 and 1994, when I was working on my MSc in Computer Science at Rhodes University, in Grahamstown, South Africa. My work on the FreeBSD project probably started out of necessity, as I was using initially 386BSD and then FreeBSD as the platform for my Masters research project. Due to instability issues, which I encountered, I became involved in initially submitting patches and then larger pieces of work. This led to me joining the FreeBSD core team in 1994. After I completed my MSc at the end of 1994, unfortunately, as if often the case, my ability to contribute time to the FreeBSD project decreased due to work demands and lack of connectivity, since I no longer had direct leased line Internet connectivity.	4
8	markm	(a) No (b) N/A Work pressures and other activities have placed a limit on my FreeBSD time.	4

9	julian	from 1993 to 1995 I was just doing FreeBSD work as a hobby but I was working professionally on MACH/bsd which was to some extent "code compatible" so I was payed to develop code which could be put into FreeBSD.. (and I did) e.g. the first scsi system. from 1996 to 1999 I was payed to do development directly on FreeBSD and netgraph, divert sockets and some other code came from that period. In 2000/2001 I took a year sabbatical and in that time threaded the kernel (created kernel threads) From 2001 to 2006 I worked USING freebsd but not really doing much development. Some fixing of bugs as I found them at work. (e.g. in USB). from 2006-2009 I was employed to do some network development, some of which found its way into the system.My current work does not involve any FreeBSD development though I use it. So, generally with the exception of 2000-2001 the amount of activity you see from me depends pretty much on my employment. I started a family in 2002 so free time after that almost completely vanished.	5
10	pst	a) It increased, then decreased, I was most prolific from 1993-1998, then got busy with other work and started sponsoring projects for FreeBSD as part of the company I started, so those commits did not show up in the logs. b) non-coding activities	1
11	brian	In answer to (a), I'm afraid it has decreased over time due to other commitments - both work and family. I am still very much an in-depth user of FreeBSD, but don't get much time to further it's development lately. If and when my time-on-the-project increases, it will be primarily coding activities. I've never been much of a manager, sticking always to technical positions at work, so coding is my only real forte... >....whether, since the first year you committed code, you tended >to spend more time on the project over the years until 2007? I guess the real answer is that I have a big interest in FreeBSD but have difficulty committing time. From 1996 when I first became a committer, 'till around 2001 I was most active. From 2001-2005 I was quite inactive due to work & family commitments, and my time doing FreeBSD specific stuff deteriorated. From 2005 'till 2010 things picked up due to my working directly with FreeBSD, although my activity didn't reach the levels they were in my earlier years. Through 2011 things deteriorated again due to two job changes. I still have a big interest in FreeBSD, so I expect my involvement to	5

		increase again in the future... >..were those activity peaks (from 1996 until 2001 and from 2005 >until 2010) accompanied by increased non-coding activities (i.e. >time spent on coordinating)? The first was purely coding. The second was a mixture of coding and non-coding as a project lead developer.	
12	dfr	Simple answer: No. To elaborate, my involvement with the project tends to go in cycles depending on free time, personal interest and other factors.	5
13	hm	- yes, the time spent increased over the years (i think it's normal, because one realizes that one can "move" things forward) - no, it was not due to coordinating although coordinating activities increased over the years because more people started to help	3

14	jhb	For (a) I would say that, yes, my time has increased. I was not really active as a committer until 1999 or so and became more active through 2000. I have probably maintained approximately the same level of activity since the last half of 2000 up through now however. (There might have been a spike in terms of commit count in late 2000 / early 2001, but back then a single logical change was split up into many separate commits. My workflow since about 2002 or so has changed such that I tend to commit larger logical changes as a single commit.)
		
		For (b), I'm not sure entirely what you are asking. I still spend a lot of time on FreeBSD working on code, but I also spend time answering questions on mailing lists, and I have served several terms on the governing body (core@), as well as worked in the release engineering team for several years during that span. I would say that the amount of time I spend on non-coding activities probably followed a similar pattern to my overall involvement of ramping up from 1999 through 2001, but holding relatively steady since then.
			3

15	jasone	a) Yes. b) No, it was due to spending more time coding.	2
16	imp	Yes. > (b) if yes, whether that was due to non-coding activities (e.g. time >spent on coordinating). Yes. As I did more in the project, I got pulled into many disputes that weren't just about code...	1
17	des	It has varied over time based on a number of factors. My most active	5

		period was probably 2001-2005, but I've had peaks of varying lengths since then as well; I'm currently in an active phase. I was heavily involved in non-coding activities during that period, but that was only one of several reasons for my increased activity level.	
18	alfred	I would say that the amount of time I've spent on the project has declined. Some of this is due to coordinating other developers as I introduced and mentored a few developers, but this is also due to other interests outside of FreeBSD such as work and other hobbies. I still contribute, but at a much lower less significant rate than in the early to mid 2000s. > If I may ask one follow-up question, was that period of peak >activity (from the early to mid 2000s) accompanied by increased >non-coding activities (i.e. time spent on coordinating)? Yes, I would say that I did spend more time mentoring and helping other people bring code into FreeBSD.	4
19	marcel	a) I don't think so, but then again I haven't kept track of what I've done. I can easily be mistaken.	4
20	davidxu	a : yes b : no, it was due to increased the tasks in the project.	2
21	trhodes	Yes on increased workload (a); somewhat on non-coding (b).	1
22	njl	I started with a single side project (SCSI CAM target mode driver) that was relatively easy to get integrated into the tree. It didn't involve any changes to the core of the kernel. A committer mentored me for this work and explained the rules of the project. I expanded out from there to fix USB mass storage bugs. This started more interaction with developers because the process for handling bug reports needed work. I then moved over to power management and ACPI in 2003 in order to fix my laptop. This is when the most commits started happening. I wrote some major subsystems (cpufreq driver) that had to integrate with core APIs. So I had to discuss such changes with other developers. Mostly these discussions would be in private or small lists to get a general plan. Once a plan was ready to propose, I'd email it to the public lists for a critique. I think the increased time spent on the project was related to the scope of projects I tackled. As they grew bigger, the need to deal with core subsystems and all the developers they affect was the major coordination difficulty. But it wasn't that bad. The harder part was answering, getting debugging assistance, and solving problems for users on the mailing list. Because of the wide variety of hardware	3

		out there and the difficulty of debugging by proxy, it was hard to solve some issues for them.	
23	ru	(a): yes (b): no	2
24	harti	the time increase was due to coding directly for FreeBSD and coding for FreeBSD-related stuff in the organizations I worked for. I tried to reduce non-coding activities as much as possible.	2
25	brooks	Off hand, I'd say yes to both questions. I was elected to core in 2006 so that increased time spent on non-coding activities.	1
26	glebius	I started to contribite to the project in 2003, and got committer status in 2004. For a first couple of years I was very active in CVS. But later my activity had abated. The reason for that were mostly due to my personal time management failure and a temporary loss of motivation. Then in 2007 I became daddy and personal time got even more limited. During 2007, 2008, 2009 my presence in the project was small. The last year I have optimised my personal time management and motivation and I am back to spending a lot of time for the project and checking in a lot to SVN. I hope that birth of our second child would not speed down my committing activity a lot. :) I have never spent my time on coordinating. Probably because I am too far geographically, and I don't travel a lot. Only in 2011 I have visited EuroBSDCon, and recently have tried to make mini-conference for local (Russian and Ukrainian) committers.	5
27	mjacob	a) Yes b) No	2
28	kmacy	(a) Yes. >(b) if yes, whether that was due to non-coding activities (e.g. time >spent on coordinating). Coordination, but not in the sense of management but in the sense of more time spent trying to reach consensus on changes that had more far-reaching impact than those localized to device drivers or a less used architecture.	1

APPENDIX V: BIBLIOGRAPHICAL REFERENCES

Abdel-Hamid T.K. 1989. The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach, *IEEE Transactions on Software Engineering* 15 (2): 109-119

Abelson R.P. 1995. *Statistics as Principled Argument*. Lawrence Erlbaum Associates

Adams J., Capiluppi A., Boldyreff C. 2009. Coordination and Productivity Issues in Free software: the role of Brooks' Law. *Proceedings of ICSM.* Edmonton, Canada, pp. 319-328

Albrecht, A., Gaffney J. 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions of Software Engineering* 9 (6): 639-648

Alexander C. 1964. *Notes on the Synthesis of Form.* Harvard University Press

Anderson J.J. 1984. David tells Ahl – the history of Creative Computing. *Creative Computing* 10 (11): 66-68

Anderson P. 1983. Decision Making by Objection and the Cuban Missile Crisis. *Administrative Science Quarterly* 28 (2): 201-222

Andrews J. 2008. FreeBSD: Tinderbox Failures. *KernelTrap* (Feb. 28), accessible online at <http://kerneltrap.org/node/595>

Argyres N.S. 1999. The Impact of Information Technology on Coordination: Evidence from the B-2 "Stealth" Bomber. *Organization Science* 10 (2): 162-180

