

Open Source Community Building

as

Licenciate

at the

Faculty of Economics and Social Science

of the University of Bern

submitted to

Prof. Dr. Thomas Myrach

Institute of Information Systems

by

Matthias Stürmer

from Pfäffikon ZH

in the 9th Semester

Matriculation Number: 00-114-256

Address:

Wabernstrasse 51

CH-3007 Bern

(phone: +41 31 371 80 87)

(e-mail: matthias@stuermer.ch)

Bern, 2005-03-02

Abstract

Building an active and helpful community around an open source project is a complex task for its leaders. Therefore investigations in this work are intended to define the optimum starting position of an open source project and to identify recommendable promoting actions by project leaders to enlarge community size in a healthy way. For this paper eight interviews with committed representatives of successful open source projects have led to over 12 hours of conversation about community building. Analysing the statements of these experienced community members exposed helpful activities that led to the presently prospering communities of their projects. Summarizing the conclusions of this qualitative research a table with conditions for successful open source project initialisation and a subject-level promotion matrix of community building could be created. They include suggestions on how to start a new open source project and how to improve and increase the community of an already advanced open source project.

Zusammenfassung

Eine aktive und hilfreiche Community um ein Open Source Projekt zu bilden ist eine anspruchsvolle Aufgabe für dessen Leiter. Deshalb ist es das Ziel dieser Arbeit herauszufinden, welches die optimalen Ausgangsbedingungen eines Open Source Projektes sind und welche Aktivitäten der Projektleiter erfahrungsgemäss zu einem gesunden Wachstum der Community führen. Für diese Forschungsarbeit wurden acht Interviews mit engagierten Vertretern von erfolgreichen Open Source Projekten durchgeführt, die insgesamt zu über 12 Stunden Gespräch über den Aufbau einer Community führten. Die Auswertung der Aussagen dieser erfahrenen Community-Mitglieder zeigten, welches die erfolgreichen Eingriffe sind, die zu der aktuellen, wirkungsvollen Community der Projekte geführt haben. Als Resultat dieser qualitativen Forschungsarbeit ergab sich einerseits eine Tabelle mit den Voraussetzungen für die erfolgreiche Initialisierung eines Open Source Projekts und andererseits eine Themen-Ebenen Matrix, die die Förderung des Aufbaus einer Community beschreibt. Die Darstellungen beinhalten Empfehlungen zum Start eines neuen Open Source Projekts und zur Verbesserung und Vergrösserung der Community in einem bereits fortgeschrittenen Projekt.

Résumé

La création d'une communauté active et bénéfique pour un projet open source représente pour leurs responsables un défi de taille. Les objectifs de ce travail de recherche sont d'une part la définition des conditions initiales optimales d'un projet open source et d'autre part l'élaboration des activités des responsables qui conduiront de manière empirique à un bon développement de la communauté. Ce document rassemble les interviews de huit personnes actives dans différents projets open source menés avec succès. En tout, ils représentent plus de 12 heures de conversation sur le fonctionnement d'une communauté. L'évaluation des avis de ces personnes a montré quelles sont les actions qui conduisent à la constitution de communautés actuellement efficaces. Pour conclure cette recherche qualitative, on est parvenu à créer un ensemble de conditions permettant d'initialiser avec succès des projets open source ainsi qu'une matrice au niveau du thème pour la promotion de la constitution d'une communauté. Ces représentations comportent des recommandations permettant le lancement d'un nouveau projet open source ainsi que l'amélioration et l'agrandissement de projets open source déjà existants.

Table of Contents

ABSTRACT.....	2
ACKNOWLEDGEMENTS.....	8
1. INTRODUCTION.....	9
1.1. Research Issue.....	9
1.2. Goals.....	9
1.3. Methodology.....	10
1.4. Realization.....	11
1.4.1. Getting Started.....	11
1.4.2. Selecting Cases.....	11
1.4.3. Crafting Instruments and Protocols.....	11
1.4.4. Entering the Field.....	12
1.4.5. Analysing Data.....	12
1.4.6. Shaping Hypotheses.....	12
1.4.7. Enfolding Literature.....	12
1.4.8. Reaching Closure.....	13
1.5. Investigated Open Source Projects.....	13
2. THE OPEN SOURCE PROJECT.....	14
2.1. Software.....	15
2.1.1. Definition.....	15
2.1.2. Life Cycle.....	15
2.2. Contributors.....	17
2.2.1. Definition.....	17
2.2.2. Roles.....	18
2.2.2.1. Contributor.....	18
2.2.2.2. Developer.....	18
2.2.2.3. Leader.....	19
2.2.2.4. Core Developer.....	19
2.2.2.5. Project Owner.....	19
2.2.2.6. Initiator.....	19
2.2.3. Motivation.....	20
2.3. Community.....	20
2.3.1. Definition.....	20

2.3.2. Collaboration Technologies.....	20
2.3.2.1. IRC.....	20
2.3.2.2. Mailing List.....	21
2.3.2.3. Wiki.....	21
2.3.2.4. Blog.....	21
2.3.2.5. Sprint.....	22
2.3.2.6. Revision Control Systems.....	22
2.3.2.7. API.....	23
2.3.3. Reasons for Community Building.....	23
2.3.4. Characteristics of Desired Communities.....	25
2.3.4.1. Productivity.....	25
2.3.4.2. Self-Motivation.....	26
2.3.4.3. Diversity.....	27
2.3.4.4. Correct Behaviour.....	28
2.3.4.5. Altruism.....	28
2.3.4.6. Perseverance.....	29
2.3.4.7. Common Vision.....	30
3. INITIALISATION OF OPEN SOURCE PROJECTS.....	31
3.1. Leadership Skills and Behaviours.....	31
3.1.1. Assertiveness.....	31
3.1.2. Commitment.....	32
3.1.3. Communicativeness.....	33
3.1.4. Experience.....	34
3.1.5. Helpfulness.....	35
3.1.6. Openness.....	36
3.1.6.1. Open to Join.....	36
3.1.6.2. Open to the Choice of Work.....	37
3.1.6.3. Open to Leave.....	37
3.1.6.4. Open to Communicate.....	37
3.1.6.5. Limits of Openness.....	38
3.1.7. Patience.....	38
3.1.8. Personality.....	39
3.1.9. Presence.....	39
3.1.10. Programming.....	40

3.1.11. Responsibility.....	41
3.1.12. Visionary.....	42
3.2. Prerequisites of the Project.....	43
3.2.1. Programming Language.....	44
3.2.2. Open Source License.....	46
3.2.3. Great Initial Source Code.....	49
3.2.4. Public Demand.....	50
3.2.5. Degree of Novelty.....	52
3.2.6. Applicability.....	53
3.2.7. Level of Communication.....	54
4. PROMOTION OF COMMUNITY BUILDING.....	56
4.1. Modularity.....	57
4.1.1. Recruiting.....	57
4.1.2. Collaboration.....	59
4.1.3. Production.....	61
4.2. Documentation.....	62
4.2.1. Recruiting.....	62
4.2.2. Collaboration.....	64
4.2.3. Production.....	66
4.3. Release Management.....	67
4.3.1. Recruiting.....	68
4.3.2. Collaboration.....	69
4.3.3. Production.....	70
4.4. Collaboration Platform.....	72
4.4.1. Recruiting.....	72
4.4.2. Collaboration.....	73
4.4.3. Production.....	74
4.5. Physical Meetings.....	75
4.5.1. Recruiting.....	75
4.5.2. Collaboration.....	77
4.5.3. Production.....	79
4.6. Foundation.....	80
4.6.1. Recruiting.....	80
4.6.2. Collaboration.....	82

4.6.3. Production.....	85
4.7. Internationalisation.....	86
4.7.1. Recruiting.....	86
4.7.2. Collaboration.....	87
4.7.3. Production.....	88
4.8. Recruiting Specific Subjects.....	89
4.8.1. Spreading the Word.....	89
4.8.2. Credit System.....	91
4.9. Collaboration Specific Subjects.....	92
4.9.1. Communication Channels.....	92
4.9.1.1. IRC.....	93
4.9.1.2. Mailing List.....	94
4.9.1.3. Blog.....	95
4.9.2. Community Structure.....	96
4.9.3. Task List.....	97
4.10. Production Specific Subjects.....	99
4.10.1. Code and Feature Quality.....	99
4.10.2. User Interface Design.....	100
4.10.3. Installation.....	101
5. CONCLUSIONS.....	103
5.1. Successful Initialisation.....	103
5.2. Subject-Level Promotion Matrix.....	105
5.3. Interpretations of Interview Responses.....	109
5.3.1. Demand Driven Actions.....	109
5.3.2. Dependence on the Progress of the Project.....	110
5.4. Do's and Don'ts for OSP Leaders.....	111
5.5. Future Research and Closing Comment	113
APPENDIX.....	114
LIST OF FIGURES.....	148
LIST OF TABLES.....	148
GLOSSARY.....	149
BIBLIOGRAPHY.....	150
DECLARATION OF DISCRETENESS.....	153

Acknowledgements

- Bernhard Bühlmann, Boris Kraft, Bertrand Delacrétaiz, Guido Wesdorp, Michael Wechner, Daniel Hinderink, Bård Farstad and Gregor Rothfuss for their time and openness talking about their fascinating open source projects
- Ekkehard Stürmer for his essential English support and interesting conversations about open source community building
- Prof. Thomas Myrach for his supportive supervision and motivation during the writing of this paper
- Christian Laux for advices about licensing issues of open source projects
- Stefan Häfliger for interesting conversations about open source research
- Andreas Voss for his comments on the paper's structure
- Michel Dufour for the French support and conversations about open source issues
- Florian Lüchinger for his helpful comments about citations and others
- Emanuel Indermühle for his inspiration building a community for the Vida project
- Anita Stürmer for the French support and her love and patience

1. Introduction

The introduction describes the motivation, the goals and the methodology of this paper and introduces the investigated open source projects and their representatives.

1.1. Research Issue

In recent time the influence of open source software in IT industry has become more and more important. Existing open source projects have grown, new projects have been founded and previously proprietary software has been released under open source licenses. To quantify the success of such open source projects it has been proposed, among others, to measure the size of the development communities, the activity inside the projects, the time for bug fixing and number of downloads.¹ Such measurements show that some of the major goals of all open source projects are to increase community size, improve internal collaboration and to further develop the software. To this end it is important to understand the inner functioning of such communities around open source projects and identify their central topics of actions. Also it is necessary to determine the optimum prerequisites for the successful start of new open source projects.

1.2. Goals

The goal of this paper is to identify successful ways of how to initiate and promote community building of open source projects. To this end, the actions of the project's key persons, the open source project leaders, will be analysed since they have the largest experience and most influence on the community. First, the optimum initial conditions for new open source projects are described. Then, for already running projects, common ways of leadership behaviour will be found that have raised the projects' attractiveness and thus motivated voluntary contributors to join the community. Also it will be investigated how collaboration among active community members can be improved and how the outcome, the software in use, is further developed. Thus it is the goal to outline best practices of how to successfully direct open source projects.

¹ See Crowston et.al. (2004), p. 1

1.3. Methodology

Since the promotion of community building has not been investigated much this paper is mainly based on case study research. First, the scientific approach is outlined describing the process of creating theory from case study research. Second, the procedure adopted in this writing is explained showing the application of case study methodology to this particular investigation.

Elaborately describing the proceedings in qualitative research, Eisenhardt shows a roadmap for building theories from case study research.² Table 1 lists the 8 steps of the proposed process:

#	Step	Activity
1	Getting Started	<ul style="list-style-type: none"> • Definition of research question • Possibly a priori constructs • Neither theory nor hypotheses
2	Selecting Cases	<ul style="list-style-type: none"> • Specified population • Theoretical, not random, sampling
3	Crafting Instruments and Protocols	<ul style="list-style-type: none"> • Multiple data collection methods • Qualitative and quantitative data combined • Multiple investigators
4	Entering the Field	<ul style="list-style-type: none"> • Overlap data collection and analysis including field notes • Flexible and opportunistic data collection methods
5	Analysing Data	<ul style="list-style-type: none"> • Within-case analysis • Cross-case pattern search using divergent techniques
6	Shaping Hypotheses	<ul style="list-style-type: none"> • Iterative tabulation of evidence for each construct • Replication, not sampling, logic across cases • Search evidence for “why” behind relationships
7	Enfolding Literature	<ul style="list-style-type: none"> • Comparison with conflicting literature • Comparison with similar literature
8	Reaching Closure	<ul style="list-style-type: none"> • Theoretical saturation when possible

Table 1: Process of Building Theory from Case Study Research³

This paper is intended to rely as closely as possible on the proposed process guiding the way through a rather unstructured qualitative data collection and analysis.

² See Eisenhardt (1989), p. 532

³ See Eisenhardt (1989), p. 533, Table 1

1.4. Realization

The realization of this paper's research followed the steps outlined above by Eisenhardt. The actions in all eight phases are described briefly in the following paragraphs.

1.4.1. *Getting Started*

Initially literature about open source communities was studied with special emphasis on the current state of the scientific research about community building in open source projects. The following research question helped to focus on the topic of interest: *What actions attract professional developers and helpful community members to participate in an open source project?* Literature and previous personal experiences with open source software gave a basic idea where major sources of data could be expected.

1.4.2. *Selecting Cases*

All leaders of open source projects published under an Open Source Initiative (OSI) approved software license were defined as case population. Following Eisenhardt no random selection of investigated units was made. For accessibility reasons a convenience sampling of open source project representatives known to the author was realized. According to Joppe, such a convenience sampling is appropriate for exploratory research where representativeness of the information doesn't have the highest priority.⁴ Out of ten contacted project representatives eight persons replied positively to an interview request.

1.4.3. *Crafting Instruments and Protocols*

Although scientifically interesting, for lack of capacity and time no multiple data collection methods were employed, no quantitative data about the projects could be collected and no multiple investigators could be assigned with interrogations.

4 See Joppe (2005)

1.4.4. *Entering the Field*

In November and December 2004 eight interviews with representatives of eight different open source projects were conducted. As shown in table 2 the conversations range from one to almost two hours of duration, all recorded as MP3 files. Following the advice of Eisenhardt, the interview guideline was continuously adapted taking advantage of emerging topics and unique features of the case.

1.4.5. *Analysing Data*

To ensure the complete availability of conversation data all interviews were transcribed and those conducted in German were translated into English.⁵ Thereafter the texts were semantically categorized into 842 codings using the qualitative data analysis software MAX.QDA2.⁶ This allows various possibilities of evaluation, especially cross-case pattern sampling where comparisons between different answers to the same question could be carried out.

1.4.6. *Shaping Hypotheses*

Already during the process of coding the interview fragments certain repeating patterns could be observed leading to a first version of the subject-level promotion matrix of community building.⁷ Iteratively continuing text analysis and framework design the final structure of the subject-level matrix was achieved. Also, two different points of time in an open source project, before and after official publication of the software, could be detected forming the major content of chapters 3 and 4.

1.4.7. *Enfolding Literature*

Existing research papers about open source community behaviour and structure allowed comparisons between literature and the case study results of this writing. Remarkably, a number of times exactly the same phenomena were discovered underlining the generalizability of the constructs.

5 It needs to be remarked that although the translations were done with great care not all nuances could be transferred to the English language. Also all of the interview partners speak English very well and thus probably would have formulated answers differently if the interviews had been conducted in English.

6 Unfortunately the tested open source software for qualitative data analysis didn't meet the requirements of this research at that time.

7 The final matrix can be found on page 108.

1.4.8. Reaching Closure

The final version of both tables, initialisation and promotion of community building, were drafted when all of the relevant interview statements could be assigned to one of the indicated fields of the matrix. Thus all available data was utilised forming the definite conclusions of the research.

1.5. Investigated Open Source Projects

All interviews were conducted with closely attached community members of the open source project. Thus, according to the definition in this writing, they can all be called open source project leaders. The role they occupy in the community gives them strong influence therefore enabling them to accomplish actions concerning promotion of community building. Either they have founded the project themselves, worked intensively in further development or are very active in promotion of the open source project. Table 2 shows the main variables about the investigated projects, their representatives and the interviews itself.









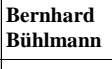

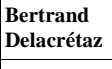
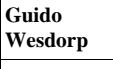
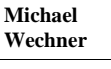
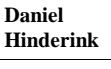

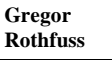
Open Source Project								
	 Plone	 Magnolia	 Cocoon	 Kupu	 Lenya	 TYPO3	 eZ publish	 Xaraya
Category	CMS Framework	CMS	Web Application Framework	WYSIWYG Browser Editor	CMS	CMS	CMS	CMS and Application Framework
Prog. Lang.	Python	Java	Java	JavaScript	Java	PHP	PHP	PHP
Initiator(s)	Alexander Limi & Alan Runyan	obinary	Stefano Mazzocchi	Guido Wesdorp and others	Michael Wechner	Kasper Skårhøj	eZ systems	?
Association	Plone Foundation	none	Apache Software Foundation	none	Apache Software Foundation	TYPO3 Association	none	Digital Development Foundation
Interviewee								
	 Bernhard Bühlmann	 Boris Kraft	 Bertrand Delacrétaiz	 Guido Wesdorp	 Michael Wechner	 Daniel Hinderink	 Bård Farstad	 Gregor Rothfuss
Role in the OSP	Community Member	Community Representative	Core Developer	Core Developer	Initiator and Core Developer	Marketing Leader	Community Representative	Core Developer
Company	4teamwork	obinary	codeconsult	Infrae	wyona	dpool	eZ systems	wyona
Interview								
Date	2004-11-17	2004-11-19	2004-11-24	2004-11-25	2004-11-30	2004-12-03	2004-12-03	2004-12-07
Duration	95 min	108 min	63 min	79 min	106 min	116 min	78 min	103 min
Language	German	German	German	English	German	German	English	German
Location	Bern, CH	Basel, CH	Cugy, CH	Rotterdam, NE (phone)	Bern, CH	München, DE (phone)	Skien, NO (phone)	Boston, USA (phone)

Table 2: Information about the Investigated Open Source Projects, Interviewees and Interviews

2. The Open Source Project

This chapter aims to give a short introduction into the world of open source projects (OSP). It is far from giving a complete picture but intends to clarify the most important terms and relations. First, the main characteristics of open source software are explained briefly including the definition and a typical life cycle of such software. Second, the persons and groups of people involved in the open source project are outlined describing the different roles they might occupy during the project's life cycle. Finally, the community as the aggregation of all participating persons is characterized pointing out common values, habits, rules and means of communication. Also reasons are given why community building is essential by recruiting new contributors and improving collaboration and production processes in the project. In figure 1, the OSP is modelled to provide a short overview of the involved research objects.

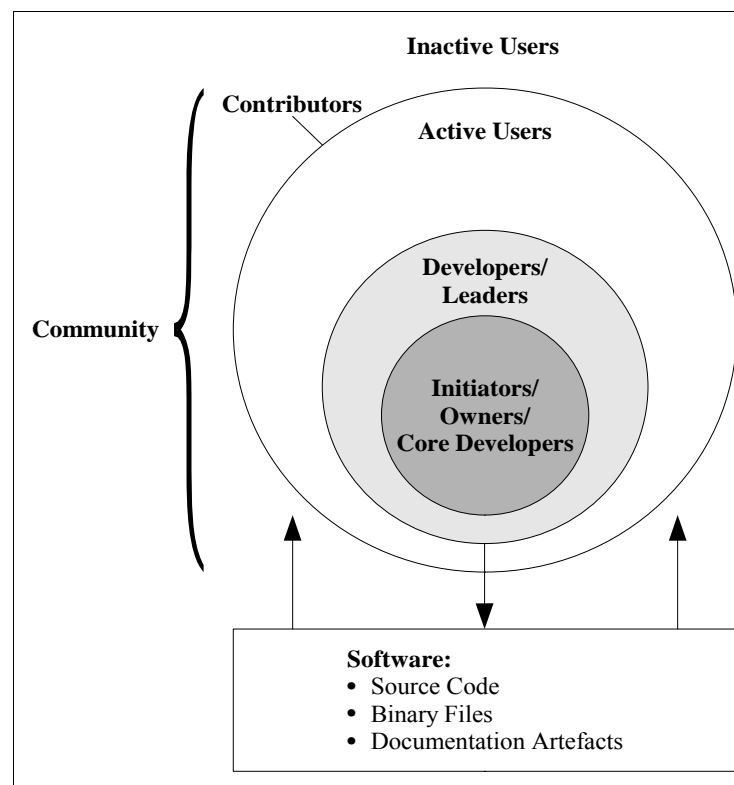


Figure 1: A Very Abstract Picture of an Open Source Project (OSP)

2.1. Software

The open source software consists of the complete source code, the executable binary files and, depending on literature, also all documentation artefacts.⁸ The software is the end product of all the community members' work.

2.1.1. Definition

As defined on the OSI (Open Source Initiative) website open source software has to be published under a license which fulfils all the 10 criteria of the open source definition.⁹ Especially important are the first two stating the license must not require any fee for the distribution of the software and the source code has to be included in a well-readable form together with the program.

2.1.2. Life Cycle

In his paper about a framework for OSPs Rothfuss describes the typical life cycle of a community founded OSP.¹⁰ Starting with the famous “*scratching a developer's personal itch*” phenomenon¹¹ the initiator of the project develops a first solution for his problem. Before widely publishing it he might ask interested persons for some feedback and initial support. In his study about profiling open source project ecologies Koch assumes the first commit of source code into the code repository as the start of the project.¹² Once the source code is publicly available the initiator announces it over various news channels and open source platforms attracting interested persons to his project. If everything fits they get involved in the OSP first using the software and later on contributing themselves to the continuous progress of the project. The community grows and evolves always adapting to the changing environment. Thus a clear difference of OSPs compared to classical software projects becomes obvious: As long as the ambience is healthy the open source software is continuously advanced by its community as opposed to classical projects where their life cycle ends when the development process has finished.

⁸ See FOLDOC (2005)

⁹ URL: <http://opensource.org/docs/definition.php> (2005-02-14)

¹⁰ See Rothfuss (2002), p. 38

¹¹ See Raymond (1999), p. 23

¹² See Koch (2004), p. 86

Applications actively used and embedded in a real world domain have to undergo constant adaptation even if they have reached a state of maturity. Otherwise they become progressively less satisfactory, as stated by the first law of software evolution of Lehman/Ramin.¹³ This law complies well with the progression of open source software, Bauer and Pizka conclude in their research about the evolution of free software.¹⁴ A famous example of a successful OSP is the Apache Webserver. This project is still advancing although it exists for almost ten years now. During the interview Rothfuss mentioned: *“Release 2.2 is just about to come out. There won’t be such radical changes as from version 1.3 to 2.0 but minor ones for sure. For example the filter API which is also used for PHP will be more flexible and other modules are rewritten. So you can say projects like the Apache Web Server are very stable because they implement a specification which by itself is very stable so there won’t be big surprises. Also this project is some sort of infrastructure software. In other types of software like book-keeping or desktop publishing it’s not that well defined what the spectrum of functionality should include. So there seems to happen more than in projects like the Apache Web Server.”* Although a content management system isn’t infrastructure software the TYPO3 project has advanced so far that quick improvements are not possible any more, as Hinderink knows: *“[To add changes] you have to get into it quite deeply because a lot is available already. I believe it’s much easier to get into a project where not much has happened so far. There are still things to do on a relatively low level. It’s different in TYPO3. Probably no matter what you want to do there are always at least attempts or beginnings of others. It’s really difficult to have a new idea. So if you have a new idea that’s great but if not, you need to at least look at the existing attempts and work with them. So participation needs more.”*

As the software grows in technical size measured e.g. in lines of code, also the community evolves continuously splitting into subsets and assigning new tasks to the community members according to their skill, experience and current project responsibilities.¹⁵ The healthy evolution of open source software is described by Rothfuss when he distinguishes the stages of planning, pre-alpha, alpha, beta, stable and ma-

13 See Lehman/Ramin (2001), p. 16

14 See Bauer/Pizka (2003), p. 173

15 See Bauer/Pizka (2003), p. 174

ture.¹⁶ On the other hand, conflicts in the community may lead to interruption in the software evolution.¹⁷ The most dramatic exit option is the fork, the independent further development of the software enforced by one or more developers.¹⁸

2.2. Contributors

The term “contributors” encompasses all the active members of the OSP emphasizing their individual aspects and roles. Thus a definition of a contributor is supplied, the different roles of the contributors are discussed and finally some aspects of their motivation for participation are mentioned.

2.2.1. Definition

Gathering data about project participation Koch only considered persons actively programming as part of the manpower of the community excluding people participating in discussions, bug reporting, documentation etc.¹⁹ Rothfuss on the other hand stands up against a two class community by assigning equal value to contributions of non-developers as to those of programmers: *“I believe there must be the same level of incentives for them [i.e. non-developers] as there is for developers. It’s necessary to have people who are motivated to work on the documentation. It’s important not to create a two class community and to communicate that this kind of contribution is as important as coding. So it’s important to value feedback and tests by the users. In this way, a self-dynamic cycle starts letting people realize that their contributions and skills are appreciated by the community. In a certain aspect it can be more complex to do these things. If for example the code doesn’t compile it’s quite clear what to do. But if the usability is bad, it’s not so clear what should be improved. Such issues are much less visible so people are needed who point out the problems.”*

So in this paper the definition of a contributor includes all persons actively participating in the progress of the OSP, software developers as well as non-developers.

¹⁶ See Rothfuss (2002), p. 38

¹⁷ See Van Wendel de Joode (2004). p. 110

¹⁸ See Wikipedia (2005)

¹⁹ See Koch (2004), p. 86

2.2.2. Roles

In the literature there is no common terminology concerning the various roles in the open source movement. Whereas e.g. the term “user” is defined as a non contributing person in Rothfuss’ framework for open source projects²⁰, Zhao and Deek on the other hand conducted their survey by assigning this term to all actively participating and contributing persons.²¹ In the examination of the Freenet project by von Krogh et. al. persons not participating in software development were named “non-developers.”²²

To simplify matters only those roles are defined which are mentioned frequently in this writing. These are the roles of the contributor, the developer, the leader, the core developer, the project owner and the initiator. Figure 1 shows the various relationships of these roles and their combination in an abstract way by drawing cascaded circles.

2.2.2.1. Contributor

All persons actively progressing the OSP are acting in the role of a contributor. On one side there are active users participating e.g. in the mailing list discussions, reporting bugs or donating money to the projects’ foundation. On the other side there are all the software developers and leaders of the OSP managing the progress of the project itself. Not included are the inactive users who have downloaded and installed the software and might even have subscribed to the mailing list but never write a message nor actively show any other sign of their existence.

2.2.2.2. Developer

A developer is everyone who actively works on the source code of the OSP. This person may or may not have commit access to the code repository neglecting the distinction made in the research about community joining by von Krogh et. al.²³ The role of the developer primarily refers to a person’s programming skills.

²⁰ See Rothfuss (2002), p. 109

²¹ See Zhao/Deek (2004), p. 91

²² See von Krogh/Spaet/Lakhani (2003), p.1220

²³ See von Krogh/Spaet/Lakhani (2003), p. 1220

2.2.2.3. Leader

As opposed to the software developer, the project leader is more concerned with the human beings in the OSP. He cares about communication and collaboration issues, manages for instance the software releases and organizes a supportive environment in general. In coordination with the developers he is responsible for the promotion of community building but focussing on management tasks rather than programming issues.

2.2.2.4. Core Developer

As confirmed in various investigations about the level of developer activity in OSPs there is usually a small group of programmers that is responsible for the largest part of software development.²⁴ These persons differ from the other developers in their long experience with the OSP and their knowledge about the history and architecture of the project giving them a large influence in the community.

2.2.2.5. Project Owner

When a certain “critical size,” measured in lines of code, number of contributors etc., is surpassed by an OSP it can’t be managed by a single person any more. A group of people forming an executive board will then coordinate and control further evolution of the OSP.²⁵ They are called the OSP owners possessing root access to the development server or the collaboration platform and thus being able to grant CVS commit access to new developers.

2.2.2.6. Initiator

The initiator is the first person in the project contributing from the beginning on. In the early stages of the life cycle of an OSP the initiator is the author of most of the source code. In advanced projects his initial contribution may almost vanish among the participation of the other developers. In the case of an OSP explicitly founded by a software development company, this company can be viewed as initiator uniting all initial developers.

²⁴ See Koch (2004), p. 82

²⁵ See Bauer/Pizka (2003), p. 174

2.2.3. *Motivation*

Various authors have investigated the motivation behind the participation in an OSP.²⁶ Hars and Ou name two different types of motivations, intrinsic and extrinsic, as basis for all contributions OSPs. The research of Lakhani and Hippel concludes the helping effort of the community results in direct learning benefits for the contributors. And Lakhani and Wolf found as major motivations fun, user need, intellectual challenge and programming skill improvement. All these kinds of motivations are also found in the interview responses of this paper but won't be further examined here.

2.3. Community

While contributors are part of the microcosmic view of the open source world, the community around an OSP is part of the macrocosmic perspective. Again a definition shall clarify the meaning of the term "community" in this paper.

2.3.1. *Definition*

As shown in figure 1, all the contributors of an OSP together form the community of this project. In contrast to the single contributors' view where e.g. the motivation of the individual developer was important, the community view is concerned with topics like shared values and common vision. Thus the contributors' perspective is a look from inside the OSP whereas the community perspective is an outside view of the OSP perceiving the members of the project as a closed unity.

2.3.2. *Collaboration Technologies*

To facilitate the reading of this paper the most commonly used collaboration techniques and terms in open source communities are briefly explained.²⁷

2.3.2.1. IRC

IRC (Internet Relay Chat) is an internet technology that allows group communication in chat rooms and also one-to-one communication for private talks. Since it was created as early as 1988 it enjoys a very high diffusion among software developers. To start to chat, one downloads a client application, then connects to an IRC server

²⁶ See Hars/Ou (2001), Lakhani/Hippel (2003), Lakhani/Wolf (2005)

²⁷ See Wikipedia (2005)

and finally chooses one or more of the available channels. Since chatting is a synchronous way of communication and there might be people who are interested in the conversations but can't follow them continuously there exists the possibility to log the discussions and publish them e.g. on a website.

2.3.2.2. Mailing List

A mailing list is a collection of email addresses of people who are interested in the same topic of discussion. There are different types of mailing lists, closed lists where the administrator has to add the recipients manually, and public lists where people can subscribe on their own by sending an email to a specific address. Usually there is also an online mailing list archive with search functionality where the conversations are grouped into the discussion threads. As email is a very popular communication channel most OSPs have one or more public mailing lists where the main discussions about the project take place.

2.3.2.3. Wiki

A Wiki is a website where everyone can view, add and modify its contents. The basic idea is that the creation of new pages with text, links and images should be as simple as possible to ensure a low entry barrier for all contributors. The system uses a very simple syntax to format text being written in the internet browser including the possibility to create new pages just by following certain writing rules. To prevent data loss, all changes in the Wiki are recorded and can be recovered at any time. There are many different Wiki implementations ranging from very simple solutions to sophisticated applications with lots of additional functionality like searching and media integration.

2.3.2.4. Blog

A Weblog or simply a Blog is a web application that manages the frequent posting of news messages. Usually, a Blog is used as the personal diary of an author categorizing his entries into different topics. Sometimes there exists also an OSP-specific Blog where various writers can post articles or an OSP hosts a so called Planet where all the messages of the contributors' Blogs are aggregated bearing the OSP's category. Most Blogs offer the possibility to add public comments so they act as an interactive

medium between the authors and the readers. With protocols like RSS (Really Simple Syndication) the news can be monitored with RSS clients and they can also be referenced among the different Blog websites.

2.3.2.5. Sprint²⁸

A Sprint is a meeting of open source software developers for two or three days. The goal is intense further development of the software in an XP (Extreme Programming) styled manner which includes pair programming among other things.²⁹ Developers travelling from all over the world meet with their laptops in one single room enjoying for some days the high-level communication and thus knowledge transfer of a physical meeting. Often, short presentations are held by the core developers and social events like community dinners follow the working days.

2.3.2.6. Revision Control Systems

Revision control systems are used in software engineering to record all changes of the source code. They allow multiple developers to work on one software project simultaneously and ensure no file revisions get lost. Most of these systems use delta encoding to note only the differences between two file versions rather than the complete files themselves. Thus the repository size is kept much smaller. Various systems are in use. The most commonly known is CVS (Concurrent Versions System), available since the late 1980's. A newer revision control system is Subversion (SVN) which has various technical advantages compared to CVS. An important aspect concerning revision control systems in OSPs is the difference between read-only and write access to the code repository. Because in OSPs the source code is freely available the read-only access is also obtainable by everybody. On the other hand, to ensure control of the software and its quality the write access, usually called commit access, is only granted to experienced developers of the project. Newcomers can only write patches, small code changes saved in files, which are then implemented by the committers because only these people have write access to the code repository. The 'joining script' is the level and type of activity new developers have to go through before receiving commit access.³⁰

28 See miniGuide (2005)

29 See Wikipedia (2005)

30 See von Krogh/Spaet/Lakhani (2003), p. 1227

2.3.2.7. API

The API (Application Programming Interface) refers to the overall structure and design of a software. An API provides a set of routines, protocols and tools for constructing software applications.³¹ In object oriented environments the API specifies how the behaviour and state of classes and objects is accessed.³² It defines what routines are available, how to call them, and what they do, but it does not explain how the subroutines are implemented and what algorithms are applied.³³ So the API tells developers how to access e.g. core functionality of a software framework in order to write their own application.

2.3.3. *Reasons for Community Building*

Reasons for community building are recruiting new contributors as well as improving collaboration and the project itself. In his research about OSP ecology Koch found evidence for the intuitive assumption that input and output of projects is correlated. He concludes the attraction of a large number of developers is vital for the success of an OSP.³⁴ So one of the reasons for the initialisation of the investigated content management system (CMS) Magnolia was to ensure the further development of the software apart from the internal development effort at obinary, its founding company: *“Another goal of course is that we don’t have to keep on developing the CMS by ourselves alone so we hope this will change in the future, too. Also we hope Magnolia helps us finding more skilled developers who are motivated by working on a well-known open source project. Such persons don’t have to work on a project which nobody will see but can participate in a cool open source project and can also make a name for themselves in the open source community.”*

In Cocoon there is already a large developer community pushing the progress of the project. Here the continuous improvement of the already available source code is important as Delacrétaz knows: *“Maybe somebody even sees my code and thinks ‘Oh I can make that better.’ so he improves my code.”* He also adds that the different views of the various community members can be an advantage for the project: *“Also*

³¹ See Webopedia (2005)

³² See Gale IT (2005)

³³ See Java Programming (2005)

³⁴ See Koch (2004), p. 81

writing tests could be something for non-core developers. Maybe it's even better if there are others who write the tests because they have a different view." Although the Cocoon project is already very advanced there are still large enhancements possible: *"There is one major change coming concerning the configuration of the functional Blocks. Now you have to compile them but we would like to do this more dynamically at run time. So you would just need to download the Blocks and plug them into Cocoon. This will be a big internal change in the future. Otherwise I believe we've used the 2.1 release for almost one year. It's stable and you can do a lot with it. I think it might even stay like this for the next couple of years. Maybe 2.2 will include the new Block system."* So a large amount of work has already been accomplished, but the documentation could still use improvements: *"Technically there will be not much change. We speak about documentation almost every week so I guess we will improve in this area. We also see many new names on the users' mailing list, names we don't know. So this means it grows and more and more people use it."*

Besides objective advantages like source code improvements and enhancements there is also a psychological component in community building. For Rothfuss a growing community is important for his own motivation: *"If I'd work on a project that not many people are interested in, I wouldn't be motivated enough."* Also Wesdorp enjoys the increasing attention of software developers for the Kupu project at the moment: *"It's good for the project obviously. More people use it and more people develop it. It depends whether it's not going to be chaotic. You don't want something chaotic. But as long as it is managed in a bit of a proper way I think it's nice to have a large community of people thinking about and discussing it. And it's just fun in a way to see something growing. That's probably what I personally like about the project."* In general Wechner states one doesn't have to fear a growing developer community because the code commits, too, can be monitored by the community: *"Actually there is practically no committer who harms a project. They are usually very careful checking in code. In the worst case we can always rebuke the people about their behaviour. We also have a commit mailing list where we get an email whenever there is a commit."*

2.3.4. Characteristics of Desired Communities

Before starting to recruit new contributors for an OSP the leaders should be aware of what kind of community they intend to build for their project. The interview responses showed there are certain common characteristics of desired community members.

2.3.4.1. Productivity

Most important, the community has to be productive and really contribute to the project. Wechner thinks this is not self-evident since there are often political issues, too, that drain energy from the communities: *“Of course it [the community] should be able to create something and not to get stuck in discussions and let the project stand still.”* Describing what kind of people Wesdorp would like to meet in the Kupu community he also names the productivity aspect: *“I think I’m quite happy with the people we have right now. Of course I would be happy for just about anyone joining. As long as they try to be productive I definitively stimulate everyone to join if they can.”*

Besides the helpful community members there appears to be the well-known issue of people and companies just benefiting from the software without contributing anything. Wesdorp mentions this problem in relation to the Kupu project: *“There seems to be a group of people that do not really understand that if they benefit from something it would be nice if they contributed something. It’s not that I wouldn’t want those people as users but we can’t really use them in the community. I think that counts for any community, that’s just a general rather than a personal issue specific to Kupu.”* Wechner even believes there is only a small amount coming back to the project of what people actually accomplish with it: *“Yes, [the community participates] a lot. Unfortunately many things never get published in the community. Sometimes I see that it’s just the top of an iceberg that gets back to the project. For example just some days ago somebody said something about DocBook so I asked him what he did with it. He answered he had built sort of a document management system for his company based on Lenya.”* Answering the question of how such people could be motivated to contribute their enhancements back to the project Wechner states: *“It’s difficult. You shouldn’t embrace people too much and invite them too openly because they might be shocked and run away and never show up again. Still I*

asked him if he would like to check the code into the project and after a while he answered yes, he could do that.” Also in the TYPO3 community there are freeriders but Hinderink thinks such behaviour is just stupid: “It’s almost the nature of an open source project to attract freeriders. There are many people who use it, earn money with it but don’t contribute anything. That’s usually also the majority. And there are times when you get frustrated about this fact. It’s especially not funny when the other one acts relatively aggressively. For example an agency realizes a lot of projects with TYPO3 and creates new functionality along with them. And still after getting the investment back they are not willing to publish the additional code e.g. as an extension. [...] There are a lot of agencies who save hundreds of thousands on licensing fees but still don’t donate anything. In some respect I also think that’s just stupid. The involvement in such a system is also some sort of long term egoism. You want the project to go on existing so it’s very stupid not to do anything for it.”

2.3.4.2. Self-Motivation

To be productive in an OSP one has to work independently and thus needs a high level of self-motivation. To achieve something in projects like TYPO3 the tasks have to be executed from beginning to the end: *“To work independently is almost the most important capability such people need to have. You don’t sit together in an office and usually you can’t meet; so people have to be able to work on their own. [...] Kasper calls this ‘selfstarting fireworks’: somebody makes a proposal, writes down the details and is still open for criticism. Others look at the draft, propose improvements and the initiator implements them or defends his own propositions. In the end he has to be motivated to do everything all by himself without expecting Kasper to be extremely thankful or some other positive attention of the community.”* Usually the various contribution barriers allow none but the sufficiently eager contributors to succeed.³⁵ Raymond calls these barriers “filters” that developers actually have to pass to contribute something useful.³⁶ And Delacrétaz describes how the lack of documentation may also have a barrier effect as described later in detail.

³⁵ See von Krogh/Spaet/Lakhani (2003), p. 1231

³⁶ Raymond (1999), p. 32

2.3.4.3. Diversity

To ensure the longevity of the OSP it is advantageous to have a variety of differently skilled members in the community as Wechner notes: *“Diversity helps to guarantee stability by including different perspectives.”* For a new project to become accepted by the Apache Foundation there is even a certain limit on how many of the developers may come from the same institution: *“Apache didn’t want to become a second SourceForge so they introduced the incubation process. In this way they can see e.g. if the community works. One common problem with donated projects is that they are often driven by a mono culture - all the developers are from one company only or there is only one developer. For the long term it’s important that the projects have a diverse community. So projects which come out of the incubation time may not have more than 50% of the developers from one single company. Lenya - as our project is named now - achieved this and we even became a top level project this summer.”*

Concerning the different levels of skill and experience Farstad is open to any kind of community members and enjoys the diversity in the eZ publish project: *“I like all the community members. Of course we have all different types of members in our community. Some are just testing it for a few days and trying to install it for the first time. Others are experienced and maybe have been there since the start. It’s not that we would prefer one in front of the other.”* Looking at the different tasks that need to be accomplished in an OSP there results a long list of diverse jobs. Rothfuss tries to outline them shortly: *“Certainly someone will be the promoter. He doesn’t code much but he’s active in many different communities. He will do the marketing for a project and will make sure it’s mentioned all over the place. Of course good coders are needed and people who like to interact with the community and help new users with their problems. And then people are needed who look after the infrastructure like CVS and so on. This might be delegated to some service provider like SourceForge. Also people must be found who do releases and maybe also a project leader is needed who tries to control the general direction of the project. Also documentation must be done by somebody.”*

2.3.4.4. Correct Behaviour

In order to guarantee the sustainability of a community it is important to maintain good communication behaviour among the members. Wechner defines a good community as “...one that is honestly friendly.” And for Farstad a healthy community includes a high responsiveness: “Basically if you can ask a question and get a good, relevant answer then you have a good community. And also if you can discuss future features and improvements.” Unfortunately there are still people who don’t follow the rules as he continues: “Basically [these are] people who don’t follow the etiquette, for example having 1000 exclamation marks in their subject. Then I think I really don’t want to answer this message. Sometimes they just don’t know how to behave. So that’s what we have moderators for who get rid of all the exclamation marks.” One way to educate new community members is to inform them about the communication habits established in the particular OSP for example by offering them a mailing list guideline as in Lenya: “[We have] just the standard Apache mailing list guidelines. Communicating via the mailing list is an advantage for the efficiency but it’s also good for the culture in the project. To be friendly, and humble...”

Another issue are repeating subjects in the mailing list discussion which should be avoided in order not to scare away productive but impatient contributors. So to focus the online discussions on new topics only, Kraft expects participants to read through existing documentation before asking questions on the mailing list: “Everything can be a barrier. Just subscribing to the right mailing list might be one. It’s not that easy in our project because I want people to first read through the Wiki and the mailing list archives before posting something on the mailing list.” Also in TYPO3 there is the notion to inform oneself first: “People have to follow the rules like first using the FAQ and searching for an answer in the mailing list archives. Since the community is mostly defined by the mailing list the things important for mailing lists are also important for a healthy community.”

2.3.4.5. Altruism

Obviously, contributions are essential for a productive community – but they are not everything. As Rothfuss points out it’s the community that plays the essential part in a project, not the single contributor: “Usually you don’t want to have people on an ego trip or running amok as soon they get commit access. [...] If a single person

codes a lot but harms the community by his actions it's not a healthy project. The situation might change and he suddenly might not be interested in the project any longer. So it was great that he contributed so much but it's bad he destroyed the fertile soil for others to join. Open source projects are not one-man shows."

Often community and business interests are opposed. Being in the position of offering services for Lenya with his software company and simultaneously being one of Lenya's leaders Wechner knows the challenge of this situation: *"You need to learn to distinguish between issues that only concern the open source project and thus must be discussed in public and between issues that are company specific, which you don't want to communicate to the outside world. After all, everyone on the mailing list has his own interests and that's ok. A problem arises only if you push your interests in ways which usually aren't possible for everyone in a community like consciously leaving others outside of the decision process. In the long term that's bad for the community."*

So in crucial situations it is especially important to bear in mind the importance of the community and not to come to self-centered decisions although technically they might be possible: *"Take e.g., backwards compatibility: If somebody breaks backwards compatibility in a stable branch then all our customers won't be happy any more; and not only our customers but all the people who use the software. I will think of all our customers who will need an update so it's bad for us and all other installations of the CMS. And yet, I won't go and say 'We don't do that.' It's the way of communication. We don't say: 'We are Wyona and because we don't want to have that we don't do it.' You can't communicate like this, but you can say 'I know customers who will have a problem with this change.' And it won't be different with others [having customers who use the software]."*

2.3.4.6. Perseverance

As Bertrand states long-term dedication is one of the main prerequisites for a new developer to get commit access for Cocoon because introducing somebody to the source code, the community members and their habits always involves some effort: *"So in the PMC [Project Management Committee] we discuss about the applicant 'What do you think, is this guy ready or do we have to wait a little longer to see more?' We try to evaluate all the criteria because we want the committers to stay in*

the project for a long time. This is important for us because it's some work to introduce somebody to the project. Or of course it's ok if they say 'I can only work during 6 months but intensively.'" The Cocoon community has learned this lesson from experience: *"It did happen a couple of times concerning the documentation: Some people came and said 'Hey your documentation is nothing, I can do it way better.' So they developed large plans how they wanted to do that but after a few weeks we didn't hear anything any more. And that's what we fear, people who come with very great plans and then nothing happens in the end - sometimes it does, but usually not. So we try to find people who make their way into the community gradually, learn how to deal with others, learn how to handle conflicts - of human or technical nature."* In eZ publish, too, long-term participation is valued highly: *"Another point is to have the same people not only for one week but for several years. That's what we have, members who've been around for three or four years."*

2.3.4.7. Common Vision

One of the challenges for a community is to be productive by focussing on the same goals as Wechner briefly describes: *"The community must be able to find consensus and at the same time define priorities."* The Kupu project members, too, seem to agree upon a common vision: *"It's nice to be talking about features and directions to go. A group with really completely different people that still have the same goal. That's just cool to watch that."* The difficulty is to be open for new directions and at the same time focus on certain strengths. Delacrétaz describes this process of finding a common vision in the Cocoon project: *"Back to the vision: The point is Cocoon is very open and we are just starting to call it a web application platform. This is very open. [...] So the vision is very open and it took quite a while to get one. I think we have it now. [...] The vision is to become an XML based application platform. This is very broad so we are always discussing where we are going, what we are doing, what has high priority, what is of low priority and so on. But at least we have the clear aim to keep the core of Cocoon as small as possible. Everything else is modular."*

The TYPO3 project appears to be an exception: there hardly seems to be a common vision as Hinderink states: *"I think there are about as many visions as there are community members and also the expectations are different."*

3. Initialisation of Open Source Projects

This chapter describes in a twofold manner the optimal environment to start off a new OSP. First, human based conditions are examined to give an overview of the positive leadership skills and behaviours of persons who are in charge of the new OSP. Secondly, the necessary decisions and the positive preconditions of the project itself show the settings that promote the success of the OSP.