Ashby W.R. 1960. *Design for a Brain: the origin of adaptive behavior.* Wiley

Asterisk News. 2004, 'Interview With FreeBSD Developer'. *Asterisk News.* Accessible online at <http://www.venturevoip.com/news.php?rssid=95>

Babbage C. 2009. *On the Economy of Machinery and Manufactures*. Cambridge University Press

Baldwin C.Y., Clark K.B. 2006a. 'Modularity in the Design of Complex Engineering Systems', in A. Minai, D. Braha & Y.B. Yam (Eds.) *Complex Engineered Systems: Science Meets Technology.* New England Complex Systems Institute Series on Complexity. Springer-Verlag

Baldwin C.Y., Clark K.B. 2006b. The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? *Management Science.* 52 (7): 1116-1127

Baldwin C.Y., Clark K.B. 2000. *Design Rules: The Power of Modularity.* MIT Press

Banker R.D. 1984. Estimating most productive scale size using data envelopment analysis. *European Journal of Operational Research* 17 (1): 35-44

Banker R.D., Chang H., Kemerer C.F. 1994. Evidence on economies of scale in software development. *Information and Software Technology* 36 (5): 275-282

Banker R.D., Slaughter S.A. 2000. The Moderating Effect of Structure on Volatility and Complexity in Software Enhancement. *Information Systems Research* 11 (3): 219-240

Banker R.D., Slaughter S.A. 1997. A Field Study of Scale Economies in Software Maintenance. *Management Science* 43 (12): 1709-1725

Barker J.R. 1993. Tightening the Iron Cage: Concertive Control in Self-Managing Teams. *Administrative Science Quarterly* 38 (3): 408-437

Barley S.R. 1996. Technicians in the Workplace: Ethnographic Evidence for Bringing Work into Organizational Studies. *Administrative Science Quarterly* 41 (3): 404-441

Belady L.A., Lehman M.M. 1976. A model of large program development. *IBM*

Systems Journal 15 (3): 225 - 252

Bell S. 2002. *Economic Governance and Institutional Dynamics*. Oxford University Press

Bendor J., Moe T., Shotts K. 2001. Recycling the Garbage Can: An Assessment of the Research Program. *The American Political Science Review* 95 (1): 169-190.

Benkler Y. 2006. *The Wealth of Networks: How Social Production Transforms Markets and Freedom.* Yale University Press

Bernstein D. 2011. Lecture notes on 'An Introduction to Functions/Methods and Modularity', Computer Science Department, James Madison University. Accessible online at <https://users.cs.jmu.edu/bernstdh/web/common/lectures/slides_functions-and-methods.php>

den Besten M., Dalle J.M., Galia F. 2008. The allocation of collaborative effort in open-source software. *Information Economics and Policy* 20 (4): 316-322

den Besten M., Dalle J.M., Galia F. 2006. 'Collaborative Maintenance in Large Open-Source Projects', in E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto & G. Succi (Eds.) *Open Source Systems*. IFIP International Federation for Information Processing, Vol. 203, Boston: Springer, pp. 233-244

Biancuzzi F. 2004. 'Behind DragonFlyBSD'. *ONLamp.com*. Accessible online at <http://onlamp.com/lpt/a/4991>

Blackburn J.D., Lapre M.A., Van Wassenhove L.N. 2006. *Brooks' Law Revisited: Improving Software Productivity by Managing Complexity*. Accessible online at <http://papers.ssrn.com/sol3/papers.cfm?abstract_id=922768>

Blackburn J.D., Scudder G.D. 1996. Improving Speed and Productivity of Software Development: A Global Survey of Software Developers. *IEEE Transactions on Software Engineering* 22 (12): 875-886

Blau P.M. 1970. A Formal Theory of Differentiation in Organizations. *American Sociological Review* 35 (2): 201-208

Boehm B.W. 1987. Improving Software Productivity. *Computer* 20 (8): 43-58

Boehm B.W. 1981. *Software Engineering Economics.* Prentice-Hall

Bolici F., Howison J., Crowston K. 2009. 'Coordination without discussion? Socio-technical congruence and Stigmergy in Free and Open Source Software projects'. Paper presented at *2Nd International Workshop on Socio-Technical Congruence, ICSE,* Vancouver, Canada, May 19

Le Bon G. 2002. *The Crowd: A Study of the Popular Mind*. Dover Publications

Bowman L. 1998. *Conceptual architecture of the Linux kernel*. Accessible online at <http://plg1.cs.uwaterloo.ca/~itbowman/CS746G/a1/>

Bradner S. 1999. 'The Internet Engineering Task Force', in C. Dibona & S. Ockman (Eds.) *Open Sources: Voices from the Open Source Revolution*. O'Reilly

Brooks F.P. 1995. *The Mythical Man-Month.* Addison-Wesley

Brusoni S., Marengo L., Prencipe A., Valente M. 2007. The value and costs of modularity: a problem-solving perspective. *European Management Review* 4 (2): 121-132

Brusoni S., Prencipe A. 2006. Making Design Rules: A Multidomain Perspective. *Organization Science* 17 (2): 179-189

Brusoni S., Prencipe A. 2001. Unpacking the Black Box of Modularity: Technologies, Products and Organizations. *Industrial and Corporate Change* 10 (1): 179-205

Brusoni S. 2005. The Limits to Specialization: Problem Solving and Coordination in 'Modular Networks'. *Organization Studies* 26 (12): 1885-1907

BSD Certification Group. 2005. *BSD Usage Survey* (Oct.). Accessible online at <http://www.bsdcertification.org/downloads/pr_20051031_usage_survey_en_en.pdf>

Cain J.W., McCrindle R.J. 2002. 'An Investigation into the Effects of Code Coupling on Team Dynamics and Productivity'. *COMPSAC, 26th Annual International Computer software and Applications Conference*, Oxford, England

Canback S., Samouel P., Price D. 2006. Do diseconomies of scale impact firm size and performance? A theoretical and empirical overview. *Journal of Managerial Economics* 4 (1): 27-70

Canetti E. 1984. *Crowds and Power*. Farrar, Straus and Giroux

Capiluppi A., Adams P.J. 2009. 'Reassessing Brooks' Law for the Free Software Community', in C. Boldyreff et al. (Eds.) OSS 2009, IFIP AICT 299, pp. 274-283

Capra E. 2008. *Software Design Quality and Development Effort: An Empirical Study on the Role of Governance in Open Source Projects.* PhD Dissertation, Politecnico di Milano

Capra E., Francalanci C., Merlo F. 2008. An Empirical Study on the Relationship among Software Design Quality, Development Effort, and Governance in Open Source Projects. *IEEE Transactions on Software Engineering* 34 (6): 765-782

Card D.N. 2006. The Challenge of Productivity Measurement. *Proceedings of the Pacific Northwest Software Quality Conference*, Portland, USA. Accessible online at <http://www.compaid.com/caiinternet/ezine/card-prod.pdf>

Carpenter B. 1996. *Architectural Principles of the Internet*. IETF Network Working Group. Accessible online at <https://www.ietf.org/rfc/rfc1958.txt>

Chalmers R. 2000. 'The unknown hackers'. *Salon* (May 17). Accessible online at <http://www.salon.com/technology/feature/2000/05/17/386bsd/index.html>

Chamberlin J. 1974. Provision of Collective Goods As a Function of Group Size.

The American Political Science Review 68 (2): 707-716

Chen X., Ender P., Mitchell M., Wells C. 2003. *Regression with Stata*. Accessible online at <http://www.ats.ucla.edu/stat/stata/webbooks/reg/default.htm> .

Chesbrough H. 2003. *Open Innovation: The new imperative for creating and profiting from technology*. Harvard Business School Press

Clark, D.D. 1992. 'A cloudy crystal ball: Visions of the future'. Plenary presentation at 24th meeting of the Internet Engineering Task Force,Cambridge, Mass.,13-17 July

Clark K. 1985. The interaction of design hierarchies and market concepts in technological evolution. *Research Policy* 14 (5): 235-251

Coase R.H. 1937. The Nature of the Firm. *Economica* 4 (16): 386-405

Cohen M, March J., Olsen J. 1972. A Garbage Can Model of Organizational Choice. *Administrative Science Quarterly* 17 (1): 1-25

Coleman J.S. 1993. The Design of Organizations and the Right to Act. *Sociological Forum* 8 (4): 527-546

Coleman J.S. 1990. *Foundations of Social Theory*. Belknap Press

Conway M.E. 1968. How Do Committees Invent? *Datamation* 14: 28-31

Corbet J., Kroah-Hartman G. & A. McPherson, 2010. *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*. Linux Foundation White Paper. Accessible online at <https://www.linuxfoundation.org/docs/lf_linux_kernel_development_2010.pdf >

Crowston K., Howison J. 2006. Hierarchy and Centralization in Free and Open Source Software Team Communications. *Knowledge, Technology, & Policy* 18 (4): 65-85

Curtis B., Krasner H., Iscoe N. 1988. A field study of the software design process for large systems. *Communications of the ACM* 31 (11): 1268-1287

Cusumano M., Kemerer C.F. 1990. A quantitative analysis of US and Japanese practice and performance in software development. *Management Science* 36 (11): 1384-1406

Cusumano M., Selby R.W. 1997. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. HarperCollinsBusiness

Daft R., Lewin A. 1993. Where Are the Theories for the "New" Organizational Forms? An Editorial Essay. *Organization Science* 4 (4): i-vi

Dallal G.E. 2001. *The Little Handbook of Statistical Practice*. Accessible online at <http://www.tufts.edu/~gdallal/LHSP.HTM>

Debian. 2007. *Debian Constitution: Constitution for the Debian Project* (v. 1.4). Accessible online at <http://www.debian.org/devel/constitution>

Debian. 1998. *Debian Constitution: Constitution for the Debian Project* (v. 1). Accessible online at <http://www.debian.org/devel/constitution.1.0>

Dhama H. 1995. Quantitative Models of Cohesion and Coupling in Software. *Journal of Systems Software* 29 (1): 65-74

Dillon M. 2003. *Announcing DragonFly BSD!*. Message posted to freebsd-current mailing list (Jul. 16). Accessible online at <http://lists.freebsd.org/pipermail/freebsd-current/2003-July/006889.html>

Dinh-Trong T.T., Bieman J.M. 2005. The FreeBSD Project: A Replication Case Study of Open Source Development. *IEEE Transactions on Software Engineering* 31 (6): 481-494

DSS 2007. *Interpreting Regression Output*. Princeton University (Data and

Statistical Services). Accessible online at <http://dss.princeton.edu/online_help/analysis/interpreting_regression.htm>

Dunbar R. 1993. Co-evolution of neocortex size, group size and language in humans. *Behavioral and Brain Sciences* 16 (4): 681-735

Eisenhardt K. 1989. Building Theories from Case Study Research. *The Academy of Management Review* 14 (4): 532-550

Ensemenger N., Aspray W. 2000. Software as labor process. *Proceedings of the International Conference On History of Computing: Software Issues*, Paderborn, Germany, 5-7 April

Ernst D. 2005. Limits to Modularity: Reflections on Recent Developments in Chip Design. *Industry and Innovation* 12 (3): 303-335

Ethiraj S.K., Levinthal D. 2004. Modularity and Innovation in Complex Systems. *Management Science* 50 (2): 159-173

Feitelson D., Adeshiyan T., Balasubramanian D., Etsion Y., Madl G., Osses E., Singh S., Suwanmongkol K., Xie M., Schash S. 2007. Fine-grain analysis of common coupling and its application to a Linux case study. *Journal of Systems and Software* 80 (8): 1239-1255

Fielding R.T. 1999. Shared Leadership in the Apache Project. *Communications of the ACM* 42 (4): 42-44

Fisher R.A. 1925. *Statistical Methods for Research Workers.* Oliver and Boyd

Foucault M. 1975. *Discipline and Punish: The Birth of the Prison*. Random House

Fox J., Weisberg S. 2011. *An R Companion to Applied Regression* (2Nd ed.) Sage Publications

FreeBSD. 2012. *Release Information*. Accessible online at <http://www.freebsd.org/releases/>

FreeBSD. 2011a. *Committer's Guide*. Accessible online at <http://www.freebsd.org/doc/en_US.ISO8859-1/articles/committers-guide/>

FreeBSD. 2011b. *FreeBSD Handbook*. Accessible online at <http://www.freebsd.org/doc/handbook/>

FreeBSD. 2011c. *New Account Creation Procedure*. Accessible online at <http://www.freebsd.org/internal/new-account.html>

FreeBSD. 2011d. *The FreeBSD Committers' Big List of Rules.* Accessible online at <http://www.freebsd.org/doc/en_US.ISO8859-1/articles/committers-guide/rules.html>

FreeBSD. 2011e. *FreeBSD Project Administration and Management*. Accessible online at <http://www.freebsd.org/administration.html>

FreeBSD. 2011f. *Frequently Asked Questions for FreeBSD 6.X, 7.X and 8.X*. Accessible online at <http://www.freebsd.org/doc/en_US.ISO8859-1/books/faq/introduction.html>

FreeBSD. 2011g. *Tinderbox*. Accessible online at <http://wiki.freebsd.org/Tinderbox>

FreeBSD. 2011i. *Maintainers file* (v. 1.166). Accessible online at <http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/MAINTAINERS?>

FreeBSD. 2010a. *Developers' Handbook*. Accessible online at <http://www.freebsd.org/doc/en/books/developers-handbook/>

FreeBSD. 2010b. *Release Engineering Information*. Accessible online at <http://freebsd.unixtech.be/releng/>

FreeBSD. 2010c. 'Kernel source file style guide', in *FreeBSD Kernel Developer's Manual*. Accessible online at <http://www.freebsd.org/cgi/man.cgi?query=style&sektion=9&manpath=FreeBSD+9-current>

FreeBSD. 2010d. *Contributors to FreeBSD (v 1.452)*. Accessible online at <http://www.freebsd.org/doc/en/articles/contributors/>

FreeBSD. 2009. *Additional Contributors*. Accessible online at <http://www.jp.freebsd.org/cgi/cvsweb.cgi/~checkout~/doc/en_US.ISO8859-1/articles/contributors/contrib.additional.sgml?rev=1.886>

FreeBSD. 2008. *Additional Contributors.* Accessible online at <http://docs.freebsd.org/doc/7.3-RELEASE/usr/share/doc/en/articles/contributors/contrib-additional.html>

FreeBSD. 2007. *Additional Contributors*. Accessible online at <http://docs.freebsd.org/doc/7.0-RELEASE/usr/share/doc/en/articles/contributors/contrib-additional.html>

FreeBSD. 2006. *Contributors to FreeBSD* (v. 1.448). Accessible online at <http://www.pl.freebsd.org/doc/en/articles/contributors/>

FreeBSD. 2005. *Additional Contributors*. Accessible online at <http://docs.freebsd.org/doc/6.0-RELEASE/usr/share/doc/en/articles/contributors/contrib-additional.html>

FreeBSD. 2004. *Additional Contributors*. Accessible online at <http://docs.freebsd.org/doc/5.4-RELEASE/usr/share/doc/en/articles/contributors/contrib-additional.html>

FreeBSD. 2003. *Contributors to FreeBSD* (v. 1.421). Accessible online at <http://docs.freebsd.org/doc/5.2-RELEASE/usr/share/doc/en/articles/contributors/>

FreeBSD. 2002. *Additional Contributors.* Accessible online at <http://docs.freebsd.org/doc/4.6.2-RELEASE/usr/share/doc/en/articles/contributors/contrib-additional.html>

FreeBSD. 2001a. *Contributors to FreeBSD* (v. 1.17). Accessible online at

<http://docs.freebsd.org/doc/4.4-RELEASE/usr/share/doc/en/articles/contributors/>

FreeBSD. 2001b. *Quarterly Status Report* (June). Accessible online at <http://www.freebsd.org/news/status/report-2001-06.html>

FreeBSD. 2000. *Core Bylaws*. Accessible online at <http://www.freebsd.org/internal/bylaws.html>

FreeBSD. 2000a. *Additional Contributors*. Accessible online at <http://docs.freebsd.org/doc/3.5-RELEASE/usr/share/doc/handbook/contrib-additional.html>

FreeBSD. 1999. *New Committer Guide*. Accessible online at <http://docs.freebsd.org/doc/4.0-RELEASE/usr/share/doc/en/articles/committers-guide/>

FreeBSD. 1999a. *Additional Contributors*. Accessible online at <http://docs.freebsd.org/doc/3.4-RELEASE/usr/share/doc/handbook/contrib-additional.html>

FreeBSD. 1998. *Additional Contributors*. Accessible online at <http://docs.freebsd.org/doc/2.2.6-RELEASE/usr/share/doc/handbook/handbook258.html#598>

FreeBSD. 1997. *Additional Contributors*. Accessible online at <http://docs.freebsd.org/doc/2.2.2-RELEASE/usr/share/doc/handbook/handbook328.html>

FreeBSD. 1996. 'Source Tree Guidelines and Policies', in *FreeBSD Handbook*. Accessible online at <http://doc.ctrlaltdel.ch/freebsd/handbook/policies.html#POLICIES-MAINTAINER>

FreeBSD. 1996a. *Additional Contributors*. Accessible online at <http://docs.freebsd.org/doc/2.1.5-

RELEASE/usr/share/doc/handbook/handbook.html>

FreeBSD Foundation. 2011. *About the FreeBSD Foundation*. Accessible online at <http://www.freebsdfoundation.org/about.shtml>

Frenken K. 2006. A fitness landscape approach to technological complexity, modularity, and vertical disintegration. *Structural Change and Economic Dynamics* 17 (3): 288-305