3.1. Leadership Skills and Behaviours

This chapter summarizes the optimal skill set and behaviour patterns of the leaders of an OSP. Of course such skills can be learned any time during work on the project or people with the required skills can be motivated to join the project. Nevertheless it's very helpful for the initiators to possess as many of these skills as possible. So this chapter is part of the initialisation process of an OSP. As any order of relevance would be arbitrary the skills and behaviours are listed in alphabetical order.

3.1.1. Assertiveness

Judged from the number of mentionings in the interviews assertiveness is not one of the most common yet an important leadership skill of persons in charge of an OSP. Especially release managers need the ability to assert themselves as Kraft experienced in Magnolia: *“The release manager must be somewhat strict and decide when we're doing a new release. It's not typical for an open source project.”* Rothfuss explains why assertiveness is not a typical way of interaction in OSPs, answering the question about the lack of a feature freeze³⁷ in the Xaraya project: *“In a way there was [a feature freeze] but nobody wanted to stand in the way of somebody who was doing something - like in many other open source projects. If some people are checking in new features all the time it's very difficult to say stop. But these features have to be tested again and have to be mentioned in the documentation.”* In contrast to the important attitude of openness release managers must be able to enforce feature freezes to guarantee the on-time shipping of new software releases. Bühlmann remembers an incident in the beginnings of the Plone project where this was not achieved: *“For example the transition from Plone 1 to Plone 2 was very difficult be-*

³⁷ Definition “feature freeze” in FOLDOC (2005): Locks out modifications intended to introduce new features but still allows bug fixes and completion of existing features.

cause the developers wanted to implement too many features. So the product never got mature and release 2 was delayed. One company from Austria then said that they needed Plone 2 because one of their projects required features of release 2. So they began to develop their project on a beta version of Plone 2. But because there were so many changes all the time they had to rewrite their software every week. Finally, Jodok Batlog wrote a long email where he complained that he got sick of this because they had to start from scratch every single week. He demanded a professional release management.” This demonstrates the necessity of having strong OSP leaders in the community who take the responsibility assuring strict release management. Summing up the skill of assertiveness, Wechner explains why strong and thoughtful leadership is crucial for building a healthy community: *“Nevertheless I believe in strong leadership. This doesn’t mean shouting: ‘Shut up everybody, I’m talking.’ This is a misunderstanding of strong leadership. It means to be very clear but still open for consensus, bring others to consensus, and without compromise draw a line to those who aren’t willing to. Because if we want to be such and such a community everyone has to cooperate and if anyone doesn’t he’ll have to leave.”*

3.1.2. Commitment

Commitment in the context of leadership behaviour can be defined as an intense way of working on the OSP and investing lots of personal time into the project. Bühlmann knows Plone community members who do this: *“You would need two hours a day just to read all the emails, and in addition you’d have to be in the IRC chat. There are people who do that, they live for Plone. It’s become so broad. If you want to work with Plone in a company you must do that 100% otherwise you don’t have a chance. Plone has become very large and complex so you have to invest a lot of time.”* Hinderink observes the valuable commitment of Kasper Skårhøj, founder and leader of the TYPO3 project, and also knows the results of such a behaviour: *“I think it’s not really leadership which caused it [the TYPO3 community] to grow so big. From my perspective Kasper has two extraordinary skills. The first one is the impressive ability to concentrate. You can discuss with Kasper from eight o’clock in the morning until ten o’clock in the evening without getting the impression of him getting tired. [...] Thus he produces things above the average and with consequences. So maybe somebody proposes something, Kasper works on it in silence for weeks and then shows up with a result anyone else would have produced in no less than*

three months. I believe this is the secret of his success.” As initiator and core developer of Kupu, Wesdorp confirms this committing attitude by describing his own perspective of his role: “Concerning me I think it’s an advantage because I can look at it from a pet project kind of view. I can just decide ‘Hey I think it needs that.’ and can spend like 20 hours on that specific thing even though no one is asking for that.”

3.1.3. Communicativeness

Raymond believes that for OSP owners communication skills may be even more important than their design cleverness.³⁸ Developers must be attracted to join and be *“kept happy about the amount of work they are doing.”* Hinderink perceived the same in TYPO3 and also remembers another OSP where good communication led to high motivation: *“I believe it’s important to have somebody who thinks and communicates orderly, somebody who thinks in advance what the website needs - because this is somewhat the heart of the community. What information do I need to communicate to the community? And it’s important to take this really seriously. Looking at a project like Bricolage the guy is doing this really well. Although I don’t participate in the project I see the guy - it’s more or less a one man show - always makes a release and writes an email when he does anything new. He’s doing this consequently and brings it to a point, just writing what’s new and what are the tasks. I believe it’s enormously important to take communication really seriously.”* On the other hand not all software developers are highly communicative, especially not in written language. Wechner experienced this when his company intended to open development of Lenya obliging the core developers not only to chat when they were physically together in the same room but to communicate via the mailing list: *“You have to make your people aware that if they want to discuss something they have to do it via the mailing list. That’s not very easy. I believe in our project it’s working this way now but it wasn’t easy to achieve. You always had to say ‘Write it in the mailing list. Write it in the mailing list. Write it in the mailing list.’ People are not used to open communication, they usually communicate in a closed way. You can give them incentives, however. For example give them a way to become visible. But there are people who don’t want that or don’t need that. It’s important that they know that the project needs this open communication, otherwise it dies and this is bad for the company.”*

³⁸ See Raymond (1999), p. 47

Concerning uncommunicative head developers of OSPs Hinderink suggests to look for deputies who like the task of sharing information about the project: *“So if somebody as a developer is not able to do this [serious communication] - which is absolutely ok - he must find somebody who can do this and also likes to do it. Otherwise the whole thing doesn’t serve much, it just stays on the technical level and doesn’t get distributed. Neither will there be professionalising concerning building a community which helps and works together. The less information you make available in a bundled form the less people will be able to collaborate.”* Rothfuss, however, assigns a certain responsibility for communication to each software developer. He proposes the use of weblogs to accustom oneself to write more frequently: *“Traditionally developers were rather autistic and didn’t use weblogs. It’s changing now, I believe. With weblogs you can see better what the developers are working on and thus the reputation might improve. Also it’s a possibility to enlarge the fan club. Another effect is the developers get used to express themselves and thus the mailing list also benefits. It helps when people learn to write better.”* In almost all international OSP communities English is the primary language of communication although it might not be the native language of most of the contributors. Fortunately, a high level of English skills is not necessary for communication as Farstad states: *“I don’t think it’s that important, you just need a certain level. You don’t need too much language skills to join the community at all.”*

3.1.4. Experience

Working in a large software project with many thousands of lines of source code poses among other things the mere challenge of knowing about the presence of certain parts of code. Those who wrote most of the code, the core developers, are usually also the ones with the most knowledge about and experience with the source code. Bühlmann knows about the huge knowledge of Alan Runyan, one of the initiators of Plone: *“He does know a lot but today you can’t know everything about Plone. There are so many additional products. Still he’s the one with the greatest experience.”* Wesdorp even considers informing contributors about the current state of the OSP to be one of his and other core developers’ specific duties in Kupu: *“And sometimes I do have to point out that ‘Hey, wait, what you are trying to do is already covered by some helper library or something.’ [...] Since I wrote most of the API I know the API better than anyone else I’d say - maybe Duncan nowadays knows as*

much as I do. So Duncan and me are usually the people who help out with API questions.” Delacrétaz confirms this “highlighter” task in Cocoon where experienced developers respond to eager contributors: “Only if we see it’s already incorporated in Cocoon we’d tell them ‘Oh we’ve already got that.’ Often they just don’t know it already exists in Cocoon.”

3.1.5. Helpfulness

One of the very central attitudes of OSP owners is their helpfulness. In their empirical study about the OSP Freenet, von Krogh/Spaet/Lakhani found out, 89.5% of all mailing list participants received an answer when they initiated a dialogue by starting a new discussion thread.³⁹ Bühlmann observes the same characteristic on Alan Runyan: *“Yesterday, e.g., Alan Runyan, one of the Plone initiators, was in the chat. One guy asked a very basic question and Alain took his time to answer the question. This really helps community building if you patiently answer the stupid beginner questions, too, even if you are the main developer.”* Since newcomers to OSP communities usually do not possess personal relationships with existing community members they depend on the support of insiders. Wesdorp knows about the importance of this issue and suggests *“to make sure to help everyone who wants to join the project.”* In Kupu he observes it is not hard to get information by asking on the mailing list being aware of the positive impact this has on community building. On the other hand, time of the OSP owners is always limited as Delacrétaz notices. Developers receive most help when they show personal attempts and thus can specify their needs very clearly: *“So it works only if people do the programming by themselves or at least try and then ask for help for the remaining 10%. Then we see that they’re doing something. Maybe somebody can explain it to them and needs only five minutes for that. So people who have ideas but can’t finish them on their own will get lots of help by the community. [...] Just make sure people receive help when they come. Even if we think it’s rather a strange idea we try to make the people explain it.”* Since Farstad observed the multiplying effect of helping people out might recruit them as new helpers themselves, he formulated the following rule: *“You need to be interested in helping people out, to get them started. We have the philosophy ‘If you recruit one person to the forum he will probably recruit one or two more.’ So it’s really worth the time.”*

39 See von Krogh/Spaet/Lakhani (2003), p. 1225

3.1.6. Openness

Expressed in various ways, openness is the basic mentality of all the leaders of the examined OSPs. Openness is a specific skill of OSP owners since it is not the usual way of leading closed source software projects in the business world. The different aspects of openness concerning joining, choice of work, leaving and communication of an OSP are described in detail.

3.1.6.1. Open to Join

Bühlmann believes one of the main reasons why the Plone community grew so quickly is the fact that the OSP owners of Plone granted CVS commit access to all who demanded it: *“Right from the start they had created a SourceForge project and let everyone do CVS commits into the project. So from the beginning they gave everyone the chance to participate in the project.”* Nevertheless, quality assurance had to be cared for as Bühlmann mentions: *“Still they had kind of a code review. For example if somebody checked in something that didn’t work first of all they got a telling on the mailing list. And then these changes immediately got deleted. In this way a very open culture was developed where criticism is possible without generating a tense atmosphere. So one could easily change things, which worked out fine in the beginning because there were only about 30 developers.”* Also all interviewees confirmed that commit access for the repository is open to anybody who is willing to contribute to the project. Hinderink describes the joining script in TYPO3 as following the principle of meritocracy of the Apache Software Foundation: *“There is a good text on the Apache Software Foundation website concerning meritocracy. The same holds for TYPO3. There is quite some social mobility inside the community. It’s not an old boy’s club, many people joined only recently but already have lots of authority concerning the project. All entered about the same way: They proposed improvements and realized them - so they all convinced by their achievements. One can’t expect Kasper to grant access to someone just because he’s asking for it, jeopardizing the entire quality. If somebody makes a proposal, demonstrates how it works and what the advantages are it’s probable he’ll get access if he is doing so continuously. Nevertheless, the overall control over the project will remain with Kasper.”*

3.1.6.2. Open to the Choice of Work

In general, OSPs are also open in respect to the kind of contributions called the “*bazaar style*” of software development by Raymond.⁴⁰ As OSP leaders can’t force contributors to do certain tasks they are free to choose what to work on. Wesdorp knows this from the Kupu community but also has experienced specialisation tendencies: *“Everyone is allowed to work on every part but as usually everyone has his own favourite parts to work on and some stuff he benefits from. I think there isn’t really someone in the project that controls it and says ‘This should be done and this should be done.’ Everyone just works on the thing that he thinks is important for the project or for his personal benefit at this point. And so far it works quite well.”* The challenge of this leadership skill is to accept the work of others as Wechner says: *“The other skill is to be able to let go and let others do the work in their own way. [...] This is part of being open and letting people come into the project.”*

3.1.6.3. Open to Leave

As OSP leader one even needs to be open to leave the project voluntarily. Raymond called this his lesson number 5: *“When you lose interest in a program, your last duty to it is to hand it off to a competent successor.”*⁴¹ Wesdorp confirms this with his own attitude of working in the Kupu project: *“Personally I’m a hacker who likes to work on new projects and I’d love it if these projects could develop a life of their own. That’s more or less what I’m trying to do with Kupu. It’s not that I want to step out of it completely at some point but it would be nice if most of the work and thinking could be done by someone else.”*

3.1.6.4. Open to Communicate

To start community building communication must be opened to be available to a broader audience as Wechner states: *“One thing that we did consciously was to open our development. Initially, the developers came only from our company so we had a lot of internal communication. So to make sure that the outside world was informed about what was going on we consciously decided to communicate via the public mailing list only. So now we practically don’t communicate about Lenya internally*

⁴⁰ See Raymond (1999), p. 21

⁴¹ See Raymond (1999), p. 26

any more. This is very important because otherwise it would seem like we did something for our private business cut off from the outside world. I've heard of projects where this became a major problem - even in Apache projects."

3.1.6.5. Limits of Openness

Although open communication mainly has advantages Delacrétaz describes a certain situation where business reasons limit the openness of the developers: *"Everything has to be done as open as possible. Sometimes it's not possible because if somebody has a big contract with a company he can't tell publicly for whom he is working. Maybe he has sort of an NDA, a Non Disclosure Agreement, which tells him not to betray the customer's name. [...] Everything we do is open. Everybody can see one another's code and so on."* Openness can also do harm on a personal level e.g. during the election of a new project committer. In such situations strong confidentiality of the OSP leaders is necessary to avoid conflicts in the community. Rothfuss explains: *"One problem in open source projects is it's common to do everything as open as possible, like over the mailing list and so on. On one hand this prevents rumours like 'Oh this is a club where everything runs behind closed doors. The code is available but all the discussions are secret and I can't join them.' On the other hand it can be quite difficult to live with. For example when I'm a committer and somebody gets nominated to become one and I have doubts about this person, saying this on the public mailing list would be like hitting this person into the face. Although my doubts might be justified, they still shouldn't be announced all over the place. So the voting is done mostly via a private mailing list and only the result gets published. You don't want to attack these persons personally but still you would like to be able to express your thoughts uninhibitedly. Someone e.g. might not yet have contributed enough to the project or doesn't know how to behave in this community. If this happens internally the decision might be to wait a little longer and this person never knows he's watched until it's time."*

3.1.7. Patience

As OSP leader it is indispensable to possess a good amount of patience as many of the interviewed persons experienced. For Wechner, it's important to work with a long term perspective and also Wesdorp recommends: *"And do take your time. That's really it, it needs a lot of time, a lot of patience, a lot of openness."* Even the at

present very large TYPO3 community grew slowly at first after Kasper Skårhøj published it in a PHP news channel, remembers Hinderink: *“He opened up a new website and first announced it at Hotscripts. It didn’t start like an explosion but grew continuously. Only in the last two and a half years did it grow dramatically.”* Besides concerning community growth patience is also necessary in communication issues, for example repeatedly answering the same questions as Wesdorp practices in the Kupu community: *“But currently our conversations are not logged indeed so it does happen that people ask the same questions twice but then we will answer twice, it’s not that bad.”* Even as initiator and core developer of Plone, Alan Runyan, too, takes his time and answers beginner questions on the IRC chat as Bühlmann observed.

3.1.8. Personality

Linus Torvalds is *“a nice guy who makes people like him and want to help him”* as Raymond remarked.⁴² He also says *“it helps enormously if you have at least a little skill at charming people.”* Similarly, Hinderink perceives the presence of the TYPO3 initiator Kasper Skårhøj as giving a very personal touch to the project and the entire community: *“There’s a much more personal relationship to the project if there is an individual programmer at the centre. [...] Personal wishes of the project owner like ‘I always wanted to communicate in a friendly voice.’ also give continuity to the project. So it stays more personal.”* Charisma also helps when speaking at conferences, especially in front of media representatives. Bühlmann remembers a presentation of Alan Runyan: *“Additionally in Las Vegas there was the Computer Associates Conference in September of this year [2004] where the Plone project was presented. Alan Runyan was on stage talking in front of 3000 journalists in the room. This also promoted the project.”*

3.1.9. Presence

The continuous presence in the community, either physically or virtually, is a very positive behaviour of most OSP leaders. Bühlmann observes the high availability of the core developers of Plone: *“It was high in the beginning and it’s still very high. Sometimes I get the feeling the core developers are in the chat almost 24h a day!”* And Kraft, initiator of the Magnolia project confirms this behaviour by actively

⁴² See Raymond (1999), p. 49

participating in the mailing list: *“I wrote about 400 emails. A quarter of the entire traffic we had from November 15, 2003 to November 15, 2004, during the first release.”* Physical meetings are also very important so the company eZ systems lets Farstad travel around the world to strengthen personal relations with community members of eZ publish: *“My role is software developer and I’m one of the main designers behind eZ publish, me and Amos. I also led development of eZ publish for the past five years. Now I’ve got another job: I’m in charge of the open source relations as we call it. So I’m working a lot now with the community and basically getting the message out to the people.”* OSP owners usually enjoy meeting their community so Bühlmann could motivate both initiators of Plone to participate in a developers’ meeting in Switzerland: *“I then asked him [Alan Runyan] if we shouldn’t organize a Plone Sprint, I would organize one in February 2003. So he promised to participate and since Limi attended this Sprint, too, I got him on the participants’ list as well.”*

3.1.10. Programming

Raymond has a clear opinion about the fundamental programming skills of initiators of OSPs:⁴³ *“A certain base level of design and coding skill is required, of course, but I expect almost anybody seriously thinking of launching a bazaar effort will already be above that minimum. The free-software community’s internal market in reputation exerts subtle pressure on people not to launch development efforts they’re not competent to follow through on. So far this seems to have worked pretty well.”* Bühlmann confirms this hypothesis by pointing out the long software development experience of the founders of the Plone project and the importance of having such people in the OSP community: *“In addition the combination of people was favourable. They were not freaks but people of at least 30 years, i.e. not so very young. [...] Plone isn’t a toy. Nothing against PHP but this language can be learned quickly by computer newcomers and they can create applications with it. [...] These jobs require a well founded education in computer science to be able to use Plone for developing applications. You need to know what happens behind the scenes. [...] The members of the core team are all very skilled software developers. Part of them has come from the Java world with about 10 years of experience in software engineering. This is obvious. The people working on Plone really knew what they were doing and how it*

43 Raymond (1999), p. 48

should work. They are software architects who don't just start coding but first think about how to design the system to make it extendible in the future." In the end, the only thing that counts is the software as Hinderink states. So high programming skills of OSP leaders like Kasper Skårhøj are indispensable to develop a great application: *"It's mainly his work. It's the same in about every open source project. If the product is not good, you can communicate as much as you want, it doesn't help. Of course both must be present. If you want to grow you have to communicate well. But you can't sell bad stuff."*

3.1.11. Responsibility

In contrast to users and extension developers, the OSP owners are interested in the overall success of the project. Their long-term perspective assigns them a large responsibility. Wesdorp summarizes this duty as follows: *"I think it's always important that there is someone who is really taking care of the project in a sense that he always keeps an eye out if things are not really going wrong and that sort of thing. And there is quite a large probability that I'll always be that person and that people somewhat depend on me. There should be always someone who answers emails on mailing lists that don't get answered and that sort of thing. I don't see someone else picking that up easily."* Later on he describes some examples of how this responsibility is expressed in the daily activities of an OSP owner: *"Usually if you are a developer of a team you answer to the emails you find interesting or important. And that means that some of the emails will not be answered. And I think it is important that those get answered, too. So that's something I still do a lot for the community."* Similar answers were given by Ian Clarke while talking about the Freenet project.⁴⁴ As owner of this OSP he remarked the following: *"You've got all of these tasks floating around and then people take tasks that they want to do. I think what makes the core programmers special is that they're willing to do the tasks that no one else does because I think they take a greater sense of personal responsibility and a more long-term approach."* Besides taking care of the leftover but necessary work OSP leaders must also assure healthy communication, Hinderink is convinced: *"I think the tone is very important. There must be people around who take care of the way of how people deal with each other. Communication has to stay factual, with humour but not too much."* Delacrétaz holds the same view and also shows the positive effect

44 Compare von Krogh/Spaet/Lakhani (2003), p. 1230

polite communication has on community building in Cocoon: *“I think respectful communication is very important for us. If a conflict happens on the mailing list, for example people get angry and disrespectful, we interfere immediately. I believe the Cocoon community is very good. I’m there since about 2000 and I think since then only two people left the community because of disagreements. That’s not much compared to the about 100 daily emails on the developers’ mailing list. Of course we don’t agree all the time but as long we can discuss openly and with respect it’s ok. But if for example somebody says something wrong another might write back ‘Please apologize, what you said was not right.’ It has happened a couple of times so far. After some incidents people start acting respectful automatically. So when I realize I wrote something wrong I reply myself: ‘Oh I’m sorry, I didn’t mean it like that.’ I believe this is very positive.”* Next to solving communication conflicts OSP leaders also have to make an extra effort to support the community when they introduce new parts of the software. Hinderink calls this the principle of “completeness:” *“There is Kasper’s ideal of ‘completeness.’ He doesn’t just throw something at the world. He develops something, writes documentation and does lots of other things and only when all the dimensions of the package are covered and it can be presented he actually publishes it. I believe this is extremely important.”*

3.1.12. Visionary

As initiator and OSP leader it is essential to follow one’s own vision of the project but also to justify one’s reasoning as Wechner says: *“Besides some amount of this good humbleness it’s important to communicate your vision clearly - and to give arguments why you have this specific vision. For example explaining why one believes a trend has arrived and how to react to that.”* The founder of Cocoon, Stefano Mazzocchi, is not much involved on the programming side any more. He is, however, still engaged in the future of the project according to Delacrétaiz: *“He’s not very active on the code but rather concerning the vision and so on.”* Also Hinderink was and is a visionary person for the TYPO3 project: *“In the following year 2002 I was already involved a lot and wanted to take an active part to find out where to go with TYPO3 in the future. I participated in ongoing discussions, especially the one about the introduction of a modular concept.”*

Especially for software companies it is vital to advance their products continuously and to have a clear direction for this. Kraft has the strong vision to improve the Magnolia CMS further in the enterprise CMS area: *“But there is still a large potential inside the CMS area. We would like to advance it especially in the enterprise CMS area because Magnolia already has some important features like clustering and separation of authoring and live environment.”* Although Wesdorp’s desire to advance Kupu is high at the moment other technologies set limits for the WYSIWYG HTML editor: *“As I said I would like to rewrite the whole Kupu but this may not be in one year and not in five years either. We are just aimed for stability and hope for new browser features. [...] It’s one of the goals of Kupu to employ new browser features as soon as possible. But unfortunately we are stuck with Internet Explorer because currently it looks like development there is almost dead.”* While working actively on the Kupu project Wesdorp is already visionary about his next project: *“That project is a pet project of mine that I’m currently the only developer of. I’m going to release that and I hope probably in December. It’s going to be a JavaScript WebDAV client which does sort of all the content management tasks completely from the client. So the only thing you need is a WebDAV capable server and then you just can login to that and create new directories and new files and that sort of thing. It’s going to be released in the GPL so the idea is after the 0.1 release has been done people can join - maybe I can build a community too, we’ll see. As I said I like starting up new projects but at some point I like to see them getting picked up by others, too.”*

3.2. Prerequisites of the Project

In contrast to the chapter above, which covered person-related properties, the following chapter discusses prerequisites of the project itself. Initially, the programming language and the open source license of the new OSP must be chosen, which has a large influence on the further evolution of the project and mostly can’t be changed any more afterwards. To start with, a usable part of the software must be present, as well as the demand for such a software. Its degree of novelty and the possible applicability define the range of the project in the OSP market. Furthermore, a high level of communication of the initiators promotes community building strongly in the critical starting phase.

3.2.1. **Programming Language**

When starting a new OSP the question of a deliberate choice of programming language usually doesn't arise since the initiators already have their preferences or are limited by their skills to one language. Still there are important considerations to make since the type of programming language determines the future of the OSP to a high degree. Rothfuss dares a comparison between Java and PHP: *"If you look at the number of projects you'll find much more PHP projects than Java projects. Java isn't that easy to get into and see results as quickly as in PHP so it might not be that interesting for a single developer. Usually, Java projects are used in large scale applications so you find only few Java based message boards and so on. So there are also different kinds of people who are interested in these different software areas. Easy accessibility is the positive aspect of PHP and other scripting languages. On the other hand this may be a problem as you've seen in PHPNuke or PostNuke. Because the bar is much lower people quickly think they know all about software engineering. This leads to poor code and a bad position of the project itself."* The difference in the learning curve of programming languages is also confirmed by von Krogh et al.: *"Some languages are complex and difficult to learn, while others, i.e. simple script languages can be mastered fairly easily. Additionally, some computer languages are widespread and can attract a large number of potential contributors, while others are known by few, and thus raise contribution barriers."*⁴⁵

Farstad is convinced of the advantages of PHP for their content management system: *"We chose to use PHP. It is a language which is very simple to use and the learning curve of PHP is quite low. Also looking at the worldwide hosting providers it's the most used language as well. As far as I know it is the most frequently used programming language for web applications. That's good because it means you have a lot of potential users and contributors for your community. If we had used another programming language which is more obscure, harder to learn and not so widespread, that would have changed the whole potential for the user base. So PHP is a very good language for this purpose, at least this is our experience."*

45 See von Krogh/Spaet/Lakhani (2003), p. 1232

Although Python, the programming language of Plone and Zope, is not very popular Bühlmann likes the language because of its high technical advantages: *“Python is a cool programming language and we have many developers in Plone who’ve come from the Java world. The disadvantage of Python is that it’s not well-known. At the moment Java and .net are number one, then comes PHP and finally comes Python, I think. But Python has some important advantages: It’s a scripting language, so it’s not necessary to bother about compilation. Also if there are 10 developers in Java writing an algorithm there will be 10 different solutions. If there are 10 Python developers their solutions will be very similar. You can read Python very easily and you’ll quickly understand how a Python application works. The guy who developed Python tried to pick all the advantages of different languages like C++, Perl, Java and so on and combine them into Python. This is one of the reasons why Plone advances so fast: it’s so easy to understand someone else’s Python code! I believe problems can be solved with much less code lines in Python than in other programming languages.”*

In general, Rothfuss observes a certain evolution of the programming languages with a tendency to correspond to each other more and more: *“In PHP, VisualBasic and Perl it can be observed that the criteria of a well written software increase continuously. For example with PHP5 the difference to Java has shrunk again so if somebody really knows how to program PHP5 he shouldn’t use shortcuts like global variables and all this stuff. This means VisualBasic.net is much stricter than the old VisualBasic, they tried to move the entire architecture. Time will show if they will be successful. On the other hand there are efforts to integrate Java more into the scripting world. There are examples of JSRs [Java Specification Request] for this, like Jython or Groovy. These are implementations of scripting languages inside the JVM [Java Virtual Machine]. Or there are some starting points in Cocoon where there is Flowscript which is in JavaScript. I believe this grows together more and more so the differences between the languages won’t play a big role any longer.”*

3.2.2. Open Source License

Next to the programming language another important decision in the beginning of an OSP is the choice of the open source license. On the OSI (Open Source Initiative) website alone there are 58 different open source licenses⁴⁶ so the range is wide. The statements of the interviewees show the various considerations about the license they have taken for their own OSP or their opinions about the licenses of other projects.

Hinderink recommends to consider the choice of the open source license very carefully because later on it becomes difficult to change without losses: *“One important thing in the beginning is to think about what license to take. It’s not good to do that in a haste. You have to consider precisely what kind of contribution should come from the community, where do I want to be able to control things and where shall the forces play freely. A license can communicate and organize this to some degree. You can write your own or you can take an existing one certified by the FSF. [...] The problem is once you’ve decided e.g. for the GPL and later you change to something like the Zope License, many community members might get angry about it and feel exploited by the central company.”*

Rothfuss explains how the GPL influences community building and that the Apache License might be a reason why only few contributions are returned to the project: *“I believe for smaller projects the GPL is not bad because it would generate a bad mood if somebody just took the code and didn’t return anything. The GPL sort of forces community building for legal reasons because people have to give back their code changes of GPL projects. That’s different with the Apache License. Somewhat exaggerated, GPL communities are sort of forced communities and Apache communities are voluntary ones.”⁴⁷ There are many companies that use Apache code but never return anything. So in an Apache project it’s more difficult to get the people to return their code back into the project because you can’t use legal means. So you must find other ways to persuade them to contribute their development work, for example by offering them to maintain their own fork or by convincing them of*

46 URL: <http://opensource.org/licenses/> (2005-02-09)

47 In the OSI draft “Licensing HOWTO” Raymond explains in detail how copyleft licenses like the GPL influence community building compared to the non-copyleft licenses like the Apache Software License. URL: <http://www.catb.org/~esr/Licensing-HOWTO.html> (2005-02-22)

their contribution to the good reputation of the project. But if you want larger companies to adopt an open source project the GPL is not the right thing. We worked with customers who explicitly didn't allow the use of the GPL or LGPL because they were afraid of the loss of control over their intellectual property. If you want your software to be used as widely as possible you have to publish it under the Apache License." According to Rothfuss the GPL is not suitable for business use : *"I believe it's more difficult to build a business upon GPL code because the easiest way to make money with open source software is to keep the changes for yourself."* Delacrétaiz confirms the freedom given by the Apache License which may lead to greater business opportunities: *"To simplify, most everything is allowed with Apache licensed code as long as credit is given to the Apache Software Foundation. It's a very open license; you can even sell your software. But you have to mention that the code comes from the Apache Software Foundation. That's all. [...] Having a GPL or LGPL license would make a big difference, BSD not as much. The ASF license is friendly to companies so they can earn money with it."*

Kupu has its own, brief open source license in reliance on the BSD license. The popularity and the freedom of this license were the main reasons why the core developers decided to go away from the Zope license: *"That's when we decided to change the license to a better known open source license. So it's now BSD style rather than the Zope Public License it used to be. [...] Zope Public License, it's liberal and all that but it's not too well known. So if people hear Zope Public License... We wanted to get rid of the association between of what is now called Kupu and Zope because Epox used to be a Zope only thing and at that point we already decided that we wanted to have it working - because it was quite easy to get it working - on different systems. So that's why we decided to get rid of the ZPL and to put in a plain one, a better known one. It came to a C license. It's also easier for instance for the project we're working on at Infrae, Silva. It's also to be a D license so it cannot use something like the GPL, it has to be something more broad. BSD is just very plain and simple."*

Concerning the GPL, collaboration in such projects has to be coordinated well to really benefit from the power of the community as Hinderink experienced: *"The licenses organize at different levels. For example the GPL controls only few things*

so there must be other instruments installed to enable collaboration. All the marketing advantages of open source software also have their price. The widely distributed development bears the risk that contributions can't be put together again or that powers fritter away. So it's important to think about collaboration and control - which in the end is the construction plan of a community."

A quite different strategy is followed by the two companies eZ systems and obinary that both keep the copyright of their OSP. Technically eZ publish is one software only but it is available under a dual licensing model: two different licenses, one that allows free distribution and use and the other being a proprietary one.⁴⁸ *"I think we and MySQL are about in the same situation because we have the full copyright of the software. As an external developer you can't say I want to join the project and change this and that, because we do have all the copyright, because we have the dual licensing policy. This can be a disadvantage. Still we have external developers working on eZ publish as well but then they have to give the copyright to us."* Although obinary doesn't follow the dual licensing strategy external developers still have to sign a committer's agreement as Kraft explains: *"Until they can commit on their own they have to send us patches after signing some committer's license agreement. This is important for us since we want to keep the possibility to change the license. In general we had a lot of licensing questions lately, it's a very complex topic and unfortunately there are only few people who really know about it thoroughly. Simply said we are sort of afraid that if we took an Apache license someone could just take our project away."* In the TYPO3 project, too, efforts to receive legal support only resulted in new question about licensing issues: *"For every license there are questions. Strangely we get a lot of questions about the GPL and discussions. Once we tried to answer them with the help of a lawyer's office in Munich who gave a lot of responses. But this really had the opposite effect because afterwards there were even more questions about legal issues. It's just difficult for most of the people to think in these juridical terms - which is just normal."*

Concerning the influence of the open source license on user attraction of the OSP Hinderink can't observe clear tendencies although he always assumed them to be present: *"I can't really judge it. I always thought it has consequences if somebody*

48 See Välimäki (2003), p. 6

chooses the GPL compared to somebody writing his own license. Users would think there must be something strange about the project if it has its own license. But I think I was wrong. Most people actually using open source software are developers so they just get the code of the software and that's it. They don't care at all what's written in the licenses. There are cases where the license is very restrictive but these usually aren't real open source licenses either. Jahira is such an example. I believe if you use it commercially then a licensing fee has to be paid. I wouldn't call that an open source license because it's a handicap for distribution, no doubt. But with GPL it doesn't matter. Everybody knows it. If you take an Apache license or write your own one like Zope or Apple doesn't have a large effect on the spreading of the software. I always thought it would be different but I can't see the effect now.“ Yet before deciding for one license Hinderink recommends OSP initiators should inform themselves very well about those available and not just choose one for popularity reasons only: *“These are all topics you have to be aware of. It's not just done saying 'I'll put a GPL sticker on it and then you can do what you want.' I believe it's good to look at other licenses, even self-written ones - even if in some case the FSF (Free Software Foundation) doesn't accept the license. In the end people don't care much about the FSF. They just get the software no matter under what open source license.”*

3.2.3. Great Initial Source Code

There has to be some source code available when starting a new project as Rothfuss explains: *“It's always better if there is already code available instead of just ideas. In SourceForge there are many ideas that never catch on. If some code is already available it's much easier to find people because usually nobody has time to study ideas and plans. You can't submit patches for ideas. There are so many other things to do, so it must be attractive.”*

So to what degree should the software be developed to successfully initiate the project? Raymond states the most important thing is the visibility of the project's potential: *“When you start community-building, what you need to be able to present is a 'plausible promise'. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is convince potential co-developers that it can be evolved into something really neat in*

*the foreseeable future.”*⁴⁹ Bühlmann’s first contact with Plone was exactly like this: *“Quite soon the first release was made available. I downloaded it but found it was unusable because it was way too slow. But I realized the potential behind it.”* Still the source code should be of a reasonably good quality for publication as he adds: *“You should publish a software only when it’s quality is sufficient.”* Apparently, Magnolia was of very high quality when it was brought online in 2003 so the development team received a lot of positive feedback from the beginning: *“Astonishingly there are many people who started working with Magnolia release one and were immediately amazed by the beauty of the code.”* Although the source code can be modified at any point during the life of an OSP it is still essential to put enough effort into the architecture and create a sophisticated API design as Hinderink recommends: *“One other thing that should be thought of in the beginning is the basic API. I don’t know of projects that didn’t lose people when they did major changes in the API. For example imagemagick is such an example. It’s a great software but the history of the interface is weird so they’ve got a bad image from that.”* With TYPO3 the wealth of features it had already in the beginning was another reason for its successful start, since the software could be applied directly to new projects: *“I believe it was the sophisticated user interface and the availability of a large range of functionality like a simple address book and a small shopping system. There were about ten plugins that came along with the system. Compared to other PHP projects at that time the code quality was very high. Maybe the worst example is PHPNuke which started out small but many people joined and then control got lost. [...] TYPO3 was different because Kasper had already prepared a lot before publishing it. It was much more complete than other PHP projects.”*

3.2.4. Public Demand

As in other areas of economics it is very difficult to anticipate the demand for a certain software product. There remains a large amount of luck of doing the right thing at the right time. Wesdorp summarizes this effect: *“It’s so hard to say what people find interesting. That’s one of the things with open source, you can find something interesting yourself and you may have a lot of benefit from it. But if other people don’t find it interesting you’ll not find any developers. I don’t know, it’s really hard to say. It’s also something dependent on trends. It’s very hard to say if a project is*

⁴⁹ See Raymond (1999), p. 47

going to be successful or not.” Bühlmann, too, believes the success of an OSP remains unpredictable since it’s mostly luck to hit the right moment for publication: “It’s difficult to say when is the right moment - it might be always and never. You need a lot of luck and the right momentum. It’s as in soccer when Greece won the European Cup 2004: nobody would have guessed it. This can also happen in the open source area when the right momentum is present and the right people join the project and talk about it.” As mentioned the right people need to be interested in the project so a large user basis is positive for community building as Kraft explains: “I believe a community builds around a project when it’s interesting enough and many people are going to use it.”

One important aspect is to estimate the required level of skill of the OSP’s audience. Some projects are intended for users who have little programming skills, others like Kupu (*“There are usually only developers, hardly any users.”*) and Cocoon are rather for experienced software developers. Delacrétaiz gives an example concerning documentation: *“Bad documentation is not that bad for Cocoon because we agreed Cocoon isn’t a toy for ‘Joe User’ but a tool for developers. Developers can be very productive with Cocoon but there is a lot of experience necessary because of its combination of various tools.”* He has also noted different types of communities. Not many code contributions can be expected from a pure user community. The Cocoon community is different because people who intensively use Cocoon have a large knowledge of the software and thus can participate in further development quickly: *“Cocoon is really a tool for developers. Usually after some months they get into the code and so they can also help to contribute. This is a big advantage of the project. The users of Cocoon must have almost the same competences as the developers which means they can quickly change from users to developers. In other open source projects that’s not the case. For example I started Jfor some years ago, an XML converter to RTF, Rich Text Format, for word processors. There the user competences vary a lot. Users just want to generate documents so many of them don’t know Java and thus can’t help with the code. After several months there still was no community. It was much more difficult to grow a community because the people were different from those in Cocoon.”*

Keeping an eye on the market of the specific software category may also help to find the right moment for initialisation according to Rothfuss: *“It’s difficult to find a niche today. [...] Long ago I’ve realized that there is a ‘time to market’ for every product, some sort of a time window where there is a chance to succeed. After this there will be other projects that do similar things and bind their community to them.”* Changes in the CMS market were essential for the success of eZ publish: *“It has much to do with how we made the software but also has something to do with the timing. [...] When we started in 1999/2000 everybody went bankrupt. That’s the time when people look for alternative solutions, when they can’t afford a high end solution. Open source is just a natural place to look because the price tag is zero.”*

3.2.5. Degree of Novelty

The range of innovation in OSPs goes from replication of existing software architectures⁵⁰ to radical innovations called de novo design.⁵¹ The Cocoon framework presented a large degree of innovation in 1998 when it was the only XML publishing framework available. Delacrétaz states: *“There were already solutions around for that kind of job but they were not sufficiently open or mature enough. So the concept of Cocoon was indeed innovative at the time.”*

Nevertheless, even OSPs that are a replication of already available software always include some amount of new concepts and features. The famous open source project Linux isn’t a radical innovation⁵² yet in many aspects the most successful open source project of all time. Similarly, Plone wasn’t the first Zope application framework at the time of publication but it became a success nevertheless: *“There was another product called CPS, Collaboration Portal Server of Nuxeon company from France. They did the same, develop their own portal based on CMF. But Plone, apparently, was better in the initial phase so more people got into Plone than into CPS.”* Kupu is only one of many available open source WYSIWYG editors, but it features innovative properties. As Wesdorp states, Kupu applies modern properties of browsers: *“And what we try to do is leveraging modern techniques of browsers. Rather than using the old style JavaScript coding we are trying to use new features*

50 E.g. Linus Torvalds took the Unix operating system as template architecture for Linux, see von Krogh/Spaet/Lakhani (2003), p. 1218

51 See von Krogh/Spaet/Lakhani (2003), p. 1219

52 Raymond (1999), p. 29

of the JavaScript language and also newer features of browsers to achieve a better user experience. For instance the PUT, we use HTTP PUT rather than HTTP POST. You can use POST, too, if you want to but that's definitively something that appeals to any user." These novelties are recognized by the contributors of the project: *"A lot of people are impressed by what Kupu does. And I still get a lot of feedback from people saying 'Wow, I didn't know this was even possible in JavaScript!' I think the feedback is generally very good."* Implementation of new standards as in Magnolia is another way of innovativeness but also bears risks: *"Very early we started with this JSR-170 standard that defines how CMS have to access data in the content repository. At that time we didn't know if it would be published and if it would ever get any importance. Indeed there were people who found us because of our JSR-170 compliance so, apparently, we are interesting for a certain group of people. [...] We did all the effort, the GUI, the JSR-170 stuff and marketing because there wasn't anything like that at the time."*

3.2.6. *Applicability*

The broader the OSP can be applied, the larger is the potential user community of the software. Content management systems e.g. are one software category that can be used in many different ways: *"In addition the possibility to create your own templates is important for the community. That's how many different people got involved with the system. [...] People work with TYPO3 in many different ways. There are designers who create the templates, developers who write TypoScript only or some newbies who just do a mapping onto HTML pages. This is a good platform for people with different working methods. TYPO3 can be used in many different ways. Some use it to prepare printing, others manage the website of their association. This variety is very important for the community."* Also Kraft intends to offer Magnolia for different user groups with the application of specific extensions: *"It's possible to extend Magnolia for a special interest group like 'Magnolia for Group Collaboration' and also as a commercial solution."*

Usually there are design and platform decisions that broaden the applicability of the OSP. When Kupu was formed from the Zope based Epox project Wesdorp and the other developers favoured such changes: *"It wasn't very configurable and it was Zope-only. This was something we decided wasn't a good marketing aspect of the*

editor so we decided to change that, too.” In eZ publish the initiators planned to write a framework from the start on: “From the beginning we have programmed eZ publish as a general CMS, not for one customer only but as general as possible. [...] We wanted to make a web application framework to produce general web applications and we wanted to have the CMS on top of that.” So the advantage of frameworks like Cocoon is their extremely broad area of use as Delacrétaz explains: “The core of Cocoon itself is very small but it includes lots of tools and libraries. So for someone Cocoon might only be a system to publish XML to HTML, another one builds a large web application with databases, XML feeds, including multi channel publishing to web, WAP, PDF, images and so on.” This leads to a broad community: “I think the result will be a very open community. There are very active people in the team who work with very different technologies although they all work with Cocoon. We must be able to communicate because we have to agree on certain points. This is interesting because the people are very different and they are not doing the same work. There is a broad application spectrum of Cocoon so there is also a broad spectrum of community members.”

3.2.7. Level of Communication

As shown above the leadership skill communicativeness is in general a very important quality of OSP leaders. Since there were several explicit references in the interviews concerning interaction between developers during the start-up phase of an OSP a high level of communication in the beginning can be assumed to have a strong positive influence on community building.

Starting on the physical level, the core developers of Kupu met personally in the beginning: *“For Paul Everitt and me it has been nice to discuss some issues one-to-one, just for the start-up.”* Such synchronous communication can also be accomplished over IRC as was necessary in Plone to compensate the lack of documentation: *“Most important is the IRC channel. This was one of the most important elements in the beginning so people could talk to each other directly. [...] This was very important in the beginning of the project when there was not yet much documentation but people received help in the chat.”* To initiate discussions on the mailing list one has to be very active in the beginning before community members also start helping with the mails as Kraft experienced: *“You need a certain*

momentum which you only achieve if you're active in the beginning. As soon as you get a certain number of people involved they will also start answering emails." Although documentation might not be the most interesting task for the OSP initiators it can essentially influence community building as Rothfuss believes: *"For example I'd invest a lot of time into documentation although you have to write it over and over again. It's something I never really did, but I believe you'd be astonished what influence good documentation can have. Documentation just lowers the entry bar and gives others the chance to collaborate. Although nobody likes writing documentation it probably results in a better 'return on investment'."* What Rothfuss states as assumptions Kasper successfully executed in the beginning of TYPO3: *"Documentation was an important topic from the beginning. Kasper invested a lot of his time into documentation which is definitely unusual for a developer."*

4. Promotion of Community Building

While the chapter above treats the important aspects of the initialisation phase of an OSP this section is about further improving the “nutritive soil” for community building. Different ways in which the interviewees propose to influence the evolution of the community or actually did so by certain actions are discussed. To group the interview answers and pave the ground for a promotion framework they have been categorized into a two dimensional structure shown in table 3.