Freud S. 1975. *Group Psychology and the Analysis of the Ego*. W.W. Norton & Company

Galvin G, Morkel A. 2001. The Effect of Product Modularity on Industry Structure: The Case of the World Bicycle Industry. *Industry and Innovation* 8 (1): 31-47

Gancarz M. 1995. *The UNIX Philosophy*. Butterworth-Heinemann

Garud R., Kumaraswamy A. 1995. Technological and Organizational Designs for Realizing Economies of Substitution, *Strategic Management Journal* 16 (S1): 93-109

Garzarelli G., Galoppini R. 2003. *Capability Coordination in Modular Organization: voluntary FS/OSS Production and the Case of Debian GNU/Linux*. Industrial Organization 0312005, EconWPA. Accessible online at <http://ideas.repec.org/p/wpa/wuwpio/0312005.html>

Gauthier R., Pont S. 1970. *Designing Systems Programs.* Prentice-Hall

Gershenson, Prasad and Zhang. 2003. Product modularity: definitions and benefits. *Journal of Engineering Design* 14 (3): 295-313

Ghosh R.A. 2005. 'Understanding Free Software Developers: Findings from the FLOSS Study', in J. Feller, B. Fitzgerald, S.A. Hissam & K.R. Lakhani (Eds.) *Perspectives on Free and Open Source Software*. MIT Press, pp. 23-45

Ghosh R.A. 2003. Clustering and Dependencies in Free/Open Source Software

Development: Methodology and Tools. *First Monday* 8 (4). Accessible online at <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1041/962>

Ghosh R.A., David P.A. 2003. *The nature and composition of the Linux kernel developer community: a dynamic analysis*. SIEPR-Project Nostra WP. Accessible online at <http://arno.unimaas.nl/show.cgi?fid=16233>

Gibson M.L. 1991. Public Goods, Alienation, and Political Protest: The Sanctuary Movement as a Test of the Public Goods Model of Collective Rebellious Behavior. *Political Psychology* 12 (4): 623-651

Giddens A. 1988. *Capitalism and modern social theory: An analysis of the writings of Marx, Durkheim and Max Weber*. Cambridge University Press

Giuri P., Rullani F., Torrisi S. 2008. Explaining leadership in virtual teams: The case of open source software. *Information Economics and Policy* 20 (4): 305-315

Goldstone J.A. 1994. Is Revolution Individually Rational? Groups and Individuals in Revolutionary Collective Action. *Rationality and Society* 6 (1): 139-166

Gomes P.J., Joglekar N.R. 2008. Linking Modularity with Problem Solving and Coordination Efforts. *Managerial and Decision Economics* 29 (5): 443-457

Gouldner A.W. 1955. Metaphysical Pathos and the Theory of Bureaucracy. *American Political Science Review* 49 (2): 496-507

Granger C.W. 1969. Investigating Causal Relations by Econometric Models and Cross-Spectral Methods. *Econometrica* 37: 424-438

Granovetter M. 1984. Small is Bountiful: Labor Markets and Establishment Size. *American Sociological Review* 49 (3): 323-334

Gray S.B. 1984. The early days of personal computers. *Creative Computing* 10 (11): 6-14

Hamel G. 2007. *The Future of Management*. Harvard Business School Press

Hardt M. 2006. 'Love in the Multitude', in J.T. Schnapp & M. Tiews (Eds.) *Crowds*. Stanford University Press

Hardt M., Negri A. 2004. *Multitude: War and Democracy in the Age of Empire*. Penguin Putnam

Hardt M., Negri A. 2000. *Empire*. Harvard University Press

Harrison P.M. 1960. Weber's Categories of Authority and Voluntary Associations. *American Sociological Review* 25 (2): 232-237

Hauben R. 1991. Computers for the People – A History or how hackers gave birth to the personal computer, Part II. *The Amateur Computerist* 4 (1): 1-5

Henderson R.M., Clark K.B. 1990. Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. *Administrative Science Quarterly* 35 (1): 9-30

Himanen P. 2001. *The Hacker Ethic and the Spirit of the Information Age*. Vintage

von Hippel E. 2005. *Democratizing Innovation*. MIT Press

von Hippel E. 1990. Task partitioning: An innovation process variable. *Research Policy* 19 (5): 407-418

Hirschman A. 1970. *Exit, Voice, and Loyalty: Responses to Decline in Firms, Organizations, and States*. Harvard University Press

Hoetker G. 2006. Do modular products lead to modular organizations? *Strategic Management Journal* 27 (6): 501-518

Hoffman P. 2010. *The Tao of IETF: A Novice's Guide to the Internet Engineering Task Force*. Internet Engineering Task Force. Accessible online at <https://www.ietf.org/tao.html>.

Holck J., Jørgensen N. 2004. 'Do Not Check in on Red: Control Meets Anarchy in Two Open Source Projects', in S. Koch (Ed.) *Free/Open Source Software Development*. Idea Group

Holck J., Jørgensen N. 2003/2004. Continuous integration and quality assurance: a case study of two open source projects. *Australasian Journal of Information Systems*, Special Issue (2003/2004): 40-53

Holmstrom B. 1982. Moral Hazard in Teams. *The Bell Journal of Economics* 13 (2): 324-340

Howard J. 2001. 'The BSD Family Tree'. *Daemon News* (April). Accessible online at <http://www.freenix.no/arkiv/daemonnews/200104/bsd_family.html>

Hsia P., Hsu C., Kung D.C. 1999. Brooks' Law Revisited: A System Dynamics Approach. *Proceedings of the 23rd Intern. Comp. Software and Applications Conf. (COMPSAC)*, IEEE Computer Society, pp. 370-375

Hubbard J. 2009. *Contributing to FreeBSD*. Accessible online at <http://www.freebsd.org/doc/en_US.ISO8859-1/articles/contributing/index.html>

Hubbard J. 2002. 'Foreword', in M. Lucas, *Absolute BSD: The Ultimate Guide to FreeBSD*. No Starch Press

Hubbard J. 1998a. Editorial: Pulling on one end of the rope. *Freshmeat.* Accessible online at <http://freshmeat.net/articles/editorial-pulling-on-one-end-of-the-rope>

Hubbard J. 1998b. What Is FreeBSD? *Performance Computing.* Accessible online at <http://bbs.unix-like.org:8080/boards/Server_DOC/M.1006795908.A>

Hubbard J. 1997. *My resignation as president of the FreeBSD Project*. Message posted to freebsd-announce mailing list (Feb. 5). Accessible at <http://lists.freebsd.org/pipermail/freebsd-announce/1997-

February/000305.html>

IEEE 1993. Standard for Software Productivity Metrics, IEEE Std. 1045-1992. *IEEE Standards Board*.

Ingham A.G., Levinger G., Graves J., Peckham V. 1974. The Ringelmann Effect: Studies of Group Size and Group Performance. *Journal of Experimental Social Psychology* 10 (4): 371-384

Ishii K., Juengel C., Eubanks C.F. 1995. Design for product variety: key to product line structuring. *Proceedings of the 1995 ASME Design engineering Technical Conferences–7th International Conference on Design theory and Methodology*, Boston, MA (NY: The American Society of Mechanical Engineers)

Jablan S. 1995. *Theory of Symmetry and Ornament*. Mathematical Institute, Belgrade

Jacobs M., Vickery S.K., Droge C. 2007. The effects of product modularity on competitive performance. *International Journal of Operations & Production Management* 27 (10): 1046-1068

Jones T.C. 1978. Measuring Programming Quality and Productivity. *IBM Systems Journal* 17 (1): 39-63

Jørgensen N. 2007. Developer autonomy in the FreeBSD open source project. *Journal of Management & Governance* 11 (2): 119-128

Jørgensen N. 2005. 'Incremental and decentralized integration in FreeBSD', in J. Feller, B. Fitzgerald, S.A. Hissam & K.R. Lakhani (Eds.) *Perspectives on Free and Open Source Software*. MIT Press, pp. 227-244

Jørgensen N. 2001. Putting it all in the trunk: incremental software development in the FreeBSD open source project. *Information Systems Journal* 11 (4): 321-336

Jones P. 2000. Brooks' Law and open source: The more the merrier? *IBM Developer Works* (May 1). Accessible online at

<http://pascal.case.unibz.it/retrieve/3816/merrier.pdf>

Kaldor N. 1934. The Equilibrium of the Firm. *The Economic Journal* 44 (173): 60-76

Karau S.J., Williams K.D. 1993. Social Loafing: A Meta-Analytic Review and Theoretical Integration. *Journal of Personality and Social Psychology* 65 (4): 681-706

Kauffman S., Levin S. 1987. Towards a General theory of Adaptive Walks on Rugged Landscapes. *Journal of Theoretical Biology* 128 (1): 11-45

Kemerer C.F. 1993. Reliability of function points measurement: a field experiment. *Communications of the ACM* 36 (2): 85-97