	↓ Subject Level →	<i>Recruiting</i>	<i>Collaboration</i>	<i>Production</i>
Spanning Subjects	<i>Modularity</i>	Chapter 4.1.1.	Chapter 4.1.2.	Chapter 4.1.3.
	<i>Documentation</i>	Chapter 4.2.1.	Chapter 4.2.2.	Chapter 4.2.3.
	<i>Release Management</i>	Chapter 4.3.1.	Chapter 4.3.2.	Chapter 4.3.3.
	<i>Collaboration Platform</i>	Chapter 4.4.1.	Chapter 4.4.2.	Chapter 4.4.3.
	<i>Physical Meetings</i>	Chapter 4.5.1.	Chapter 4.5.2.	Chapter 4.5.3.
	<i>Foundation</i>	Chapter 4.6.1.	Chapter 4.6.2.	Chapter 4.6.3.
	<i>Internationalisation</i>	Chapter 4.7.1.	Chapter 4.7.2.	Chapter 4.7.3.
Level Specific Subjects	<i>Spreading the Word</i>	Chapter 4.8.1.		
	<i>Credit System</i>	Chapter 4.8.2.		
	<i>Communication Channels</i>		Chapter 4.9.1.	
	<i>Community Structure</i>		Chapter 4.9.2.	
	<i>Task List</i>		Chapter 4.9.3.	
	<i>Code and Feature Quality</i>			Chapter 4.10.1.
	<i>User Interface Design</i>			Chapter 4.10.2.
	<i>Installation</i>			Chapter 4.10.3.

Table 3: Overview of the Subject-Level Promotion Matrix of Community Building

The subjects in community building form the rows of the table and are briefly described in each chapter. They represent 15 often encountered topics in OSPs and are split into two groups. The “spanning subjects” have an influence on all three levels of community building. Consequently they are the most important issues to be treated. The “level specific subjects” affect one of the three levels only and thus are more specific topics.

There are three levels of community building which form the columns of the matrix: recruiting, collaboration and production. Activities on the recruiting level intend to motivate new contributors to join the community and participate in the collaboration. Therefore topics like attractiveness of the OSP, distribution, marketing and external communication in general are discussed. Interventions on the collaboration level show how internal processes can be improved focussing on internal communication, knowledge transfer, human relationships, organisation and coordination. Actions on the production level concern the progress of the software itself describing how the source code, the API and the user documentation are improved.

4.1. Modularity

In software engineering it is well-known that modularity increases flexibility and comprehensibility of a system allowing a reduction in development time.⁵³ This holds especially for OSPs where developers usually work together only virtually. Well defined interfaces of the software parts are necessary to ensure smooth interaction of the contributions of all the developers. Modularization also has a high impact on community building as will be shown in the following chapters.

4.1.1. Recruiting

Since it's easier to write short, well-defined modules, also called extensions or plugins, modularity of the software lowers the entrance barrier thus making it easier to recruit new contributors. Writing new extensions is the main activity of the TYPO3 community. Hinderink has no doubts about that: "*[Most contributions are] definitely in the extensions. There were some donations but most of the activity is in the extension area.*" Kraft intends to achieve the same in Magnolia and build a community that writes new extensions for the content management system: "*With the module mechanism you can extend Magnolia so it's possible to build a large community in the extension area. That makes more sense than having people in the core since it's quite finished and you would have to know all about it. It's the same as in other CMS like Communicate of Day or in TYPO3.*" Answering the question about architectural aspects which might promote community building he confirms again: "*Certainly the module idea. When this was introduced into the TYPO3 project the community actually exploded.*" Also Wesdorp is convinced of the importance of

⁵³ See Parnas (1970), p. 1053

plug-in architecture so he answers concerning its influence on community building: *“I think it can be very important. Right now we unfortunately don’t focus on plug-ins that much. But I’m trying to shift from using the plug-in architecture not only to add core features but also trying to turn the core features into plug-ins and try to exploit the plug-in architecture by making it easier to add third party plug-ins and extensions.”*

So Hinderink explains the advantage of developing of new features for TYPO3: *“Being able to add new functionality at this level without touching the kernel and also implementing very easily - with the help of the quick starter - is very unique. For developers and agencies the very simple way of customizing is actually the only possibility to distinguish themselves because many use TYPO3 today.”* In spite of the huge functionality of TYPO3 it is very easy writing new extensions with help of the extension manager: *“Then you’re able to upload your personal extension into the TYPO3 extension repository. There is no control at all, the only limit is the size of the file and even this you can override by asking Kasper or Robert Lemke, responsible for typo3.org, to increase the limit of your extension. So the doorstep is very low, everyone can participate. And you can actually do almost anything. Like extending core classes or overwriting functionalities. The only difference is it’s done on a higher level so nothing will be gone or can be destroyed.”* Also in Cocoon it is the goal to keep adding new functionality simple as Delacrétaz is convinced: *“It’s quite simple to write such a Block so people can participate easily. [...] So since Cocoon itself is very modular it’s quite easy for people to contribute.”*

The integration of external developers into the OSP is the final goal. Farstad explains which technical properties of eZ publish promote community building: *“As I mentioned the plug-ins and the extension system are of course part of it. In about 20 or 30 lines of PHP code you have an extension to eZ publish like for example a template operator or a small module. This promotes third-party involvement in the community.”* Delacrétaz confirms the positive influence of modularization in Cocoon, too, since in this way many different developers and companies can use the publishing framework for their specific needs.

4.1.2. Collaboration

One major advantage of extension development is the possibility of many people working independently on the code relieving the core developers as Hinderink tells: *“Of course the single person project does have disadvantages. A major one is the non-scalability of Kasper - adopted from the Linux kernel community ‘Linus doesn’t scale.’ The creation of the extension repository was one attempt to decrease the importance of this problem so Kasper can now concentrate on the central issues again.”* In addition to the freedom of choice of topic the extension developers even have the power to invent new features: *“I’m sure that if for some reason Kasper wouldn’t be able to do something really necessary in the kernel somebody will do it as an extension. The advantage is something can be done very quickly. Of course that’s not always good since resources might get lost or because people don’t want to discuss or stay with the half finished solution. But for customer projects and short term solutions it’s ideal to work like that.”* In the eZ publish project, too, different kinds of contributions are possible: *“We call it contributions because we have many different kinds of extensions like workflows, template extensions, data types and applications. So we have like different categories. We have several extensions in our contributions area. There all the extensions get their own page with comments and descriptions and of course again the name of the author so he gets the credit.”*

Analysing source code contributions (commits) a high degree of developer specialisation on particular modules was found in the Freenet project.⁵⁴ Similarly, the kernel extensions of TYPO3 are all maintained by certain persons as Hinderink knows: *“Extensions very close to the kernel are mostly realized by people in the the sphere around Kasper. For example the workflow system is done by Christian Jul Jensen, a close friend of Kasper. There are others who offered to help, for example Martin Kutschker who wanted to take care of the typing system, especially UTF-8-compatibility throughout. He got active in several discussions on the developer list and proposed solutions. So Kasper showed him the classes involved and allowed him to take over. [...] Nobody does everything. Actually the only one who is active in all the parts is Kasper. All the others have found a certain topic and care about that.”* Also in Plone the community members are often specialised as Bühlmann noticed: *“Then there are many others who know a lot about one specific area like user management*

54 See von Krogh/Spaet/Lakhani (2003), p. 1230

or content types.” And Wesdorp experienced the natural evolution of specialisation in Kupu: “Everyone is allowed to work on every part but as usual everyone has his own favourite parts to work on and some stuff he benefits from.”

The advantage of easily developing new modules can turn into a disadvantage when there is a large number of extension commits like in TYPO3: *“There’s an issue about collaboration. For example if you see how many new extensions there are and how small their differences are you can see how important collaboration actually would be. Not to program five weak ones but to do a single one and plan and discuss what its features shall be. It’s something that has to be improved. [...] So such an extension repository gets spammed. For example recently I saw somebody who definitively meant it positively in his enthusiasm. He took about ten extensions, modified them minimally, changed their names and icons and committed them again. So these ten extensions weren’t necessary, it was rather a finger practice which should not have been published.”* Due to his long-time experience in the TYPO3 community Hinderink knows advice concerning quality assurance of OSP extensions: *“These are issues a community has to solve. So that’s the reason for the planned extension review process. There shall be a group of people who’ll do a review with strong criteria and rate the extensions. Additionally there shall be a user rating so there are two complementary elements which together form some sort of a quality seal. That’s the response to the problem. But it’s not easy to implement because highly qualified programmers are necessary for the rating and these guys usually have better things to do than reviewing other peoples’ code.”*

A modular architecture can be used in many different ways. This has the disadvantage of nonconformity of the manuals as Delacrétaz describes the situation in Cocoon: *“For example if one writes a tutorial it always goes in a certain direction. So the next one says ‘Oh but I don’t use Cocoon in this way, I can’t really use this tutorial.’ We have to find a solution and I also think there will be one. For example a book about a certain case of usage describing exactly how this works and so on - always in context with the reference documentation of the components.”* Another advice for solving this issue is given by Hinderink: *“So in typo3.org there is a documentation repository with all the files, maybe a couple hundred. There are not only those by Kasper but also by other people, mostly extension developers. You can and should*

deliver a documentation with each extension based upon an OpenOffice template. These are uploaded into the repository and transferred into HTML pages and thus a small manual website is created. Additionally the page can be viewed as PDF or again as an OpenOffice document.”

4.1.3. Production

The basic idea of modularization is to simplify the complex structure of the source code by breaking it down into well manageable parts. Rothfuss sees this advantage and mentions concerning the influence of extensibility of projects: *“I believe it [extensibility] is important. If you abstract it somewhat, it’s a modularization of the code. It’s one way for a community to grow. PHP for example is an Apache module. Just imagine PHP had to be developed inside the Apache Web Server - unrealistic! The same holds for the Linux kernel and its drivers for new hardware. Modularization is good because it makes large projects tightly structured.”* Kupu uses the principle of modularization even in the core functionality so everything is clearly arranged and ready for use by other developers as Wesdorp explains: *“There is also a set of tools which are basically plug-ins. Kupu works with those plug-ins even for the core tools. The basic tool set is already done by using the plug-in architecture. And then there is a set of helper classes to control the selection for instance and a large set of browser abstractions because there are still a lot of browser differences.”*

The positive consequence of such an architecture is that extensions can be kept small as in Plone: *“No more than a few lines of code are needed to solve a problem. We seldom write a module where we need more than 100 lines of code. Everything is connected with the framework so in the end you don’t need to program that much.”* Also the software remains flexible because only those extensions are developed that are really needed: *“We have the clear aim to keep the core of Cocoon as small as possible. Everything else is modular. This is like e.g. Linux where the kernel itself is not so big but there are many utilities and classes. Cocoon goes in the same direction, just acting as a kernel and then everyone can do with it what he wants.”*

In large software projects the integration of other OSPs as modules is fundamental so Cocoon offers this possibility: *“Then there are the Blocks, modular parts of functionality. They are completely modular so everyone can write a new one. [...]*

For example the PDF formatting is one Block. It uses the external component FOP (Formatting Object Project). So the Block is actually some Java code that integrates FOP into Cocoon. This Block consists of only a few lines of code but you still have to write it for the integration of the different properties and so on.” While functionality is important, in Magnolia also the GUI is relevant so newly added extensions shall be seamlessly integrated into the core system: *“Additionally we are planning to post projects that describe possible extensions of Magnolia. In release 2 we offer these GUI controls that allow third party developers to write extensions in the same look and feel of the application. Later on we want to write sort of an Extension Manager that lets you install the Magnolia extensions with one click just by accessing some sort of an extension repository.”* And Wesdorp even suggests the separate release of such plug-ins: *“Right now I think we still do too much in the core which is not that much different from actually writing the plug-in. It would be quite easy to grab the stuff from the core and release it separately. Maybe we do that at one point because it’s nice that you don’t have to care too much about all kinds of extensions and like niche areas if you do a core release. So it makes maintenance easier for sure.”*

4.2. Documentation

This topic includes all the various aspects of documentation in relation to an OSP. It considers all documentation artefacts like reference manuals, tutorials, feature lists, guidelines and end user documentation as well as the process of creating and structuring these informations.

4.2.1. Recruiting

Documentation is a very important factor for the recruiting aspect of community building. Hinderink mentions this influence in TYPO3 where a large amount of documentation is available: *“The documentation always had a large impact on community building. TYPO3 has documentation above average. And there was always lots of discussion about the quality. I believe it’s quite high although there isn’t a guide or a document with a general overview of all the components so far.”* Especially to start using an OSP certain basic documentation needs to be available as Wechner says about Lenya: *“Also there is documentation for the integrators, how to implement the CMS and so on.”* Kraft confirms this since he created the introductory documentation for Magnolia: *“I didn’t write any code of Magnolia but wrote the*

Template Quick Starter that describes how to write templates for Magnolia. It's not difficult at all, just making screen shots and describing how to create new users and so on. But it's still work and somebody has to do it." Also Bühlmann knows the high relevance of documentation to introduce developers into the OSP: *"Most important is the Plone manual describing, e.g. how to create new content types and so on."* Nevertheless, Plone is far from being completely documented compared to its programming language Python and to the application server Zope: *"Python is very well documented. There are several good books about it and it's not so difficult to learn. For Zope there are also a couple of books. Plone is the least documented of these three. There are large efforts to document Plone well, too. But it's a main problem and even a miracle that Plone has advanced that far without being well documented."*

Although a large amount of documentation is available in TYPO3 people still ask many questions. It might be due to the lack of focussing on unique documentation artefacts so people can't find the information they are looking for, Hinderink reasons: *"And you must be aware even in a project like TYPO3 that has an enormous amount of documentation, the questions are still asked. So you have to think about what you did wrong. We found out it was probably a bad idea not to fix say four main documents, each of them covering one perspective. One would be the installation manual, another one about template programming, thirdly one about administration and configuration, a fourth one about extension programming and maybe a few more. So for every important area there should be one central document. We should have done this in advance, thinking about what parts of knowledge are necessary to work with TYPO3, making one single document for each, and that's it. All the tutorials and extras can't replace a manual, it's just something different."*

Next to the introductory function of documentation the creation process of such material can also be a way to recruit new contributors and eventually new developers. To Farstad this kind of contribution offers a possibility for non-developers to participate in the OSP: *"That is a point which helps building the community because then people can contribute without even writing code."* Delacrétaz himself experienced this in Cocoon: *"Actually I got invited into the project because I started working on the documentation."* And Bühlmann remembers the initialisation phase of Plone

where it was open to anybody to work on the documentation: *“From the beginning we had a Plone website on plone.org that was very open. For example you could log in and write some docu. There were no restrictions for persons who wanted to contribute.”*

On the other hand, documentation can have the opposite effect when it acts as a recruiting filter letting pass only highly ambitious contributors possessing the appropriate skill set. So the positive aspect of a lack of documentation is to get very strong and self-motivated developers into the OSP community. Delacrétaz reflects about this when speaking of the documentation of Cocoon: *“Bad documentation is not that bad for Cocoon because we agreed Cocoon isn’t a toy for ‘Joe User’ but a tool for developers. Developers can be very productive with Cocoon but there is a lot of experience necessary because of its combination of various tools. [...] So in that sense we thought it’s better to have people fail early. People should realize soon if it’s a useful tool for them or not. In the current situation without much introductory documentation there are people who say ‘Uff that’s really nothing for me.’ Those that proceed really understand the code and for them Cocoon definitely fits. I wouldn’t say that’s ideal but it’s not that bad because Cocoon isn’t a tool for people who for example did ColdFusion only and don’t know anything else. Or a bit of PHP and HTML but without real programming experience. Cocoon isn’t for those people. From this perspective it’s good that the level is high.”*

4.2.2. Collaboration

The availability of information alone is not sufficient. Rather a well structured form of the documentation is necessary to really serve the audience according to Hinderink: *“Concerning documentation I believe you have to take it really serious as an important task. You shouldn’t think the mailing list archive is enough. The knowledge in there must be transferred into FAQs, must be rephrased and made searchable. Then you see the weaknesses of your documentation.”* Another weakness in OSPs is the common incompleteness of documentation material as Delacrétaz observed in Cocoon when he started to work on the Wiki: *“Since that time we talk a lot about documentation. Our documentation isn’t good compared to what you can do with Cocoon. We are not proud about that, it’s a real problem.”*

The general reason for this is that developers don't like writing documentation as Kraft experienced: *"Open source projects usually have bad documentation because developers don't like to write documentation but want to write cool code."* But what is the underlying reason for this? Delacrétaz gives a revealing explanation: *"Also e.g. the documentation is something where we need people. They're harder to find than people writing code. Because usually if people need code they just write it. So when I'm working on some project and need a component then I'm just going to code it. Of course I also contribute it to Cocoon and so it and also my project get better. Maybe somebody even sees my code and thinks 'Oh I can make that better.' so he improves my code. With documentation it's different. Usually if somebody writes documentation it's useless for him afterwards. He understands it now and doesn't need it any more for himself. That's why it's much more difficult to find people who invest personal time into the documentation. [...] When I write some code I can use it afterwards. But when I write a documentation I can't use it any more because I've now learned it, so that's it. So there's a difficulty."*

Having localized the cause of the problem, Delacrétaz recommends to install incentives for writing documentation to improve collaboration and compensation in the community: *"Many open source projects that have good documentation got it sponsored or the project itself is sponsored so they can invest money into the documentation. The authors then earn money for writing the documentation. I don't think many people do contribute to open source projects just for fun. People are willing to give but also want to get something in return. So here you see it very clearly. I've no problem with that at all; one just has to find a balance."* Farstad found a way for writers of documentation to present themselves and receive credit for their contributions: *"Let's take the documentation for example because that is where we get most of the contributions. If you go into the documentation and click on a page you'll have on the right hand side a list of all the users that have contributed to that page."*

Another advantage of a high reputation as writer of documentation is the possibility to become an author of literature about the OSP as Delacrétaz notes: *"What you might get from writing documentation is a good reputation as writer, which might lead to writing magazine articles. Or as the author of the project documentation it might be interesting to write a book. For the project it's definitively positive to have*

good documentation but for the writers it might be difficult to find incentives.” Indeed, Hinderink received the opportunity to publish a book about TYPO3 but also mentions the financial incentive was not the reason why he invested that much time into this work: “There were several efforts by community members and as it is typical in open source projects people also tried to make money out of it. So far two books have been published and two more will appear in the beginning of next year. And I believe there will be more in the next years because there is no publisher who doesn’t want to have a book about TYPO3. We actually get mails from publishers who offer this to us - even we as authors are asked if we didn’t want to write another book for another publisher. The one we wrote is quite successful. The feedback of the community and agencies is generally positive and they also believe it helps the project. And it’s good because this was the reason we did it - you don’t earn money with that.”

Another way to improve documentation is to call for donations and pay community members to write complete and updated documentation as Delacrétaz suggests for the Cocoon community: *“On the other hand writing a book is very tough because it remains a niche market and you can’t hope to sell 10’000 books about Cocoon today. Economically it’s difficult so we are looking for cheap ways to publish books or do professional documentation in some other form. Today’s situation isn’t satisfactory. Just a couple of weeks ago we discussed if every user could donate CHF 100 so we could employ some people to do a professional documentation. This would be great but it isn’t easy to organize. There must be fair play and so on.”* Finally there is the possibility of publishing educational materials originally produced for internal workshops etc.: *“For example I wrote a tutorial about Cocoon for a commercial workshop I held and after few months I contributed it to the project so others could also use it and improve on it. This can be a model, but it was just one case for me.”*

4.2.3. Production

Detailed descriptions of the code are required to ensure its comprehensibility as base for the continuous development of the software. Wechner is convinced of the high importance of API documentation: *“In a framework it’s very important because developers have to know how to use the components.”* So Wesdorp made sure there

is sufficient programming description available about Kupu: *“But for developers... I think you don’t have to be a core developer to work with Kupu. If you pick up a document ‘About extending Kupu’ you can have something working in a couple of hours.”* TYPO3 experienced a positive impact on community building when the core API was enriched with in-line comments of the author, Kasper Skårhøj, and further developer documents were created: *“The documentation of the core API and of the different kinds of extension possibilities of TYPO3 had several impacts. This was important for community building. [...] And there is an ‘Inside TYPO3’ document which describes how the API works. [...] Of the references the TypoScript Reference TSF is the most important one. It lists all the keywords of the scripting language. This has always been a core document from which a lot of information could be extracted.”* Nevertheless, in Kupu most documentation is still for developers so distinct knowledge about the programming language and the browser environment is necessary to be able to use the documentation as Wesdorp notices: *“Yes, it’s definitely geared towards developers so it’s not very elaborate and doesn’t explain in detail how to do each and everything. You are obviously working in a JavaScript environment which is really corks and there is quite a lot of stuff to be told to actually to get something to work. And the documentation doesn’t contain any side notes how to build something in JavaScript and so. But I think it’s decent in the sense that it’s short and compact but it explains what you need to know.”* In the end OSP leaders have to accept that there will never be enough documentation so Wesdorp is prepared to go on answering questions: *“Yes I think it would help a bit although I think would still be... I mean you can’t explain everything in your documentation so you’ll always get complaints from people that they don’t understand this and that and we should have explained that in documentation.”*

4.3. Release Management

Release management bundles and ships the software of an OSP and thus takes the responsibility to ensure the quality and functionality of a new version of the program intended for public use. In this section the actions of persons in charge of this task are discussed as well as established processes of professional release management.

4.3.1. Recruiting

Regular and frequent software releases are advantageous because they implicate high activity in the community and thus progress of the OSP. This is most easily shown by version numbering. eZ publish made use of this when it underwent major changes: *“We wanted to make a web application framework to produce general web applications and we wanted to have the CMS on top of that. That is the eZ publish 3 series you have today. [...] Actually it has been two projects: eZ publish 1 and 2 are the first generation, and then eZ publish 3 is the second generation which is much more general. It’s an application development framework for web applications.”* The positive influence of regularly advancing versions is also shown by an opposite situation. Doing new releases infrequently, especially not reaching release 1.0, can demotivate contributors as it did Rothfuss: *“One thing that made me decide not to work for Xaraya so much any more was that I believed the project had some sort of a perfectionism problem. For almost two years they’re working on a release 1.0 that hasn’t come out yet. The last mile is some sort of a death walk that never ends. [...] And projects which seldom do a release are not viewed as healthy because from the outside it seems like nothing happens.”* On the other hand, Rothfuss adds, to release too often alienates users since their installed software is always out of date: *“So if there is too little time between releases to develop new features or do bug fixing the delta will be too small. Additionally, it’s hard for people who use the software productively because they would see that their release is out of date all the time and that they might not get any support for it any more.”*

To recruit new contributors by reliable release management it is most important to implement the contributed patches into the new releases, Hinderink is convinced: *“It [the release management] is extremely important concerning quality assurance. It doesn’t work without him [the release manager]. Everybody could just publish something that doesn’t work. [...] You only realize the advantage of an open source system when all the bug fixing and other contributions of the community gets integrated.”* If contributions are not included, active users like Rothfuss will not continue to contribute to the project and will look for another solution: *“Before PHPNuke I tried another one called PHPWebsite, I think. But this never advanced even when I sent patches. In PostNuke there was much more going on.”*

An additional way to involve community members in the OSP is to introduce them to the different tasks of the release management process itself as Kraft thinks about doing in the Magnolia project: *“But it’s lots of work to do a new release. Just doing some bug fixes we can solve with patches, but e.g. including new features needs some documentation of these and description of their influence and so on. This is definitely something that the community can do in the future, too, building the installers and so on.”*

4.3.2. Collaboration

Inside a community release management can be a hot issue so in many OSPs it is solved in a democratic way among the committers of the project. Bühlmann states about Plone : *“Yet it [the release management] is a democratic process. The release manager would never say ‘Hey guys, next week the release comes out!’ There are discussions on the mailing lists until a consensus is reached.”* Other projects like TYPO3 have very clear procedures how to handle release management and how to continue when there are errors in the software: *“There are two who are responsible for the release management, Michael Stucki from Switzerland and Ingmar Schlecht from Hamburg. One cares about the Linux distribution and the other about the Windows stuff. When they receive bug reports by the bug tracker they follow a specific workflow. First they estimate the impact of the problem, whether Kasper has to take a look at it or if it’s just a minor issue. Then a solution is proposed and if it works they can commit the bug-fix themselves.”* In Cocoon, the release management process follows the prescription of the Apache Software Foundation as Delacrétaz knows: *“Yes, we have a release manager in the project. It’s his job to do the releases but it’s also documented how to do that so anyone could do this, technically. In addition, he has to sign the release digitally. A vote is required to do a release, according to the ASF rules.”* And Plone even has a specific process for including new features: *“We’ve introduced the Plone Enhancement Process. So if you want a new feature in Plone you first have to write a so-called PLEP where you describe what the feature has to do and how it can be realized. Today you’re no longer allowed to implement new features in Plone directly. After you hand in the PLEP the release management team evaluates it and then we decide in which release this PLEP is to be implemented. [...] Everyone implemented new features and no one had an overview any more. The features also had side effects which led to*

dysfunctionality of the system. That was the time when companies had problems with projects when Plone didn't work any more because of these side effects. So it was decided that no one is allowed to add a new feature without first writing a PLEP, a description of the feature. In this way you could see what side effects could possibly occur and if necessary improvements could be made in the design phase already." Other projects like Kupu don't have any rules and thus follow a very unstructured release management procedure: *"Usually it [the release management] is done by Philipp and me. It's just easy: We say one or two weeks in advance 'Please check in your changes, we do a branch and do a release of that branch. And that's the branch...' Nothing fancy. [...] So we do have sort of a stage where there is a feature freeze. We are not allowed to add any features unless it's like something breaks."*

Concerning the everlasting issue of feature freeze Wechner likes the principle of releases based on regular intervals: *"Somebody also proposed to do regular releases, for example every month. I like this because it's clear when you release. [...] In 0.0.1 steps, like having Lenya 1.3.1, the next month 1.3.2 and so on."* Also Rothfuss favours this kind of release management because useless discussions and delays can be avoided: *"Mozilla and Firefox, for example, have a very good release management. They announce the dates of their releases and tell the community that what's in there on these dates will be in the release and everything else will not. That's the way to avoid discussions about which features to include and which not. This always leads to delays."*

In general, it is common to use different development branches as many of the interviewees confirmed. For example in Lenya they use two main branches: *"At the moment we only do maintenance releases like bug fixing and so on. Sometimes we also transport new functionality from the development branch to the stable branch if it works well."*

4.3.3. Production

For Bühlmann stability of the software is very important when new features get integrated: *"It's one of the most important rules to integrate new features only if they work completely so the architecture doesn't change all the time. In a Subversion you create branches, merge them again and tag them for a release, as in CVS."*

Another important aspect is backwards compatibility of new releases. Wechner explains the responsibility of his company towards its customers: *“Here you see the difference between ‘hobby’ projects and those that are used in a commercial area. You will have some dependence on your customers and can’t just say we will change the API like that. In software engineering it happens that after a while you realize you should have done things differently. If you are a single developer you can say, even if others depend on it: ‘I don’t care, I want to do it my way!’ and restructure your entire API. But if you are in a commercial environment where people build on top of your software then you have to decide if you want that people continue to build on your software or if you want to do your own project. [...] There is still the possibility to create another branch. For example with Lenya we did this, one branch which is backwards compatible and another where we try new things and thus it isn’t backwards compatible.”* In case backwards compatibility of new releases is not possible, migration scripts are able to rewrite parts of the software to ensure their continuous functioning: *“The transition from Plone 1 to Plone 2 was a hard task. There were migration scripts that migrated most of the code. So projects using Plone 1 could migrate to Plone 2 without a change. It’s a very big help to be able to migrate old applications to newer versions of the platform. For every release a migration script is written which let’s you migrate your application to the next higher version of Plone. For example a script migrated applications from 1.0.4 to 1.0.5 and so on. There is a function that compares the version of your Plone site to the version of the current Plone installation. Then all the migration scripts are applied incrementally. It’s an important point not to let the community down with update changes in their applications.”*

Another issue which release management on the production level has to care about is the distinction between internal and external API. As Rothfuss explains this is often the cause of incompatibility when new releases are published: *“One problem in open source projects compared to closed source software is the difficulty to distinguish between external and internal API. Many projects don’t define exactly which parts of the software others shouldn’t take for granted. This makes it hard to do changes. Developers might think of certain parts as internal features and assume nobody will build upon them. But since people can see inside the code and if it’s not well defined – especially in web applications where it’s difficult to distinguish – problems will*

result.” To tackle this issue, Rothfuss advises to fix clear API policies and write in-line documentation that indicates which parts of the API are external and which are internal: “Some projects have a policy where they define how much time has to pass until something is called deprecated. When these promises are kept they also get the confidence of the users. The other step is to document with JavaDoc or PHPDoc which methods are internal or external. The external methods are under the control of the policy and the others are not. This is quite simple for APIs. With other things like naming conventions or specific files it’s more difficult to prevent conflicts.”

4.4. Collaboration Platform

The collaboration platform in an OSP is the infrastructure which allows the community to interact. It supports software development by providing a version control system, download sites of the software, issue tracking and other services. It is possible to start an OSP on one of the known collaboration platforms such as SourceForge where various services are available for free.

4.4.1. Recruiting

When entering the website of SourceForge the ten most active OSPs of the past week are listed on the left hand side. This leads to a positive marketing effect giving the OSP a high visibility. Kraft explains how the ranking can be influenced when working a lot with the different services of SourceForge: *“One big advantage is that if you do a lot via SourceForge you can rise quickly in the rating. For example the project Open Workflow where I’m still somewhat involved, gets into the top ten from time to time. John who initiated the project is currently the only active community member but he does a release and pre-release every week, writes news and so on.”*

On the other hand, Hinderink notes the existence of many other collaboration platforms and points out slowness and rigidity as main disadvantages of SourceForge: *“Well there is a lot of competition. There are also Tigris and others. The main disadvantage of SourceForge is its slowness and the lack of customization. It still covers a lot of the needs and also many people search there for a system.”* The Kupu project used to be managed over SourceForge but nowadays they use Codespeak, a hosting provider of five OSPs.⁵⁵ Wesdorp perceives the same recruiting effect as Kraft and Hinderink mentioned above: *“Codespeak helps a bit there too*

⁵⁵ See <http://codespeak.net>

because a lot of developers who are interested in new and innovative projects go to Codespeak. Although I have to add that most of them will probably be more innovative and a bit more Python developers than people who are just using a What-You-See-Is-What-You-Get editor.” For the Magnolia team branding of their software is important so this was one of the reasons why they built their own website parallel to the SourceForge project of the OSP: *“For us branding and control of the infrastructure is important so if you use SourceForge you’ll always do some SourceForge marketing. [...] As I said branding is also important so people receive a nice looking website and not the one of SourceForge.”*

4.4.2. Collaboration

SourceForge used to be the CVS server of Plone as Bühlmann remembers: *“In the beginning, yes, but now they have a Subversion repository. They try to install their own infrastructure for versioning, project management and so on. Nevertheless lots of things are going via SourceForge.”* As Bühlmann believes SourceForge is a perfect possibility to start with a new OSP. Later on, as Rothfuss observes, many large OSPs move away from SourceForge: *“Most projects that are successful and finish their childhood will leave SourceForge after a while. It also happened to PostNuke because the infrastructure didn’t bear the demand.”*

It is relatively easy to start a new project on SourceForge. But Rothfuss sees handicaps of this low entrance barrier as well: *“Their advantage is that they make it easy to create a new project – but this is also a disadvantage because many immature projects will get listed. It would be cool if they had more structure. There are many projects that have a release called e.g. 0.0.0.1alpha23-c1... So somebody should filter and look if the projects are really good for something or not.”* In the eZ publish project the limitations of SourceForge were the reasons why they moved to an infrastructure of their own. Farstad also points out the restriction at SourceForge not to sell services on their website: *“We host everything ourselves. We need this because of the freedom. Once you grow up over a certain level you just need this freedom to control everything the way you want. [...] There are limitations, especially if you have lots of traffic and many downloads and so on. Still it is good as supplement or an addition. For us, for example SourceForge just wouldn’t work. Especially not since we’re also a professional company and sell services. This is not allowed*

there.” Similarly, Kraft built his own development environment for the Magnolia project and uses SourceForge only as download site of the software: “If you build your own infrastructure you’ll have much more freedom using your own preferred tools, like bug tracking and Wiki. Today we use SourceForge as download site just to save bandwidth. [...] We didn’t put our mailing list on SourceForge because we don’t want to be dependent and also we knew that e.g. the CVS had problems again and again.”

Rothfuss in general favours newer technologies than those offered by SourceForge so configuration of an OSP-specific collaboration platform would be the logical consequence. But he cautions not to underestimate the effort of building one’s own infrastructure: *“There are people who like doing such infrastructure things, but many don’t. Especially in the beginning it’s practical to go to such a platform. Often it’s underestimated what’s actually necessary to build a good infrastructure. It might mean a severe distraction from the actual work. I rather dissuade against it. Even if I’ve criticised SourceForge I didn’t want to say that they don’t offer a great service. I believe such an environment is exactly the right thing for certain stages of a project. And not all projects ever outgrow SourceForge’s capacity.”*

4.4.3. Production

To actually work on the project the collaboration platform needs to be well organized so one possibility is to split up the source code of the OSP into different sub-projects as Hinderink reports from TYPO3: *“In the second [SourceForge] project some parts have been outsourced, e.g. the Database Abstraction Layer, DBAL and other things. These are extensions with a large impact on the system, having large consequences on it. And there is a maintainer for each of these areas.”* The CVS can be considered a handicap of SourceForge since today’s technologies with Subversion offer better possibilities to track the changes of the software as Rothfuss implies: *“Also I’d say SourceForge isn’t that innovative any more. Their bug tracker is quite old and so far they offer CVS only. Hopefully they’ll offer better tools soon.”* Wesdorp appreciates the benefits of the Codespeak platform: *“It has kind of a nice setup there that he has this SVN [Subversion] repository and some other stuff that is shared by all the users.”* On the other hand it has its advantages to build one’s own infrastructure to

use better tools, as was done in the Xaraya project: *“Bitkeeper was introduced, a version control system that’s also used for the development of the Linux kernel, similar to CVS but with much better features.”*

4.5. Physical Meetings

As opposed to the usual virtual way of working in OSPs, physical meetings bring a new possibility of interaction into the community increasing the communication bandwidth enormously for a short time. There are many different kinds of such meetings ranging from fun events to user group meetings to conferences, open source exhibitions and Sprints.

4.5.1. Recruiting

The first public presentation of Plone happened on a conference Bühlmann attended: *“The two initiators of Plone met for the first time in Charleroi, Belgium, at the Euro Python Conference 2002 where I also went. In a small room next to the conference they presented Plone during 45 minutes to about 20 persons. They presented the basic idea and the architecture behind it. Somehow, I don’t know why, lots of people got attracted by the project.”* Later on, Bühlmann had the idea to organize a Plone-specific Sprint in Bern: *“I announced the Sprint and hardly a few days later all the 30 places were occupied. So in February 2003 we organized the first Plone Sprint of the world here in Bern. I knew Plone needed something like this; people had to get to know each other to improve working together via the Internet. So we really went drinking some beers. This was the beginning of a whole lot of developer meetings.”* Community promotion can also be organized by companies, as eZ systems is doing by offering lectures by the software developers: *“One thing we’re working on now is to get user groups started. Currently we’re starting a user group here in Oslo. We are promoting users who want to start an eZ publish user group. Also we offer to organize third party talks so for example if they want somebody from eZ systems to talk at their user group meeting we can probably arrange that. We can also help out in promotion and hosting of their user group website. There are some user groups on the way.”* So physical meetings are also attractive because core developers are present often giving valuable assistance on coding issues as Bühlmann experienced personally: *“During a Sprint you learn much more than by reading mailing lists. You can listen to talks and gain much more than*

through reading mailing lists. You can also program together with developers who really know it. In this way you learn much faster. It's the pair-programming idea, writing code in pairs."

Community meetings don't need to be working events. The annual snowboard tour of the TYPO3 community e.g. is mostly a social meeting point for the contributors. It also attracts newcomers to the community: *"The annual snowboard tour is another psychological factor binding the community together, because it's not just a conference but a fun event."* The fun part is important and meetings must be attractive for visitors so the Plone community looked explicitly for extraordinary places for their sprints: *"In the castle we worked in the living room of the owner. The dimensions of the room were about 10x10x8 meters and dinner we had at a long table like 50 knights. Those were very special, unique moments when programmers from all over the world met for one topic only."* The personal relationships among the software developers are important so Bühlmann continues: *"Some of them I'd already seen for the third time that year so I can say now that I know Plone people better than my neighbours here in Switzerland. So the Plone community is not an IRC or email community but a community of friends who see each other regularly. This leads to unique stories every time we meet, like if you go into a training camp with a sports club. For example at the first sprint I organized a traditional Swiss cheese fondue in the city. Limi ate way too much so he won't ever again eat fondue. For two days he suffered from stomach aches. To this day he always reminds me of this incident and that it was a bad idea to go out for fondue. I even acquired the nickname 'Mr. Fondue.' This is the kind of stories you tell over and over again."*

General exhibitions of open source software are also a possibility to present the OSP to interested visitors as Hinderink experienced: *"There are always these show case events like LinuxTag or LOTS in Switzerland. These things are always very valuable."* And Farstad adds: *"Also we are active at the various open source conferences and go to trade shows. Basically we travel the world to meet people."* He points out the positive effect meeting the developers personally at these international conferences: *"Yes, there is one official eZ publish conference in Norway. And we are at many other conferences many times a year like the PHP conference, the OSCOM and others. So we meet a lot of people all the time. Now that we're starting*

an office in Germany we will most likely start a German conference as well. [...] That really makes a difference when you discuss something in a personal forum. I mean I did this for several years and suddenly you meet them in person - that is totally different to discuss after you've met them. Of course you don't need to do that but it really helps." Plone community members, too, went to a number of exhibitions: *"In Italy we simply met in a booth during an IT exhibition. Instead of exhibits we set up some tables where we could work. It wasn't really good because it was too noisy but the important thing was to meet once again after only a few months. [...] You talk about this and people are amazed that developers from all over the world meet there to program during three days."*

4.5.2. Collaboration

To enhance internal communication and collaboration, the most active community members usually go to the conferences as Farstad knows: *"Also we have a conference every year so I actually meet people in real life. That's where the community is, at least the hard core."* Hinderink, too, saw the demand of the community for more knowledge about the project and thus the community plans the first international TYPO3 conference: *"Also there will be the first conference next year [2005]. We don't know yet when, either May or September. It's in Karlsruhe, the first international TYPO3 conference! [...] We hope to relieve the snowboard tour. The need for a conference grew a lot because of the success of the project. So many people come to the snowboard tour hoping to find a conference which is not possible to offer."* Next to the large events there are also small-sized meetings for the core developers of TYPO3 to better coordinate software development: *"First there are small events for the people around Kasper, about once a year. This year we were only six persons when we met in Lüneburg. Last year there was a larger meeting because of a specific task. It's not a public meeting, people get invited."*

Because of the increased confidence in the community large sub projects can be initialized during or as a consequence of physical meetings: *"The greatest effect, as Kaspers says, is 'adding faces to emails.' A new level of trust arises between people who previously met only on the mailing list. After these snowboard tours new projects get started and collaboration as a whole is better. For example on the last snowboard tour there was the discussion about translating the typo3.com website*

into German. The issue showed up again in the marketing mailing list so an agency in Berlin called about 20 other agencies and collected 100 euros from each of them. Thus they could pay a translator and the site got translated very quickly. There were long discussions before but nothing had changed. There is a difference if you met once and drank a beer together. It's just easier to appraise somebody." In the Cocoon project, too, collaboration improved when during one of the conferences the community came to an architectural decision as Delacrétaz remembers: *"For three years now we have the Get Togethers, annual community meetings. Last year for example we decided to have one single form library, the classes which process the forms. Before there were three libraries but we decided to really focus to get a clear, common vision. We don't want to exclude the others but for the Cocoon project we have now Cocoon Forms. There are still people who do large applications with other libraries, it's not prohibited, but for the team there is now only one."* And in the Kupu project a personal meeting between the future core developers brought them to an important decision about the software license: *"And it wasn't until we went to Switzerland for some reason. I think there was a Sprint or some discussion about editors on the client side that can help open source content management systems. That's when I got to discuss with Philipp von Weitershausen by the way. We had the Epöz issue at that point. That's when we decided to change the license to a better known open source license. So it's now BSD style rather than the Zope Public License it used to be."*

Another advantage of physical meetings is the knowledge transfer as Bühlmann experienced during extreme programming with Alan Runyan: *"There I programmed together with Alan Runyan in a pair programming team. In this way I got to know him and learned a lot from him."* Also, personal relationships can overcome internal conflicts as in the case of the difficult phase transition from Plone 1 to Plone 2 when somebody wrote an email with an urgent request for a professional release management. Bühlmann remembers: *"First I thought 'Oops, now we've got trouble brewing!' But since the people knew each other personally no bad mood developed. So here we see the importance of personally knowing each other. This I believe is the main success factor of Plone: people drink beer together."*

Although Wechner, too, is convinced of the positive effect of physical meetings among developers he points out a certain danger: the formation of an inner circle: *“They’re good to advance the project and let it run. They’re motivating and giving an impulse for the community. On the other hand there is also a threat. When you meet all those people, spend time together and go out and so, then some sort of an inner circle develops. So you have to be aware that such an inner circle doesn’t start making decisions and fencing off the outside. [...] For example your boss likes to go out and drink so you join him and slowly you become part of this inner circle. Another guy who is really great, might not like to drink that much so all of a sudden he is an outsider. It’s always the same thing. In open source communities, too, there are people who are more respected than others. Like in Animal Farm: ‘All animals are equal but some are more equal.’ So it’s important that those people who have the power in the community take care that this doesn’t happen. This is another aspect of that strong leadership. [...] As everywhere connections help. [...] At least, in a community the effect of such relations is much smaller than in a company. It’s a vicious circle where only those who are stronger and have connections rise up and not those who are actually better. Maybe they’re just not good in developing these relations.”*

4.5.3. Production

At the annual meeting of the Cocoon community above all the software itself gets further developed: *“There is an annual Get Together. This year [2004] we had it for the third time. This is very good. This year we had 130 participants, quite a lot for one single project. The first day was Hackathon, bug fixes and so on, the other day was conference and in the evening we had some social event.”*

Apart from all the social effects of Sprints, Bühlmann affirms the main idea behind them: *“A sprint is a meeting for developers where intense product development takes place.”* To make sure this is really accomplished, Rothfuss recommends to set clear and realistic goals for a Sprint: *“The communication bandwidth is extremely high and it’s excellent for those who can participate. Often things can be achieved during a Sprint which normally would need months to accomplish. We already had a couple of Sprints. There are actually two different opinions. Some say they just want to meet and look what to do. The other direction is to prepare it in more detail and to have*

somebody leading the Sprint. Usually if there is just a meeting, the people end up reading their email in a room. For a Sprint to be successful, specific goals are needed and not just some abstract ideas like a new architecture. The goal should be realistically achievable during the Sprint.” To successfully carry out a Sprint the necessary infrastructure must be available as during the first Plone Sprint at the University of Bern: *“It was a perfect place to do such a Sprint because we could install WLAN and had black boards to sketch ideas quickly.”*

A distinction between the various types of OSP is made by Wesdorp who observes a preference of large projects to use Sprints as their instrument for radical progress in software development: *“As it appears, right now at least for Kupu 1.x, we will never have a Sprint I think. I mean if people are interested, I’m interested too. I think development, as it goes right now, goes quite fast. Not like these huge, monolithic projects that just need a Sprint to get the first head start or so. [...] The sprints I’ve seen so far are usually for larger projects. When I think about a Plone Sprint for instance it’s just to get stuff done for a project that is already long running. It’s not by definition for a monolithic start-up or something. I was maybe a bit overdoing it. [...] I’m not sure but maybe a difference is that Plone is way larger so you can actually help with large parties to do a lot of hacking and have a good time and also make it sort of coordinated because I reckon in a project as large as Plone it’s way harder to coordinate the development team. It’s definitively good to know with whom you are working.”*

4.6. Foundation

Most of today’s large OSPs are owned by a foundation or an association. These act as the juristic entity of the OSP holding the copyright of the program code thus representing the software company in commercial environments. But in the open source environment the foundation also fulfils other tasks as shown in the following chapters.