KernelTrap 2002. 'FreeBSD: Jordan Hubbard Steps Down' (Apr. 30). Accessible online at <http://kerneltrap.org/node/169>

Kerr N.L., Brunn S.E. 1983. The dispensability of member effort and group motivation losses: Free rider effects. *Journal of Personality and Social Psychology* 44 (1): 78-94

Kitchenham B., Mendes E. 2004. Software Productivity Measurement Using Multiple Size Measures. *IEEE Transactions on Software Engineering* 30 (12): 1023-1035

Koch S. 2008. Effort modelling and programmer participation in open source software projects. *Information Economics and Policy* 20 (4): 345-355

Koch S. 2004. Profiling an Open Source Project Ecology and Its Programmers. *Electronic Markets* 14 (2): 77-88

Koren O. 2006. A Study of the Linux Kernel Evolution. *ACM SIGOPS Operating Systems Review* 40 (2): 110-112

Koshy J. 2010 *Building Products with FreeBSD* (v. 1.8). FreeBSD Project. Accessible online at <http://www.freebsd.org/doc/en/articles/building-

products/article.html>

Kravitz D.A., Martin B. 1986. Ringelmann rediscovered: The original article. *Journal of Personality and Social Psychology* 50 (5): 936-941

Krill P. 2011. 'Why the time is now for continuous integration in app development'. *InfoWorld* (July 7). Accessible online at <https://www.infoworld.com/print/165941>

Krishnamurthy S. 2002. Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. *First Monday* 7 (6). Accessible online at <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1477/1392>

Kroah-Hartman G. 2005. *How to do Linux kernel development.* Accessible online at <http://lxr.linux.no/source/Documentation/HOWTO>

von Krogh G., Stuermer M., Geipel M., Spaeth S., Haefliger S. 2009. How component dependencies predict change in complex technologies. *EURAM Conference on Renaissance and Renewal in Management Studies*, May 11-14, Liverpool, UK

de Laat P.B. 2007. Governance of open source software: state of the art. *Journal of Management & Governance* 11 (2): 165-177

LaMantia M., Cai Y., MacCormack A., Rusnak J. 2008. Evolution Analysis of Large-Scale Software Systems Using Design Structure Matrices and Design Rule Theory: Two Exploratory Cases. *WICSA 7th Working IEEE/IFIP Conference on Software Architecture*, pp. 83-92

Langlois R. 2003. The vanishing hand: the changing dynamics of industrial capitalism. *Industrial and Corporate Change* 12 (2): 351-385

Langlois R. 1992. External Economies and Economic Progress: The Case of the Microcomputer Industry. *The Business History Review* 66 (1): 1-50

Langlois R., Garzarelli G. 2008. Of hackers and hairdressers: modularity and the organizational economics of open-source collaboration. *Industry & Innovation* 15 (2): 125–143

Langlois R., Robertson. 1992. Networks and innovation in a modular system: Lessons from the microcomputer and stereo component industries. *Research Policy* 21 (4): 297–313

Latané B., Williams K., Harkins S. 1979. Many hands make light the work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology* 37 (6): 822-832

Lee G., Cole R.E. 2003. From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development. *Organization Science* 14 (6): 633-649

Lehey G. 2003. *The FreeBSD SMP implementation*. Accessible online at <http://www.lemis.com/grog/SMPng/Singapore/slides.pdf>

Lehey G. 2002. *Two years in the trenches: The evolution of a software project*. Accessible online at <http://www.lemis.com/grog/In-the-trenches.pdf>

Lehman M.M. 1980. On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle. *The Journal of Systems and Software* 1: 213-221

Lehman M.M., Ramil J.F. 2001. Rules and Tools for Software Evolution Planning and Management. *Annals of Software Engineering* 11 (1): 15-44

Lehman M.M., Ramil J.F., Wernick P.D., Perry D.E., Turski W.M. 1997. Metrics and Laws of software Evolution–The Nineties View. *Proceedings of 4th International Software Metrics Symposium (METRICS '97)*, IEEE.

Leonard A. 2000. 'BSD Unix: Power to the people, from the code'. *Salon* (May 16). Accessible online at <http://archive.salon.com/tech/fsp/2000/05/16/chapter_2_part_one/index.html>

Levy S. 1984. *Hackers: Heroes of the Computer Revolution*. Anchor Press/Doubleday

Lohmann S. 1994. The Dynamics of Informational Cascades: The Monday Demonstrations in Leipzig, East Germany, 1989-1991. *World Politics* 47 (1): 42-101

Loli-Queru E. 2003. 'Focus on FreeBSD: Interview with the Core Team'. *OSNews* (Apr. 28). Accessible online at <http://www.osnews.com/story/3415>

Long S. 2010. *Perforce in FreeBSD Development*. The FreeBSD Project. Accessible online at <http://www.freebsd.org/doc/en_US.ISO8859-1/articles/p4-primer/article.html>

Losh W. 2006. *Working with Hats*. The FreeBSD Project. Accessible online at <http://www.freebsd.org/doc/en/articles/hats/index.html>

Lucas M. 2002. 'How to Become a FreeBSD Committer'. *ONLamp.com* (Jan. 31). Accessible online at <http://onlamp.com/lpt/a/1492>

MacCormack A., Baldwin C., Rusnak J. 2010. *The Architecture of Complex Systems: Do Core-Periphery Structures Dominate?* Harvard Business School Working Paper 10-059. Accessible online at <http://www.hbs.edu/research/pdf/10-059.pdf>

MacCormack A., Rusnak J., Baldwin C. 2008a. *Exploring the Duality between Product and Organizational Architectures: A Test of the Mirroring Hypothesis*. Harvard Business School Working Paper 08-039. Accessible online at <http://www.hbs.edu/research/pdf/08-039.pdf>

MacCormack A., Rusnak J., Baldwin C. 2008b. *The Impact of Component Modularity on Design Evolution.* Harvard Business School Working Paper 08-038. Accessible online at <http://www.hbs.edu/research/pdf/08-038.pdf>

MacCormack A., Rusnak J., Baldwin C. 2006. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code.

Management Science 52 (7): 1015-1030

MacDuffie J.P., Sethuraman K., Fisher M.L. 1996. Product Variety and Manufacturing Performance: Evidence from the International Automotive Assembly Plant Study. *Management Science* 42 (3): 350-369

Makovicky E. 1989. 'Ornamental Brickwork', in I. Hargittai (Ed.) *Symmetry 2: Unifying Human Understanding,* Pergamon Press

Malone T.W., Crowston K. 1990. What is Coordination Theory and How Can It Help Design Cooperative Work Systems? *CSCW 90 Proceedings*

Mansbridge J.T. 1977. Acceptable Inequalities. *British Journal of Political Science* 7 (3): 321-336

Marshall A. 1891. *Principles of Economics*, *vol. 1* (2nd ed.). MacMillan & Co

Marx K. 1990. *Capital–A Critique of Political Economy, vol. 1*. Penguin

Mateos-Garcia J., Steinmueller E. 2008. The institutions of open source software: Examining the Debian community. *Information Economics and Policy* 20 (4): 333-344

Mazzarella W. 2010. The Myth of the Multitude, or, Who's Afraid of the Crowd? *Critical Inquiry* 36 (4): 697-727

McAllister N. 2011. 'The futility of developer productivity metrics'. *InfoWorld* (Nov. 17). Accessible online at <https://www.infoworld.com/d/application-development/the-futility-developer-productivity-metrics-179244>

McCormick C. 2003. *The big project that never ends: Role and task negotiation within an emerging occupational community*. PhD Dissertation, University of Albany

McKusick M.K. 1999. 'Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable', in C. Dibona & S. Ockman (Eds.) *Open Sources: Voices*

from the Open Source Revolution. O'Reilly, pp. 31-46.

McKusick M.K., Bostic K., Karels M.J., Quarterman J.S. 1996. *The Design and Implementation of the 4.4 BSD Operating System*. Addison-Wesley Longman

McLuhan M. 1964. *Understanding Media: The Extensions of Man*. McGraw-Hill

Merlo F., Slaughter S., Francalanci C. 2009. *The co-evolution of organizational structures and software structures in open source and closed source projects*. Accessible online at <http://www.hbs.edu/units/tom/seminars/2007/docs/slaughter-paper.pdf>

Michels R. 1915. *Political Parties: A Sociological Study of the Oligarchical Tendencies of Modern Democracy*. Hearst's International Library

Michlmayr M., Robles G., Gonzalez-Barahona J.M. 2007. 'Volunteers in Large Libre software Projects: A Quantitative Analysis Over Time', in S.K. Sowe, I.G. Stamelos & I. Samoladas (Eds.) *Emerging Free and Open Source Software Practices.* Idea Group Publishing, pp. 1-24

Mikkola J.H. 2006. Capturing the Degree of Modularity Embedded in Product Architectures. *The Journal of Product Innovation Management* 23 (2): 128-146

Milev R., Muegge S., Weiss M. 2009. 'Design Evolution of an Open Source Project Using an Improved Modularity Metric', in C. Boldyreff, K. Crowston, B. Lundell & A. Wasserman (Eds.) *Open Source Ecosystems: Diverse Communities Interacting*. Springer, pp.20-33

Mill J.-S. 1965. *Principles of Political Economy*. University of Toronto Press

Miller E.M. 1978. The Extent of Economies of Scale: The Effects of Firm Size on Labor Productivity and Wage Rates. *Southern Economic Journal* 44 (3): 470-487

Mintzberg H. 1993. *Structure in Fives: Designing Effective Organizations.* Prentice-Hall

Mintzberg H., McHugh A. 1985. Strategy Formation in an Adhocracy. *Administrative Science Quarterly* 30 (2): 160-197

Mockus A., Fielding R.T., Herbsleb J.D. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11 (3): 309-346

Moede W. 1927. Die Richtlinien der Leistungs-Psychologie. *Industrielle Psychotechnik* 4: 193-207

Moody G. 2001. *Rebel Code: the Inside Story of Linux and the Open Source Revolution*. Perseus

Moon J.Y., Sproull L. 2000. Essence of Distributed Work: The Case of the Linux Kernel. *First Monday* 5 (11). Accessible online at <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/801/710>

Mumford L. 1963. *Technics and Civilization*. Harcourt Brace

Murmann J.P., Frenken K. 2006. Toward a systematic framework for research on dominant designs, technological innovations, and industrial change. *Research Policy* 35 (7): 925–952

Narduzzo A., Rossi A. 2005. 'The Role of Modularity in Free/Open Source Software Development', in S. Koch (Ed.) *Free/Open Source Software Development.* Idea Group

Oliver P.E., Marwell G. 1988. The Paradox of Group Size in Collective Action: A Theory of the Critical Mass. *American Sociological Review* 53 (1): 1-8

Olsen M. 2002. *The Logic of Collective Action: Public Goods and the Theory of Groups.* Harvard University Press

O'Mahony S., Ferraro F. 2007. The emergence of governance in an open source community. *Academy of Management Journal* 50 (5): 1079-1106

O'Neil M. 2009. *Cyberchiefs: Autonomy and Authority in Online Tribes*. Pluto Press

O'Reilly T. 2001. 'Remaking the Peer-to-Peer Meme', in A. Oram (Ed.) *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly

Osterloh M., Rota S. 2007. Open source software development-Just another case of collective invention?. *Research Policy* 36 (2): 157–171

Paine T. 1791. *Rights of Man: Answer to Mr. Burke's Attack on the French Revolution*. J.S. Jordan

Parnas D. 1972. On the Criteria to Be Used in Decomposing Systems into Modules. *Communications of the ACM* 15 (12): 1053 - 1058

Parsons T. 1939. The Professions and Social Structure. *Social Forces* 17 (4): 457-467

Perrow C. 1976. 'Control in organizations: the centralized-decentralized bureaucracy'. *Paper presented at the annual meeting of American Sociological Association*, N.Y.

Pfaffenberger B. 1996. "If i want it, it's OK": Usenet and the (Outer) Limits of Free Speech. *The Information Society* 12 (4): 365-386

Piazza T. 2010. 'Fundamentals of Applied Sampling', in P. Marsden & J. Wright (Eds.) *Handbook of survey research*. Emerald

Raymond E.S. 2004. 'Hacker ethic', in *The Jargon File* (v. 4.4.7). Accessible online at <http://www.catb.org/jargon/html/H/hacker-ethic.html>

Raymond E.S. 2003. *The Art of Unix Programming*. Addison-Wesley

Raymond E.S. 2000. *A Brief History of Hackerdom*. Accessible online at <http://www.catb.org/~esr/writings/homesteading/hacker-history/>

Raymond E.S. 1999. *The Cathedral and the Bazaar: Musings on Open Source and Linux by an Accidental Revolutionary.* O' Reilly

Raymond E.S. 1998. Homesteading the Noosphere. *First Monday* 3 (10). Accessible online at <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/621/542>

Reinstaller A. 2007. The division of labor in the firm: Agency, near-decomposability and the Babbage principle. *Journal of Institutional Economics* 3 (3): 293-322

Reinstaller A. 2006. *The division of labor, organizational control and the labor process: Near-decomposability and the Babbage principle*. European Network on the Economics of the Firm. Accessible online at <www.enef.group.shef.ac.uk/papers/Reinstaller.pdf>

Ritchie D.M. 1984. The Evolution of the Unix Time-Sharing System. *AT&T Bell Laboratories Technical Journal* 63 (6): 1577-93. Accessible online at <http://cm.bell-labs.com/cm/cs/who/dmr/hist.html>

Robinson A. 1934. The Problem of Management and the Size of Firms. *The Economic Journal* 444 (174): 242-257

Rosenberg N. 1994. 'Charles Babbage: pioneer economist', in N. Rosenberg, *Exploring the black box–Technology, economics, and history*. Cambridge University Press, pp. 24-46.

Rosenberg N. 1992. Economic experiments. *Industrial and Corporate Change* 1 (1): 181-203

Rosenbloom R.S., Cusumano M.A. 1987. Technological pioneering and competitive advantage: the birth of the VCR industry. *California Management Review* 29 (4): 51-76

Rothschild-Whitt J. 1979. The Collectivist Organization: An Alternative to

Rational-Bureaucratic Models. *American Sociological Review* 44 (4): 509-527

Rusovan S., Lawford M., Parnas D.L. 2005. 'Open Source Software Development: Future or Fad?', in J. Feller, B. Fitzgerald, S.A. Hissam & K.R. Lakhani (Eds.) *Perspectives on Free and Open Source Software.* MIT Press, pp. 107-123

Saers N. 2005. *A project model for the FreeBSD Project.* Accessible online at <hhttp://www.freebsd.org/doc/en/books/dev-model/book.html>

Salus P.H. 1994. UNIX at 25. *Byte* 19: 75-6. Accessible online at <http://www.wolldingwacht.de/unix/unix-at-25.html>

Sanchez R., Mahoney J. 1996. Modularity, Flexibility, and Knowledge Management in Product and Organization Design. *Strategic Management Journal* 17: 63-76

Scacchi W. 1995. 'Understanding Software Productivity', in D. Hurley (Ed.) *Advances in Software Engineering and Knowledge Engineering* 4, pp. 37-70

Schach S., Jin B., Wright D., Heller G., Offutt J. 2002. Maintainability of the Linux Kernel. *IEE Proceedings Software* 149 (1): 18-23

Scherer F.M. 1979. The Causes and Consequences of Rising Industrial Concentration. *Journal of Law and Economics* 22 (1): 191-208

Scherer F.M. 1970. *Industrial Market Structure and Economic Performance*. Rand McNally

Schweik C.M., English R.C., Kitsing M., Haire S. 2008. Brooks' Versus Linus' Law: An Empirical Test of Open Source Projects. *ACM International Conference Proceeding Series, vol. 289, Proceedings of the 2008 International Conference On Digital Government Research*, Montreal, Canada

Selby R., Basili V. 1988. *Analyzing Error-Prone System Coupling and Cohesion.* University of Maryland Computer Science Technical Report UMIACS-TR-88-46, CS-TR-2052

Selznick P. 1949. *TVA and the Grass Roots: A Study in the Sociology of Formal Organization*. University of California Press

Sewell G. 1998. The Discipline of Teams: The Control of Team-Based Industrial Work through Electronic and Peer Surveillance. *Administrative Science Quarterly* 43 (2): 397-428

Shah B.A. 2005. *Major Issues in the Development of a Hacker's Code of Ethics*. BSc Dissertation, MARA University of Technology. Accessible online at <http://eprints.ptar.uitm.edu.my/650/1/AHMAD_HAFIDZ_BIN_BAHAROM_ALAM_SHAH_05_24_.pdf>

Shankland S. 2005. 'Torvalds unveils new Linux control system'. *C Net* (Apr. 20). Accessible online at <http://news.cnet.com/Torvalds-unveils-new-Linux-control-system/2100-7344_3-5678651.html>

Sharman D., Yassine A. 2004. Characterizing Complex Product Architectures. *Systems Engineering Journal* 7 (1): 35-60

Siggelkow N. 2007. Persuasion with case studies. *Academy of Management Journal* 50 (1): 20-24

Siggelkow N., Levinthal D.A. 2003. Temporarily Divide to Conquer: Centralized, Decentralized, and Reintegrated Organizational Approaches to Exploration and Adaptation. *Organization Science* 14 (6): 650-669

Simon H.A. 2002. Near decomposability and the speed of evolution. *Industrial and Corporate Change* 11 (3): 587-599.