4.6.1. Recruiting

After being accepted by a well-known open source association like the Apache Software Foundation (ASF) Wechner knows the publicity of the OSP will increase: *“At that time the Apache Foundation had decided they wanted more top level projects - like the Apache Web Server or the Java project. [...] Stefano wanted to make Cocoon*

a top level project and have the CMS and other Cocoon based projects as sub projects. He liked our software, because we had integrated other stuff, so he asked us to donate it to the Apache Software Foundation. For us this made sense because it was open source anyway and also because of visibility reason.” The good reputation of the ASF gives the foundation a high credibility especially concerning its license as Delacrétaz experienced: *“The reputation is certainly good. People know that the Apache Foundation is here to stay, it is well known for many years now. Also our license is ‘clean’ so people can do anything with the software and they will not be legally harmed as long they retain the credits and don’t misuse the Apache label. For example they can’t call their company ‘Apache Cocoon whatever.’ Also the logos are protected and such things. But for everything else one has lots of freedom to apply the Apache License without being sued. So the reputation is definitely important.”* Wechner goes even further and calls the ASF a strong software brand transferring its label to all projects: *“This has a large influence because today Apache is a real brand. Nowadays you don’t have to explain to an IT manager why it’s good or bad to use the Apache Web Server because over 50% of all the websites are hosted with the Apache Web Server. This confidence spills over to the other projects. Of course it’s not sure that you can sell your project just because of this but at least it’s a door opener, you don’t have to do much selling any more.”* Still, Wechner has noticed no radical changes when Lenya joined the ASF: *“The community didn’t grow enormously. We already had about 150 people on the mailing list when we joined the Apache Foundation. How many we are now, I don’t know. Nothing changed radically, maybe because in the beginning we were an incubation project, one of the first ones.”* On the other hand, Kraft is somewhat sceptical if the influence of the ASF is really that high for all of its projects: *“It’s difficult to say. Apache does a lot of software but not all are well known projects.”*

The Plone Foundation plans to register a trade mark enabling professional marketing support: *“Another aspect is that Plone gets registered now as an international trade mark so nobody can misuse Plone for his own purposes. [...] The foundation is also developing a marketing concept. Porter Novelli, a world wide marketing company, donates marketing services of a value of USD 20’000 a month by letting one of their marketing specialists work for the Plone project.”*

The newly founded Magnolia Organisation intends to attract new members by offering them the same influence as Magnolia's initiating company obinary: *"We recently founded the Magnolia Organisation of which obinary shall be only one member among many others. Obinary will always be the initiator of the project but other companies shall have the same rights, too."*

4.6.2. Collaboration

The community members of Xaraya founded an organisation based on the ASF as Rothfuss describes: *"We then designed our structures based on those of the Apache Software Foundation because this is a mature organisation that found out what worked and what not. So we founded the Digital Development Foundation that keeps the copyrights of Xaraya and should be the long term supporting institution of the project."* The overall goal of a foundation is to bring sustainability to an OSP: *"In general a foundation just gives more stability to a project. People come and go, but the institution stays. So for example at the moment I don't have much interest in Xaraya anymore but the association still stays the same and people can go on working. By these means, a foundation reduces the risk for all participants."*

The TYPO3 Association was founded to distribute the various tasks of the community clearly and to relieve Kasper of work: *"It needs to be distributed among different people. One major effort to do so is the recent foundation of the TYPO3 Association with residence in Switzerland. The idea of this association is to distribute different tasks like quality assurance of the extensions and the service providers."* The basic goals of the new institution are transparency and benefits for the members as Hinderink explains: *"From my point of view there was also some Glasnost idea behind it because transparency and openness are also issues. The association is supposed to have a beneficiary effect for everybody. It won't go into competition with the agencies and offering service to customers - only for community members."* Besides specific benefits and duties an association also serves to develop a certain kind of common culture. Delacrétaç describes how this is achieved in the ASF: *"So we try to find people who make their way into the community gradually, learn how to deal with others, learn how to handle conflicts - of human or technical nature. We call this 'the Apache way.' There are written rules and there are also some sort of politeness and methods that aren't written down. That's how we work."*

Furthermore, it is possible for an association to hire people for certain tasks where no volunteers can be found: *“It’s going to change now somewhat because of the founding of the TYPO3 Association. By defining responsibilities it gets clearer. Also project groups with certain tasks will be formed. Nevertheless, just by formalizing things it’s not guaranteed that everything will work. The main problem remains: people have to invest time. So one of the ideas of the association is to pay for certain work where no volunteers can be found.”* A similar idea is behind the Plone foundation: *“Right now there are important activities going on with the initiation of the Plone Foundation. It’s like the Apache Software Foundation, an association that protects Plone and offers administrative help. Computer Associates have sponsored Plone and donated USD 100’000 to the Plone project enabling the hiring of somebody to lead the project. He has to see that everything works fine in the Plone project. He mainly coordinates the mailing list, motivates people to do certain tasks and so on.”* Concerning the hiring of people the ASF follows another strategy as Delacrétaz explains: *“When more and more projects are included one has to take care to retain a clear overview and an appropriate support of the infrastructure. Maybe it would become necessary to hire people to do that. [...] The Apache Foundation has no employees at all. The ApacheCon, the Apache conference, is organized by a company that gets a part of the money that is generated by the conference. Voluntary work wouldn’t be appropriate here since it’s an international conference. That’s the model for the moment. Maybe if the foundation continues to grow we might need part-time employees like a secretary or something like that some years from now.”* Rothfuss points out the advantage of an association concerning donations: *“If somebody wants to sponsor the project it’s much easier to donate money to the foundation but to one single person. Also it’s better because you can deduct your donations from your taxable income.”*

Another reason to form an independent association for an OSP is to hand over all legal issues to this institution thus protecting the individual developer legally. Rothfuss describes this responsibility of the association by pointing out the cumulated power of the foundation: *“First of all they [the foundations] protect the developers against lawsuits. The basic idea is if there are any legal issues not one single developer without resources has to protect himself but since the copyrights of the developers are transferred to the foundation it takes care of all these things.”* Bühlmann con-

firms this function and explains one of the goals of the Plone Foundation: *“The third action is to establish a well based licensing model. It’s a hot topic right now. They try to assign all the copyrights of Plone to the Plone Foundation. At the moment Plone is published under the GNU GPL. The Foundation is supposed to decide under what license Plone is to be published so they want to clarify the current state of right. The idea is to relieve the individual developer from responsibilities. So if a patent claim is opened, e.g. Microsoft says Plone infringed upon our patent XY, then the foundation behind Plone shall make it less likely that a company sues Plone. Before this change, e.g. Alan Runyan or Alexander Limi could have been sued because they might have broken a registered patent. But because they gave their rights to the Plone Foundation a company has to sue the foundation. They, however, can take an attorney much easier now and this might scare off potential adversaries. This instrument of power is supposed to provide security in the background for us as a company, too.”* The Apache Software Foundation, too, protects its committers with its license: *“Also the committers are very well protected by the license. Nobody can sue a committer. If anything judicial happens everything is forwarded to the foundation. We are very well protected. If for example I made an error in my code and some user of my Apache Cocoon software has problems because of that he can’t sue me since it’s all managed by the foundation.”*

As shown in the above statements any kind of association brings advantages to the OSP community. Sometimes, however, it’s difficult to find an arrangement, which all stakeholders of an OSP agree upon. Bühlmann experienced this when founding the Zope association in Switzerland: *“We founded SwissZope, an association of companies working with Zope. Unfortunately it didn’t come out so well so it’s a bad example of an open source community. In Switzerland there are now two Zope communities, but the country is actually too small for two communities. In the beginning there were people from Zürich who founded zope.ch. I wanted to help there but also intended to introduce business aspects like company presentations, a Plone service supplier directory and so on. But the people who founded zope.ch wanted to build a non-commercial community. We tried to negotiate but failed so we founded swisszope.ch. We also went to the Internet Expo with this label which*

resulted in new projects. People read some showcases on swisszope.ch and followed the link to our company. So it's not really a community now, only three companies are still on the portal."

4.6.3. Production

As explicitly illustrated in the section above an association cares intensively about legal issues concerning an OSP. While for collaboration it is important to protect the contributors, for production it is vital to protect the source code of the OSP. So when the copyright goes over to the foundation it becomes also the owner of the code and thus can be protected by the association.

Additionally, the progress of the OSP itself can be supported by funding core developers for their work: *"The main idea is to raise money to pay the core developers of TYPO3 to continue their work. So to contribute something you just become a member of the association and pay your annual fee. Membership is possible for individuals and companies."* The community should support Kasper in further developing TYPO3 following his hopes for the future: *"He wishes money and autonomy to concentrate on development. But Kasper also wants to keep his way of working, especially this intuitive and creative part. He's not an engineer who first makes exact plans and then builds everything according to them. He's really creative, thinks about an issue and then gets into some flush of work. This is something that has to be protected. I look at it as a challenge for the community to preserve an environment where TYPO3 keeps the continuity of development and also uses and protects the talents of Kasper. These are the things that have to be synchronized - and it's not very easy. The community has to offer a nutritive soil for this."*

Last but not least the technical advantage of the ASF and other OSP associations is the high end infrastructure they provide for their projects. As an example Cocoon uses it for development and distribution: *"The Apache website has very good mirrors because it's also used to distribute the Apache Web Server which has much more downloads than we do. But we can use the same infrastructure. There are mirrors in Switzerland and elsewhere. [...] And there's also the infrastructure. There is a very good infrastructure since it gets paid by sponsors, companies and individuals. We also benefit from this. So it's certainly good to be part of the Apache Foundation."*

4.7. Internationalisation

There are various aspects of internationalisation of an OSP. Basically it means the software isn't limited to one country only. Thus community members, both active and inactive users, live in different parts of the world using the software adapted to the individual conditions of their environment.

4.7.1. Recruiting

When going international, all the active developers in the global market of open source software become potential contributors of the OSP as Rothfuss concludes: *"If the project is used internationally the market also increases."* Bühlmann holds the same opinion and advises even to think about internationalisation from the beginning: *"I believe Switzerland is too small to start an open source project focussed on the Swiss market, only. There are too few people, developers and users. Switzerland is too small a country so you need to start your project internationally from scrap."* Interestingly the Plone project itself was started by two developers living in different continents: *"Alexander Limi is from Norway and Alan Runyan is from Houston, Texas. They physically met for the first time in Belgium."* Another way to recruit new contributors is to offer authoring of translations of the user interface. Rothfuss recommends to keep this in mind from the beginning: *"It's quite easy to accomplish and contributing translations is also a good opportunity for people to get into the project. If you plan this from the beginning it's a serious advantage."* So it is successfully done at the eZ publish project: *"And our system is also multilingual so you can do translations. [...] We've gotten about 20 translations, all of which are contributed by the community except the Norwegian and the English one."* Organizing international meetings at differing sites allures community members because they have a good reason to travel around and meet people from all over the world. Bühlmann mentions the various meetings of the Plone community: *"In December 2002 I went to Rotterdam to a Zope Sprint. [...] In May 2003 the next Sprint took place in Padova, Italy, and later in September 2003 the next one in a castle in Austria."*

An important factor of internationalisation is the language for communications. Farstad shows there are various considerations about the issue concentrating on the question of using one single language or offering different communication languages *"First you need to speak a common language, normally that's English. [...] So by*

having an English community some people won't join because they don't speak the language. [...] To solve the problem, you could create an English, a German, a French forum and so on. But this means some people will just discuss in the German forum so you'll lose those users for the international forum. It's a two headed horse. We haven't yet opened any language-specific forum other than in English. So all communication in our community is in English."

4.7.2. Collaboration

By knowing community members all over the world new possibilities of resource allocations become possible. Bühlmann tells about a Sprint meeting: *"There were people from Japan, India and so on. So one guy from Bangalore asked if we still needed any development resources because he would suggest renting Plone programmers from Bangalore."* Also there is a large motivational aspect of having an international community. Rothfuss e.g. is always amazed of the contributions that are submitted during 24 hours a day: *"In addition it's really great to wake up in the morning and receive a bunch of commit messages because people on the other side of the planet continued to develop the project."* Of course this is not the only reason because the more people from all over the world use the software the better its applicability becomes, as he continues: *"It also broadens the project and makes it useful for more than one application."* And not only the software becomes broader but also the community benefits from its increasing variety. Farstad mentions it is even possible to overcome inhibiting customs of one cultural area: *"You'll definitively get the feedback on use cases and set-ups which is not normal for our customers in Norway. [...] So you get feedback from many different cultures."*

On the other hand having an international OSP community also brings up new collaborative challenges. One issue is communication which gets more difficult when people don't write in their native tongue. Therefore, Delacrétaz recommends empathy and respect for each other: *"On mailing lists you can communicate only through writing and it's asynchronous. For many people English is not their native language and also the mentalities are different. We have people from Europe, America, Australia, South America, Russia, people from everywhere. So you have to take care how you say things. So when I think that somebody wrote something weird in his email I have to comment cautiously e.g. '*I believe* that's wrong.' or 'I'm a*

bit worried about that.’ Not ‘You are dumb and what you’ve written is stupid.’ So often we realize we misunderstood it or he didn’t write what he meant because of his English. It has to do a lot with open communication, politeness - not meaning political correctness, just common politeness and respect for people.” Also Rothfuss knows about these communication challenges: *“Also you have to remember that for many people English is not their primary language so if somebody writes something it might be misunderstood on the other side of the planet.”* So it is not astonishing some people might be embarrassed to participate in the ongoing discussions again because of cultural habits as Rothfuss continues: *“Often it’s underestimated how many people don’t know English very well or feel too embarrassed to participate in the mailing list. Also in certain parts of the world it’s not common to participate in a discussion, for example in Asia.”* And for physical meetings one has to remember the different cost of living. When Bühlmann organized the Plone Sprint in Switzerland the visitors from other countries encountered a much higher price level here: *“It was good but Switzerland is too expensive to organize international Sprints because of the side costs.”* Additionally, licensing problems may arise because of varying legal systems in different countries as Hinderink warns: *“These are points where you leave the developers’ corner and get into legal issues. That’s difficult because there are also different geographical areas with different juridical environments. For example in Germany the regulatory frameworks are simply different from those in the USA or other countries.”*

4.7.3. Production

As Farstad knows not all software is developed in way that translations of the user interface are easy to manage: *“The software has to be general enough to be able to be translated and localized into other languages like Chinese and Arabic. This is specifically important for CMS. So the design of the software is critical if you want to have an international community. It doesn’t help if they can discuss the software on the forums if they can’t use it in their native language.”* Also Rothfuss assumes translations will be demanded of good software: *“If a software is easy to install sooner or later people on the mailing list will demand the translation into different languages.”* Wesdorp experienced this in the Kupu community so they are planning

on extending the editor with customizable language files: *“We have a lot of requests for internationalisation. I currently wrote a JavaScript library for internationalisation so we can use that.”*

4.8. Recruiting Specific Subjects

In contrast to the spanning subjects the recruiting specific ones only concern improvements of methods to enlarge the community of the project. Those are basically marketing aspects that increase the attractiveness of the OSP so skilled developers and other contributors are motivated to join the community.

4.8.1. Spreading the Word

News channels publish general information about new software and technologies, have a widespread and established distribution and are thus interesting for OSPs to be used for press releases and articles about the project. TYPO3 uses a wide variety of such channels: *“TYPO3 is quite well represented in the press. Then it’s present in all kinds of directories like SourceForge, Hotscripts, CMS Matrix of OSCOM and so on. These play important roles. [...] Also it’s great if you get onto one of the magazine CD-ROMs of c’t or something like that. It’s always good to do press related work. I don’t know how important such directories are to start. I believe it usually goes via magazines and events.”* Also Farstad knows the power of press related work so they write professional articles about eZ publish themselves and send them to publishers: *“The first step is to increase the number of people who know about eZ publish. So we write articles and have them published in different forums and magazines.”* Kraft follows the same strategy and even could get an article on Heise: ⁵⁶ *“Our strategy is to do marketing and spread the word about Magnolia. For example last Wednesday we were on Heise and this had a lot of impact. Also newsletters are important like the one for Apple Developers where we can announce a new release, and websites like Serversite and JavaLobby.org. Developers’ magazines would be very cool but it’s quite hard to get to the right people.”*

Originally, eZ publish became known on open source websites. Farstad advises to use as specific platforms as possible: *“We also had a page on SourceForge but that hasn’t been maintained for many years. It started out on Freshmeat and some other*

⁵⁶ <http://www.heise.de>

sites. One of the first sites we also published it was Hotscripts. If you go there and look at the rating system... There are about 20'000 projects on Hotscripts. For about the last three or four years eZ publish has been the most popular project, the most viewed, the most downloaded and the most clicked on. That has been one of the channels where we became known. I think they were Freshmeat, Hotscripts and some other smaller site. [...] They help with marketing and make announcements about new releases. [...] The whole point of those channels is to get the word out. [...] Freshmeat is not very specific, the users there are not looking specifically for web applications, especially not CMS. If you have more targeted sites like for example Hotscripts then you reach the audience in a better way.” The Kupu team on the other hand started out on the open source website zope.org only to publish their OSP: “We didn’t go to OSCOM at first. The original Epox project was run on zope.org because it was a Zope-only thing. So we decided to do the first releases on zope.org, too.” Later on they moved to OSCOM, the international association for Open Source Content Management, experiencing the positive influence of this step: “It’s a cross pollination thing. We haven’t been too active on that point but I do know that a lot of OSCOM people at least know about Kupu, want to integrate it into their system or already have it done. There is a bit more communications going on. We have a couple of shared mailing lists, not really active though. It’s a bit useful at least, I think we still could improve it though. I would like it to be a bit more active.” Kupu as well uses many different channels to spread the word about the project and to be found easily: “I think a lot of people find us by just googling and reading new pages. [...] Of course OSCOM helps there and I think Codespeak helps a bit there too because a lot of developers who are interested in new and innovative projects go to Codespeak. Although I have to add that most of them will probably be more innovative and a bit more Python developers than people who are just using a What-You-See-Is-What-You-Get editor. Zope.org helps also, we still post some news items to zope.org. And of course also to news channels and Freshmeat. We do only post items on Freshmeat so we sort of use it as an announcement channel.”

Last but not least there are often companies behind OSPs or at least offering services for them so they are also interested in doing marketing for their project. For Kupu there is Infrae employing core developers of the project: “Also we try to do some marketing from the company Infrae I work for.” And so does Bühlmann in his com-

pany: *“The intention of 4teamwork always was to spread the word about Plone throughout the nation so we did lots of marketing.”* Hinderink even believes the major part of marketing is done by the service providers: *“But the major part is probably done by the agencies doing marketing on behalf of their customers. I think that’s the marketing channel number one.”* It’s self-evident for companies like eZ systems or obinary who initiated their OSP to do intense marketing for their product and offer a wide variety of marketing material as Kraft explains: *“[We have] lots of marketing stuff like a features list and even a video walk through the process of how to create a new page.”*

4.8.2. Credit System

Delacrétaz describes the importance of acknowledgements in an OSP noting business reasons: *“It’s important to recognize people for their work. And for someone like me as a self-employed person it also has business value. This is my reference. People contribute so we have to give them something back. We can only give acknowledgement, neither money nor rights because with the Apache code everyone can do what he wants. Of course there’s also the community and that’s fun, but the other side is very important.”* In this way an OSP can also be attractive for a student doing e.g. a research project to write tests or do other contributions because Delacrétaz knows they can make themselves known like that: *“In this way these students could make a name for themselves, as contributor or even as committer. If somebody writes a lot of code it will be noted somewhere - you get credits.”* Contributions are credited in different ways depending if they are minor or major ones: *“We have a status file where all the changes are logged. So when I get a patch from somebody I as committer integrate it into Cocoon. Then I’d enter into the status file that I implemented the patch, which was created by this other person. That’s for minor contributions. People who contribute larger parts are mentioned in the credits file. There you’ll find, ‘part X is donated by company Y’ or ‘developer A wrote code B.’ We take care to give people their credits.”* In other projects like Kupu the credit system is not so minutely maintained but it’s still present: *“We have a credits text file in the package. I don’t think it mentions all the developers. Maybe I should correct that. It has been a while since I last did changes in that.”* To appreciate the contributions, on the website of eZ publish the names of the authors of translations are listed: *“For example the translations page is a very visible page. You see a map of the whole*

world and a list of all the translations. And of course the names of the persons are listed so they get credit for what they have done including a link to their web page. I think this is important. As a contributor you feel like my contribution is appreciated.”

Rothfuss points out the risk connected with writing names into the source code so he states about explicit credit systems: *“There are different opinions. Some believe they need this absolutely. But Apache for example thinks it wouldn’t fit their goal ‘The community is more important than individuals.’ So they have a credits file but it’s not said who did which part. This practice is meant to prevent, say, a company from accusing specific programmers for their code. I wouldn’t list credits inside the code. Another, more marketing way is to do weekly interviews with the developers and show them on the project website. KDE is such a project. Generally, promotion shouldn’t take place in the code. It might even hold off people contributing when they see that 95% of the code comes from one single person.”*

4.9. Collaboration Specific Subjects

The following topics pertain only to the possibilities of improving collaboration in OSPs. The ways of communication are essential in such projects so the different aspects of each communication channel are pointed out. Also central aspects of a community’s structure are discussed with a special emphasis on the influence of task lists on community building and improvement.

4.9.1. Communication Channels

For virtual collaboration as it is used in OSP communities very elaborate communication means are vital. With the Internet, many different ways of communication are possible so the interviewees were asked what specific channels they use and what their experience is with the advantages and disadvantages of each of them. To give an overview, Wechner outlines the interaction of today’s most commonly used communication channels in OSPs: *“So in summary I’d say IRC is for very quick and simple questions and answers, the mailing list is to discuss issues in detail with the entire community and the Blogs are in the end the consolidation of all the discussions and decisions, which are published. And the aggregated Blogs in the Planet XY show the different opinions of the developers.”* Additionally, all of the

OSPs investigated maintain a project website with all the permanent information like documentation, frequently asked questions, feature overviews, demos etc. Because this channel is assumed well-known and for a lack of specific statements it will not be treated further in this paper.

Comparing impermanent and permanent communication channels, Rothfuss shows the roles of the different types of technologies: *“The mailing list is the working horse. It’s quite old, widely used and practical if the traffic is not too high. You have an archive including a search option and everyone can read it even if he wasn’t online. On the other hand it’s quite annoying to see that although you reached a consensus and maybe even a decision, after a while people forget about it and the same discussion starts all over again.”* Mentioning this disadvantage of insufficiently structured knowledge in mailing lists he adds: *“Therefore tools like Wikis and bug trackers help to do write down something permanently. Additionally real time communication like IRC is also important. Not to allow some kind of conspiracy or to do design decisions but to discuss with others and just hang around with them. I believe this is important for community building.”*

4.9.1.1. IRC

Most of the investigated OSPs use IRC actively and for some of them like Plone the chat is even fundamental: *“IRC is important. It is actually the backbone of Plone. [...] It’s more like a meeting point where you get inside informations and where rumors are produced. [...] There are many channels but only one Plone chat you should use. There are companies who log the chat and read it afterwards because there you learn the most. Although information can get lost the advantage of IRC is it’s direct and fast and you can talk to each other as in a restaurant. On plone.org you can find a couple of IRC protocols of interesting chat sessions, just copy-pasted to the website.”* In the Kupu project, too, many community members meet online regularly: *“Our IRC channel is always quite active and a lot of people pop by at IRC before visiting the mailing list. It’s just easy to get an instant response, it’s an easier way to communicate. And then they choose following the mailing list or the IRC channel, too.”* And in Cocoon there are even regular online meetings of the developers to do focussed communication: *“There is an IRC channel for Cocoon. But it’s not used very much. There is a so-called First Friday meeting on IRC every*

month. It started about one year ago. As many of us as possible try to be there so it's very good if it works. But it doesn't work all the time depending on the level of occupation of each of us." The positive aspect of real-time communication is partly outweighed by the possible absence of people, a disadvantage of IRC that Wechner points out: *"The advantage with email is you can answer questions or remarks, in IRC the person might be not be there any more. But IRC is very good e.g. for asking simple starter questions which have been discussed already on the mailing list. It's also a good way to just feel the mood in the community. IRC generates something like a community feeling. [...] There was a time when people started to take decisions in the IRC. So I wrote a couple of critical emails because I believe if decisions are made in the IRC many others - who are not online or don't have time at this very moment - are excluded from the decision making process. Of course it's somewhat specific considering my personal habit of not being in the IRC often - partly because it distracts me and I can't work very well any more. Others don't have a problem with having the IRC open and doing other stuff next to it."*

4.9.1.2. Mailing List

All investigated OSPs use the mailing list as their primary communication channel. Plone started out with only one mailing list which was an advantage as Bühlmann explains: *"In the beginning there was only one mailing list. It's an advantage to use only one mailing list instead of ten different ones about different topics with nobody posting anything on them. The one list in the beginning got more and more traffic and so it could be split into a developer and in a user mailing list. Later on others were added, one for each specific product. But only step-by-step: when a topic appeared again and again a new list was opened."* In the Cocoon project there are various types of mailing lists including a closed one: *"We have one mailing list for the core developers, one for the users and a closed one for the PMC [Project Management Committee]. There was a docs-specific mailing list but it's not used any more. We handle documentation issues on the main 'developers' mailing list now."* eZ publish as well has split their discussions into different groups also using a closed mailing list for a certain group: *"We have a few mailing lists. We have a developer mailing list, a partner mailing list - of course you need to be a partner to be member of that."* Even more different lists are used in the TYPO3 community: *"There are two main mailing lists, the English mailing list and the developers'*

mailing list. Additionally there are installation, marketing and user groups' mailing lists - all countries and cities have their own ones. [...] Everything that has a deeper significance will go via the developers' mailing list. First aid and political matters are discussed in the English mailing list, for example questions about licensing. There is a German list but it's not an official one. And there is also an announcement list where a few persons - Kasper, the two release managers and I - can send news. I send announcements about new versions, press releases and so on."