Simon H.A. 1991. Organizations and Markets. *The Journal of Economic Perspectives* 5 (2): 25-44

Simon H.A. 1973. Applying information technology to organization design. *Public Administration Review* 33 (3): 268-278

Simon H.A. 1962. The Architecture of Complexity. *Proceedings of the American*

Philosophical Society 106 (6)

Simon H.A. 1957. *Administrative Behavior* (2nd ed.) Macmillan

Slashdot. 2003. 'FreeBSD Core Developer Thrown Out' (Feb. 3). Accessible online at <http://bsd.slashdot.org/story/03/02/03/239238/FreeBSD-Core-Developer-Thrown-Out>

Sosa M., Eppinger S., Rowles C. 2007. A Network Approach to Define Modularity of Components in Complex Products. *Transactions of the ASME* 129 (11): 1118-1129

Sosa M., Eppinger S., Rowles C. 2004. The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. *Management Science* 50 (12): 1674-1689

Sosa M., Eppinger S., Rowles C. 2003. Identifying Modular and Integrative Systems and their Impact on Design Team Interactions. *Transactions of the ASME* 125 (2): 240-252

Spinellis D. 2006. 'Global software development in the FreeBSD project', in P. Kruchten, Y. Hsieh, E. MacGregor, D. Moitra & W. Strigel (Eds.) *International Workshop on Global Software Development for the Practitioner.* ACM Press, pp. 73–79

Staudenmayer N., Tripsas M., Tucci C.L. 2005. Interfirm Modularity and Its Implications for Product Development. *The Journal of Product Innovation Management* 22 (4): 303-321

Stern N. 1981. *From ENIAC to UNIVAC: An Appraisal of the Eckert-Mauchly Computers.* Digital Press

Stevens W., Myers G., Constantine L. 1974. Structured Design. *IBM Systems Journal* 13 (2): 115-39

Stewart D.V. 1981. The Design Structure Matrix: A Method for Managing the

Design of Complex Systems. *IEEE Transactions on Engineering Management* 28 (3): 71-74

Stokely M. 2011. *FreeBSD Release Engineering* (v. 1.87). Accessible online at <http://www.freebsd.org/doc/en/articles/releng/>

Stokely M. 2002. *FreeBSD Release Engineering* (v. 1.26). Accessible online at <http://docs.freebsd.org/doc/4.6-RELEASE/usr/share/doc/en/articles/releng/>

Sullivan W.G., Griswold Y. Cai, Hallen B. 2001. The Structure and Value of Modularity in Software Design. *SIGSOFT Software Engineering Notes* 26 (5): 99-108

Thurman W.N., Fisher M.E. 1988. Chickens, Eggs, and Causality, or Which Came First? *American Journal of Agricultural Economics* 70 (2): 237-238

Torres-Reyna O. 2008. *Panel Data Analysis: Fixed & Random Effects (using Stata 10.x)*. Accessible online at <http://dss.princeton.edu/training/Panel101.pdf>

Torvalds L. 2001. 'What Makes Hackers Tick? a.k.a. Linus's Law', in P. Himanen, *The Hacker Ethic and the Spirit of the Information Age*. Vintage, pp. xiii-xvii

Torvalds L. 1999. 'The Linux Edge', in C. Dibona, S. Ockman & M. Stone (Eds.) *Voices from the Open Source Revolution.* O'Reilly

Torvalds L. 1998. FM Interview with Linus Torvalds: What motivates free software developers. *First Monday* 3 (3). Accessible online at <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/583/504>

Torvalds L. 1991. '*Free minix-like kernel sources for 386-AT'.* Message posted to comp.os.minix (Oct. 5). Accessible online at <http://groups.google.com/group/comp.os.minix/msg/2194d253268b0a1b>

Trotter W. 1916. *Instincts of the Herd in Peace and War*. T. Fisher Unwin Ltd

Tuomi I. 2004. Evolution of the Linux Credits file: Methodological challenges and reference data for Open Source research. *First Monday* 9 (6), accessible online at <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1151/1071>

Turner F. 2006. *From Counterculture to Cyberculture: Stewart Brand, the Whole Earth Network, and the Rise of Digital Utopianism*. Chicago University Press

Ulrich K. 1995. The role of product architecture in the manufacturing firm. *Research Policy* 24 (3): 419-440

U.S.A. Air Force Dept. (Software Technology Support Center). 2000. *Guidelines for Successful Acquisition and Management of Software-Intensive Systems*. Accessible online at <http://www.stsc.hill.af.mil/resources/tech_docs/gsam3/chap13.pdf>

Walker A. 1866. *Science of Wealth: A Manual of Political Economy* (2nd ed.) John Wilson & Son

Walton C.E., Felix C.P. 1977. A Method of Programming Measurement and Estimation. *IBM Systems Journal* 16 (1): 54-65

Warfield J.N. 1973. Binary Matrices in System Modeling. *IEEE Transactions on Systems, Management, and Cybernetics* 3 (5): 441-449

Watson R. 2006. How the FreeBSD Project Works. *Proceedings of EuroBSDCon*, Milan, Italy. Accessible online at <http://www.watson.org/~robert/freebsd/2006eurobsdcon/eurobsdcon2006-howfreebsdworks.pdf>

Weber M. 2005. *The Protestant Ethic and the Spirit of Capitalism*. Routledge

Weber M. 1994. 'Socialism', in P. Lassman & R. Speirs (Eds.) *Weber: Political Writings*. Cambridge University Press, pp. 272-303

Weber M. 1978. *Economy and Society: An Outline of Interpretive Sociology*.

California University Press

Weber M. 1947. *The Theory of Social and Economic Organization*. The Free Press

Weber S. 2004. *The Success of Open Source.* Harvard University Press

Weick K.E. 1976. Educational Organizations as Loosely Coupled Systems. *Administrative Science Quarterly* 21 (1): 1-19

Wemm P. 2008. *Notes on Subversion*. Accessible online at <http://people.freebsd.org/~peter/svn_notes.txt>

van Wendel de Joode R. 2005. *Understanding open source communities: An organizational perspective*. PhD Dissertation, Delft University of Technology

Williams K., Harkins S., Latané B. 1981. Identifiability as a deterrent to social loafing: Two cheering experiments. *Journal of Personality and Social Psychology* 40 (2): 303-311

Williamson O.E. 1985. *The Economic Institutions of Capitalism: Firms, Markets, Relational Contracting*. The Free Press

Williamson O.E. 1975. *Markets and Hierarchies: Analysis and Antitrust Implications.* The Free Press

Williamson O.E. 1973. Markets and Hierarchies: Some Preliminary Considerations. *The American Economic Review* 63 (2): 323-324

Williamson O.E. 1967. Hierarchical Control and Optimum Firm Size. *The Journal of Political Economy* 75 (2): 123-138

Wheeler D.A. 2007. *Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!* Accessible online at <http://www.dwheeler.com/oss_fs_why.html>

Worren N., Moore K., Cardona P. 2002. Modularity, Strategic Flexibility, and Firm

Performance: A study of the Home Appliance Industry. *Strategic Management Journal* 23 (12): 1123-1140

Yin R. 1984. *Case study research.* Sage Publications

Yu L., Chen K., Ramaswamy S. 2009. Multiple-parameter coupling metrics for layered component-based software. *Software Quality Journal* 17 (1): 5-24

Yu L., Ramaswamy S. 2009. An empirical approach to evaluating dependency locality in hierarchically structured software systems. *The Journal of Systems and Software* 82 (3): 463-472

Yu L., Schach S.R., Chen K., Heller G., Offutt J. 2006. Maintainability of the kernels of open-source operating systems: A comparison of Linux with FreeBSD, NetBSD, and OpenBSD. *Journal of Systems and Software* 79 (6): 807-815

CURRICULUM VITAE

George Dafermos was born in 1980 in Heraklion-Crete, Greece. In 1997 he received his Apolytirion from the 3rd General Lyceum of Heraklion in which he majored in Creative Writing, Economics, History, Mathematics and Sociology. Subsequently, he studied at the University of Hertfordshire, Durham and Sunderland in the UK, whence he graduated with a BA Hons in Business Administration, a MA in Management and a MSc in Electronic Commerce Applications, respectively. In 2006 he embarked on a PhD research project at the Faculty of Technology, Policy and Management at Delft University of Technology, whose results are reported in the present dissertation. Over the past ten years, George has given more than fifty talks for audiences as diverse as the Chaos Communication Congress in Berlin, the Berkman Center for Internet and Society at Harvard and the Open Source Institute in Tokyo, and his work has been published in various scientific periodicals such as the Journal of Peer Production, Capital & Class, the Proceedings of the Chaos Communication Congress and First Monday. More recently, George's research interests are centred on practices of user innovation and on the articulation of authority in post-fordist organisations and online communities such as free and open source software projects. George can be contacted via email at georgedafermos@gmail.com.

NGInfra PhD Thesis Series on Infrastructures

1. Strategic behavior and regulatory styles in the Netherlands energy industry
   Martijn Kuit, 2002, Delft University of Technology, the Netherlands.
2. Securing the public interest in electricity generation markets, The myths of the invisible hand and the copper plate
   Laurens de Vries, 2004, Delft University of Technology, the Netherlands.
3. Quality of service routing in the internet: theory, complexity and algorithms
   Fernando Kuipers, 2004, Delft University of Technology, the Netherlands.
4. The role of power exchanges for the creation of a single European electricity market: market design and market regulation
   François Boisseleau, 2004, Delft University of Technology, the Netherlands, and University of Paris IX Dauphine, France.
5. The ecology of metals
   Ewoud Verhoef, 2004, Delft University of Technology, the Netherlands.
6. MEDUSA, Survivable information security in critical infrastructures
   Semir Daskapan, 2005,Delft University of Technology, the Netherlands.
7. Transport infrastructure slot allocation
   Kaspar Koolstra, 2005, Delft University of Technology, the Netherlands.
8. Understanding open source communities: an organizational perspective
   Ruben van Wendel de Joode, 2005, Delft University of Technology, the Netherlands.
9. Regulating beyond price, integrated price-quality regulation for electricity distribution networks
   Viren Ajodhia, 2006, Delft University of Technology, the Netherlands.
10. Networked Reliability, Institutional fragmentation and the reliability of service provision in critical infrastructures
    Mark de Bruijne, 2006, Delft University of Technology, the Netherlands.
11. Regional regulation as a new form of telecom sector governance: the interactions with technological socio-economic systems and market performance
    Andrew Barendse, 2006, Delft University of Technology, the Netherlands.
12. The Internet bubble - the impact on the development path of the

telecommunications sector
Wolter Lemstra, 2006, Delft University of Technology, the Netherlands.

13. Multi-agent model predictive control with applications to power networks
Rudy Negenborn, 2007, Delft University of Technology, the Netherlands.
14. Dynamic bi-level optimal toll design approach for dynamic traffic networks
Dusica Joksimovic, 2007, Delft University of Technology, the Netherlands.
15. Intertwining uncertainty analysis and decision-making about drinking water infrastructure
Machtelt Meijer, 2007, Delft University of Technology, the Netherlands.
16. The new EU approach to sector regulation in the network infrastructure industries
Richard Cawley, 2007, Delft University of Technology, the Netherlands.
17. A functional legal design for reliable electricity supply, How technology affects law
Hamilcar Knops, 2008, Delft University of Technology, the Netherlands and Leiden University, the Netherlands.
18. Improving real-rime train dispatching: models, algorithms and applications
Andrea D'Ariano, 2008, Delft University of Technology, the Netherlands.
19. Exploratory modeling and analysis: A promising method to deal with deep uncertainty
Datu Buyung Agusdinata, 2008, Delft University of Technology, the Netherlands.
20. Characterization of complex networks: application to robustness analysis
Almerima Jamaković, 2008, Delft University of Technology, Delft, the Netherlands.
21. Shedding light on the black hole, The roll-out of broadband access networks by private operators
Marieke Fijnvandraat, 2008, Delft University of Technology, Delft, the Netherlands.
22. On stackelberg and inverse stackelberg games & their applications in the optimal toll design problem, the energy markets liberalization problem, and in the theory of incentives
Kateřina Staňková, 2009, Delft University of Technology, Delft, the Netherlands.
23. On the conceptual design of large-scale process & energy infrastructure systems: integrating flexibility, reliability, availability, maintainability and

economics (FRAME) performance metrics
Austine Ajah, 2009, Delft University of Technology, Delft, the Netherlands.

24. Comprehensive models for security analysis of critical infrastructure as complex systems
Fei Xue, 2009, Politecnico di Torino, Torino, Italy.
25. Towards a single European electricity market, A structured approach for regulatory mode decision-making
Hanneke de Jong, 2009, Delft University of Technology, the Netherlands.
26. Co-evolutionary process for modeling large scale socio-technical systems evolution
Igor Nikolić, 2009, Delft University of Technology, the Netherlands.
27. Regulation in splendid isolation: A framework to promote effective and efficient performance of the electricity industry in small isolated monopoly systems
Steven Martina, 2009, Delft University of Technology, the Netherlands.
28. Reliability-based dynamic network design with stochastic networks
Hao Li, 2009, Delft University of Technology, the Netherlands.
29. Competing public values
Bauke Steenhuisen, 2009, Delft University of Technology, the Netherlands.
30. Innovative contracting practices in the road sector: cross-national lessons in dealing with opportunistic behaviour
Mónica Altamirano, 2009, Delft University of Technology, the Netherlands.
31. Reliability in urban public transport network assessment and design
Shahram Tahmasseby, 2009, Delft University of Technology, the Netherlands.
32. Capturing socio-technical systems with agent-based modelling
Koen van Dam, 2009, Delft University of Technology, the Netherlands.
33. Road incidents and network dynamics, Effects on driving behaviour and traffic congestion
Victor Knoop, 2009, Delft University of Technology, the Netherlands.
34. Governing mobile service innovation in co-evolving value networks
Mark de Reuver, 2009, Delft University of Technology, the Netherlands.
35. Modelling risk control measures in railways
Jaap van den Top, 2009, Delft University of Technology, the Netherlands.
36. Smart heat and power: Utilizing the flexibility of micro cogeneration

Michiel Houwing, 2010, Delft University of Technology, the Netherlands.

37. Architecture-driven integration of modeling languages for the design of software-intensive systems
    Michel dos Santos Soares, 2010, Delft University of Technology, the Netherlands.
38. Modernization of electricity networks: Exploring the interrelations between institutions and technology
    Martijn Jonker, 2010, Delft University of Technology, the Netherlands.
39. Experiencing complexity: A gaming approach for understanding infrastructure
    Geertje Bekebrede, 2010, Delft University of Technology, the Netherlands.
40. Epidemics in Networks: Modeling, Optimization and Security Games
    Jasmina Omi, 2010, Delft University of Technology, the Netherlands.
41. Designing Robust Road Networks: A general method applied to the Netherlands
    Maaike Snelder, 2010, Delft University of Technology, the Netherlands.
42. Simulating Energy Transitions
    Emile Chappin, 2011, Delft University of Technology, the Netherlands.
43. De ingeslagen weg. Een dynamisch onderzoek naar de dynamiek van de uitbesteding van onderhoud in de civiele infrastructuur
    Rob Schoenmaker, 2011, Delft University of Technology, the Netherlands.
44. Safety Management and Risk Modelling in Aviation: the challenge of quantifying management influences
    Pei-Hui Lin, 2011, Delft University of Technology, the Netherlands.
45. Transportation modelling for large-scale evacuations
    Adam J. Pel, 2011, Delft University of Technology, the Netherlands.
46. Clearing the road for ISA Implementation?: Applying Adaptive Policymaking for the Implementation of Intelligent Speed Adaptation
    Jan-Willem van der Pas, 2011, Delft University of Technology, the Netherlands.
47. Design and decision-making for multinational electricity balancing markets
    Reinier van der Veen, 2012, Delft University of Technology, The Netherlands.
48. Understanding socio-technical change. A system-network-agent approach
    Catherine Chiong Meza, 2012, Delft University of Technology, the Netherlands.

49. National design and multi-national integration of balancing markets
    Alireza Abbasy, 2012, Delft University of Technology, the Netherlands.
50. Regulation of Gas Infrastructure Expansion
    Jeroen de Joode, 2012, Delft University of Technology, The Netherlands.
51. Governance Structures of Free/Open Source Software Development: Examining the role of modular product design as a governance mechanism in the FreeBSD Project
    George Dafermos, 2012, Delft University of Technology, the Netherlands.

Order information: info@nextgenerationinfrastructures.eu

GOVERNANCE STRUCTURES OF FREE/OPEN SOURCE SOFTWARE DEVELOPMENT

Modularity theory makes a compelling argument: modular product design increases the potential number of persons that could work on a distributed project and has a positive effect on their labour productivity because it allows them to work independently of each other, with little or no need for central coordination. This doctoral dissertation sets out to put this argument to the test by studying a phenomenon that combines both scale and modularity: Free and open source software (FOSS) development. Its central question is: Does modularity mitigate the adverse effects of increasing scale in FOSS development?

In exploring the effect of modularity and increasing scale on the dynamic of development of FreeBSD, a large and well-known FOSS project, over a period of fifteen years, the dissertation addresses several related empirical issues: How are FOSS projects organised? How are they governed? And most interestingly, how do they manage increasing scale? Does their ability to self-organise diminish as they grow larger, thereby necessitating hierarchical coordination?

The Next Generation Infrastructures Foundation

represents an international consortium of knowledge institutions, market players and governmental bodies, which joined forces to cope with the challenges faced by today's and tomorrow's infrastructure systems. The consortium cuts across infrastructure sectors, across discplinary borders and across national borders, as infrastructure systems themselves do. With the strong participation of practitioners in a concerted knowledge effort with social and engineering scientists, the Foundation seeks to ensure the conditions for utilization of the research results by infrastructure policy makers, regulators and the infrastructure industries.

www.nginfra.nl

NEXT GENERATION INFRASTRUCTURES FOUNDATION

TU Delft Delft University of Technology

9 789079 787401