So what really attracts so many people to the mailing lists and how are the skilled and experienced developers motivated to join the discussion? Kraft tells of his experience: *"The general problem is that people who know how to help don't have the time to stay on the mailing list all day and answer emails. The discussions need to be interesting enough for them to invest their time. [...] The most important thing is to show that something is going on. So if somebody asks a question and he doesn't get an answer during a couple of hours he'll be disappointed. Therefore good people come back because it's also fun to communicate like that."* Hinderink confirms this notion that keeping the discussions interesting is essential to make them attractive for highly skilled contributors: *"It's also important to raise the level of communication on the mailing list so it gets interesting for people with lots of knowledge to contribute."*

4.9.1.3. Blog

Blogs are personal internet diaries where e.g. open source software developers write entries about the projects they are working on. A collection of all entries relating to the same topic but from different authors results in a so-called Planet of the specific OSP. Wechner explains why this summary of developer comments helps structuring the information flood: *"There is no Lenya Blog nor Planet Lenya. Nevertheless I believe it would be good to have a Planet Lenya, the aggregation of all Blogs that have the category Lenya. For example Henry of Midgard, another open source CMS, is very much convinced of that. He says since there is Planet Midgard he doesn't have to write many mailing list emails any more because the essence is written down in the Blogs. So there is for example some mail thread in the mailing list and the final decision gets blogged by the members like a resume. [...] Of course Blogs are*

not perfect. They are just the opinion of one single person. You can comment on the Blog entries but not everybody will see these comments.” And Rothfuss, blogger himself, shows why Blogs reach a wide area of readers: “To feel the pulse of the community it’s good to have an aggregation of all the related Blogs like Planet OSCOM or Planet PHP. It gives an abstract overview about a community. [...] I subscribed to several to be informed when something big starts to happen. I don’t want to know about every commit in these projects but want to hear about new releases or when they found out something interesting. [...] Weblogs usually have a broader spectrum of readers than the project mailing list so a wider range of people is accessed this way.”

4.9.2. Community Structure

Structuring the different tasks in the OSP community and building organisational entities may support collaboration of the community members. Examining the different levels of formalized organisation the distinct motivations of the OSP leaders are investigated. The Kupu project seems to have the least structure of all the OSPsex-
amined : *“It’s really sort of anarchistic, really open, really anyone can join and do their thing. And as long as the releases are stable it’s a also OK to mess about with the trunk a bit if you want. I do have to add that most of the people who are sort of new either work on a branch obviously or work on integration stuff in their own directory, so it doesn’t happen very often that the core is broken or anything.”* But also concerning website updates and similar tasks nobody has a clear responsibility assigned as Wesdorp continues: *“In that sense we are also sort of anarchistic. I think now three people have access to the website and if we see that something is wrong we just change that. Usually the website is pretty static actually. So we have the release and make notes about that and that’s usually done by Philipp and me, too.”* Interestingly, the attempt of the TYPO3 community to explicitly build an internal organisation failed. Hinderink explains why it did, what forms really exists at the moment and how the community shall be structured in the future: *“There were a lot of attempts at organising the community. Once it was tried to build teams for all different topics. And we must say this actually failed. Many people reported for the positions but in the end they didn’t have the time to adopt the topics entirely. The only thing that really exists is some loosely defined entourage of people around Kasper. Depending how tight you draw the circle there are more or less people*

inside. Looking at the intensity of communication there are about four or five persons including Kasper and me. It's going to change now somewhat because of the founding of the TYPO3 Association. By defining responsibilities it gets clearer." Such responsibilities are already assigned in the Plone project as Bühlmann knows: *"We designated heads of specific areas like documentation, release management, code review, subversion management."* In the Apache Software Foundation, the appointment of such persons in charge of the project requires the installation of the PMC, the Project Management Committee: *"This is another element of Apache because every Apache project must have such an organisation. In Cocoon any committer can be member of the PMC. This is not necessary, every project can decide how they want to have it. [...] In Cocoon almost every committer is in the PMC since everyone can say if he wants to join or not. The PMC has a chairman, a director who has to report to the Apache Software Foundation Board. He has to send a report x times a year to the board on how everything goes in Cocoon, if there are technical or political problems, how we advance and so on. So the board gets some kind of an overview of the current state of the project."* In Cocoon there is only a person responsible for release management but no other clear defined organisational entity: *"Then there is the release manager and that's about it, I think. [...] The committers are responsible for the rest all together. Of course everyone is specialised somewhat but that's not official. Officially there is only the release manager and the PMC chairman."* Although Lenya is also an Apache Project according to Wechner there is almost no structure in the community: *"People just do what they like to and are able to do."* In contrast to these community founded OSPs the eZ publish project is differently organized. Most of the software development takes place inside the company and the outside relations are fixed in clearly defined procedures: *"Most of the developers come from eZ systems. But we have given commit access to some [external] developers. This doesn't mean they can change everything. We have a mailing list where you need the feature to get approved if you are an external developer."*

4.9.3. Task List

The goal of keeping a task list in an OSP is to break down the plans for future releases into achievable assignments. Rothfuss believes an explicit task list motivates interested persons to start participating in a project: *"Sometimes people just need a*

little push. Of course you can say people could look on their own. But many like it when they get a proposition or an option of doing something. So if a project has some administrative guy who likes taking care of the bug tracker already in the early stages, it might help if he assigns issues to certain people of whom he thinks they might be able to do them. And then they can comment on that. Of course you can't force anybody to do something but you can give incentives and say he or she fits very well for this job and would be welcome in the project." There is a loose task tracking in Kupu but Wesdorp doesn't use a task list for coordination of development *"No, not at all. We have an issue tracker and if people post issues to that we can sort of assign people to that. But I don't think that is managed either, people just check out the issue tracker once in a while and say 'OK, I can do that and that' and tackle the issues. Just before a release I usually crawl through the remaining issues and try to get them fixed. That works quite well, usually people do their best to pick up the work that is necessary to get a cool stable release, that's cool too."* According to Farstad in eZ publish bugs are collected and he is also open to introduce a public task list: *"We do have a bug list where you can come with suggestions, bugs and enhancements. We don't have a public task list but of course an internal one. It's because 99.9% of the code is written inside our company. Of course it's something we could do to improve the community building."* Additionally, the eZ publish community may post their wishes for features to be realized by eZ systems: *"Of course you can't send an email saying we want this system and expect us to make it. But what we do before every release we ask what are the features you'd like to see in the next version, what are the things we should work on. We take all that feedback and we see what are the most requested features. We do that in more detail every year at the conference. There we discuss the next releases and what we should work on."* TYPO3 has also a task list but there remains the difficulty to realize the features afterwards as Hinderink experienced: *"People actually sign up for certain tasks so they get the information where to start. Sometimes they do it, sometimes they don't. Mostly it's done out of a mood. Somebody likes the system and wants to contribute something but in the end it gets drowned in the daily business."* The same effect was observed at the Freenet project concluding announced interest was abandoned because of the lack of knowledge concerning the architecture and the programming language.⁵⁷ So Rothfuss realized this entry barrier and thus recommends the available is-

57 See von Krogh/Spaet/Lakhani (2003), p. 1228

sues should be small enough so potential contributors don't feel overtaxed: *"There should be some tasks available which somebody new can start tackling. Just simple things to improve like a bug tracker with some ideas. Otherwise, if somebody finds your project and the only task you offer is to build a new architecture - well, where would you start? You don't know the project yet."*

4.10. Production Specific Subjects

Production specific subjects relate to community building insofar as how the program itself appeals to potential contributors. The software quality is essential for the attractiveness of an OSP since skilled developers are willing to invest time into well designed and implemented programs, only. And the user interface design raises the overall usability of the application thus improving the user acceptance of the software. Finally, the installation process is important because it usually is the first direct contact of the user with the software so this procedure must be structured as easy as possible.

4.10.1. Code and Feature Quality

Wesdorp names the feature richness and the overall code quality as attractive points of the Kupu WYSIWYG editor: *"I would say it's definitively the feature set because it's pretty feature complete compared to other open source ones. [...] But also the way we try to tackle things: It is completely object oriented, it is very clean code, it has a nice abstraction layer. I think it's partially the appearance, what it looks like and what it feels like for users. But definitively also the API. I think the code is definitively a strong point in Kupu."* TYPO3 also has a wide variety of features, mainly because of the large number of available extensions as Hinderink explains: *"And last but not least the broad availability of functionality is a good argument for TYPO3. Most of the extensions are of very good quality and ready for production - because they weren't developed based on guesses what the users might need but were created on customer demands. And this was quite easy, mostly because it's in PHP and the extension manager enables a good handling of the extensions."* To preserve the code quality it is important to check the contributions and thus also to evaluate the code submits by potential committers as it is done in the Cocoon project according to the joining script: *"So it's important that one participates in the mailing list and then sends Subversion code patches - some sort of correction or change in*

the code - to the committers. And if one person sends a sufficient number of patches the committers get tired of integrating all these patches and then say 'It would be better if this person could integrate the code directly.' Also with the patches you can see the quality of the code, if the person knows how to write good code, does it fit into the project. Then the person is proposed to become a committer and then an election takes place." Additionally, Wechner explains the importance of preventing code ownership in order to be able to replace parts of the source code with improved solutions: *"There isn't a code ownership either. In effect we try to avoid this on purpose. [What is the consequence of code ownership?] You would become dependent on people. Code owners could prohibit changes in their code. We want to be able to throw out code if there is better one."*

4.10.2. User Interface Design

Every interviewee mentioned in the discussion at least once the high influence of the graphical user interface on the attractiveness of the software. Wesdorp for example counts, next to the code quality, the visual appearance of the Kupu editor as one of the strengths of the project: *"I think it's partially the appearance, what it looks like and what it feels like for users."* And Hinderink points out the relevance of the GUI concerning content editing in the application and its customizability for end users: *"The graphical user interface is one of the features that users like most about TYPO3. Authors, e.g. have the possibility to edit texts inside the front end, so-called 'edit while you serve'. In addition the highly configurable user interface of the back end is very popular for customers."* One reason why the appearance was always taken care of was the early presence of somebody responsible for the graphics: *"So all the visual things made a big difference, icons and so on. Very early there was Emil from Holland who created these icons about in 2001."* In Plone, it was even part of the initial motivation to create a software with a great user interface: *"Zope only serves components of applications but the CMF already offered content management, workflow procedures, a portal system. But the problem was that CMF was unusable from a visual and usability point of view. So I gave up on it because you couldn't use it out of the box. Two other guys, Alexander Limi and Alan Runyan who met via IRC, said the same. Limi said CMF looks bad and Runyan agreed although mentioning the concept would be really great. They joined with the intention to create a cool user interface for the framework. So Plone was born, about*

February 2002.” For Bühlmann the user interface was indeed the main factor in the decision to work for Plone: *“The nice layout was the key point of it, although I wasn’t quite content with the standard layout of Plone. It can still be improved but basically it looked good and this was the main reason I decided to keep on working with Plone.”* And talking about successful OSPs even Rothfuss knows and mentions this uniqueness of Plone: *“Plone for example grew from zero to one hundred in a very short time. They understood what was needed to attract attention, like a very good CSS and other things that are not really related to the code.”* The Magnolia development team also realized the high relevance of a user friendly graphical interface of the content management system so they invested a lot of effort into this part. Kraft is happy about the fact that the audience appreciated this benefit and states about the mailing list feedback: *“Yes, lots of it [feedback] via the mailing list. Mainly it’s positive, especially concerning the GUI.”*

4.10.3. Installation

For a new user the first intense contact with the OSP usually comes during the installation procedure. Already this can be a barrier as Farstad mentions: *“Of course you need to get the software running.”* So next to many other things it is important to offer an easy way to install the application as Rothfuss recommends: *“Many projects don’t understand that they have to care about the last mile. Other projects who understand this like Plone are very successful. There we can learn how important it is to invest time to provide for easy installation and some out-of-the-box appeal. Even if a project has a great architecture but it’s difficult to install, only a small number of people will be interested in it.”* So it is not astonishing for OSPs with a broad user basis like TYPO3 that the most important document is the installation manual: *“There is no more important document and there are always lots of questions about it.”* Therefore there is also a mailing list specifically for installation issues with TYPO3.

Many OSPs include a demo installation to facilitate the introduction to the usage of the application. Kraft had a positive experience with this because it lead to fewer installation questions as he states: *“Installation issues are very small compared to what I read on other CMS mailing lists. We’ve got a demo installation which you can install with a double click and then show how it’s all done.”* After installation,

configuration has to be done which again needs particular knowledge about the software. Especially in applications that require further integration efforts like the Kupu editor technical questions are asked often: *“From users I don’t get much feedback but from developers and integrators I often get complaints about it being quite hard to integrate as I said it. It’s sort of hard to configure because you don’t have a configuration file where you can turn things on and off. You really have to hack a bit of JavaScript.”*

5. Conclusions

Having looked at the various aspects concerning initialisation of OSPs and onto the ones concerning already running projects now certain conclusions can be drawn. First, a list of human and project based properties shows the optimum set-up for a new OSP. Second, a framework of actions for ongoing projects outlines possibilities of OSP leaders to positively influence community building. Thirdly, two specific conclusions identify other components for long-term OSP success. Fourthly, a short list of do's and don'ts for OSP leaders gives some hints how the interviewed persons successfully acted in their tasks of directing their OSP. In the end, overall comments of this research and ideas for future investigations terminate this paper.

5.1. Successful Initialisation

Successfully starting a new OSP is a very complex process, which isn't completed by just publishing a download link of source code on a website. Its success vastly depends on the project's characteristics and on the human beings behind the project. To summarize the findings of the interview responses in chapter 3 the human and project based properties positively influencing community building are listed once more in table 4 together with a "rationale" and a "challenge." "Rationale" contains a short explanation of the property describing its relevance concerning community building. "Challenge" briefly describes where the difficulty lies in realizing the leadership skill or precondition of the project.

<i>Leadership Skills and Behaviours</i>	<i>Positive Preconditions of the Project</i>
Assertiveness <i>Rationale:</i> To ensure code quality, announced milestones, community culture and other critical elements of an OSP a strong but thoughtful leadership is necessary. <i>Challenge:</i> Since OSPs are based on the voluntary work of contributors and thus their personal motivation it is a delicate task to direct people and enforce decisions.	Programming Language <i>Rationale:</i> The choice of the programming language determines part of the image, the scope of applicability, the potential developer community and other aspects of the OSP. <i>Challenge:</i> Since programming languages have huge syntactic and semantic differences a change at a later stage in the OSP is virtually impossible.
Commitment <i>Rationale:</i> Every leader of an OSP must show an above-average dedication to work on the project investing time for the further development of the software and the community. <i>Challenge:</i> The huge engagement of the OSP leaders requires a lot of time for which they usually aren't paid because it's non-productive work in a monetary sense.	Open Source License <i>Rationale:</i> The type of open source license highly influences the future of the community because it may promote resp. force or inhibit resp. unbound community building. <i>Challenge:</i> For legal reasons and community habits it is unusual to change the license of the OSP later on so serious thought has to be given to the choice.
Communicativeness <i>Rationale:</i> To spread the large amount of knowledge OSP leaders need to communicate well in order to motivate potential contributors to get into the project. <i>Challenge:</i> Serious written communication is very laborious in general and specifically in OSPs not the main skill of many software developers.	Great Initial Source Code <i>Rationale:</i> The initial source code determines the attractiveness of the OSP by letting potential contributors estimate its potential thus influencing their decision to join. <i>Challenge:</i> It is difficult to determine the best moment when to publish the project since if it's too early the potential isn't revealed and if it's too late the demand might have gone.
Experience <i>Rationale:</i> In order to give an introduction for beginners and also to help in complex problems OSP leaders need a vast experience with the source code of the project. <i>Challenge:</i> Since OSPs usually consist of thousands and more lines of code programmers must be part of the developing community for a long time to gain sufficient experience.	Public Demand <i>Rationale:</i> The usefulness of the software for the intended audience defines its interest in the OSP and thus determines the motivation to join the community and start contributing. <i>Challenge:</i> It's impossible to anticipate the future demand for an OSP. The only possibility is to adapt the software to the skill level of the assumed users to improve the chances of success.
Helpfulness <i>Rationale:</i> To attract newcomers and facilitate their entrance into the project helpful leaders are necessary since often there isn't enough documentation available yet. <i>Challenge:</i> Time of skilled OSP leaders is limited so interested persons must show sufficient personal effort before they can expect support from experienced community members.	Degree of Novelty <i>Rationale:</i> New OSPs can be radically innovative, partially innovative by introducing new features or marginally innovative by improving existing similar applications. <i>Challenge:</i> One way to increase the novelty of a project is to implement new standards and technologies but then there is the risk of them not becoming mainstream.
Openness <i>Rationale:</i> For OSP leaders openness is necessary in various forms like being open for beginners and new ideas and to communicate openly as far as possible. <i>Challenge:</i> Openness is not a basic human mentality so for example to accept solutions of others or to be open to leave a project requires a high degree of maturity of the leader.	Applicability <i>Rationale:</i> The breadth of applicability determines the potential user community of the OSP so e.g. letting the software run on various platforms increases the audience massively. <i>Challenge:</i> Frameworks intend to be broadly applicable but on the other hand often require a high amount of initial effort and a lot of programming.
Patience <i>Rationale:</i> The growth of a healthy community takes long since experienced contributors have to introduce newcomers and show them the software and the community culture. <i>Challenge:</i> Eager developers not only need to endure the low growth rate but also should be patient enough to e.g. answer similar beginner questions several times.	Level of Communication <i>Rationale:</i> In the beginning nothing is known about the OSP and the number of involved people is small so especially in the beginning a high level of communication is important to attract new contributors. <i>Challenge:</i> Usually the OSP founders prefer to invest time in further development instead of into writing documentation – also since it becomes obsolete because of software changes.
Personality <i>Rationale:</i> The charisma of OSP leaders helps in communication and by fascinating potential contributors for their project and thus increases the attraction of being part of the community. <i>Challenge:</i> Skilled developers may not have a naturally charming character so they need to substitute it with other specific traits of personality while still remaining themselves.	

<i>Leadership Skills and Behaviours</i>	<i>Positive Preconditions of the Project</i>
<p>Presence</p> <p><i>Rationale:</i> A constant presence in the chat room, mailing list or other communication channel motivates new contributors to join the project and not feeling left alone.</p> <p><i>Challenge:</i> It requires conviction for the project and a long-lasting endurance of the leaders to stay in the OSP over several years and keep on going to be active in the community.</p>	
<p>Programming</p> <p><i>Rationale:</i> High programming skills are required of all OSP leaders since e.g. to be able to help out in architectural issues an in-depth understanding of the software is necessary.</p> <p><i>Challenge:</i> Besides a certain natural programming talent also a high education and long time experience in software development is required to fulfil this leadership skill.</p>	
<p>Responsibility</p> <p><i>Rationale:</i> Compared to developers interested in a specific topic the OSP leaders bear the overall responsibility for the success of the project including support of new contributors.</p> <p><i>Challenge:</i> There are various demanding aspects such as answering leftover emails or being responsible for a clean and friendly communication and atmosphere.</p>	
<p>Visionary</p> <p><i>Rationale:</i> OSP leaders need to be able to communicate a vision for things that are not yet realized so a clear direction can be fixed as to what and how the project has to grow.</p> <p><i>Challenge:</i> Visionary people need to be creative and generate new ideas but also have to be patient if e.g. the project is not advancing by the expected speed.</p>	

Table 4: Human and Project Based Properties for Successful OSP Initialisation

5.2. Subject-Level Promotion Matrix

The subject-level matrix is the consolidation of all interview responses concerning promotion of community building. For each “spanning subject” there are summaries for every level of community building structured into goals of this subject respectively the reasoning why it is important to undertake, and short conclusions on how to promote community building in ongoing OSPs.

	<i>Recruiting</i>	<i>Collaboration</i>	<i>Production</i>
<i>Modularity</i>	<p><i>Goal:</i> Extensions or plug-ins enable external contributions without much knowledge of the source code thus attracting different kinds of people and leading to a broad applicability of the OSP.</p> <p><i>Conclusion:</i> The structure of the OSP should be extensible and all the necessary resources like documentation of plug-in development and extension manager should be made available.</p>	<p><i>Goal:</i> Because of modularity of the software, specialization for certain parts of the program is possible among the developers. Thus they are able to freely contribute new features to the project without dependence on the core developers.</p> <p><i>Conclusion:</i> Contributed extensions should undergo a quality assurance process by experienced developers in the community. Also the documentation of the extensions must be assured.</p>	<p><i>Goal:</i> Modularization makes large software tightly structured defining clear dependencies among the modules. When most of the functionality is put into external components the kernel remains small and robust. External software can be plugged in and can also be used in other OSPs.</p> <p><i>Conclusion:</i> The architecture of the OSP has to be designed smartly to give the source code a high level of modularity.</p>
<i>Documentation</i>	<p><i>Goal:</i> Documentation fulfils the very important function of knowledge transfer thus enabling newcomers to use the software. Also it presents an easy way to start contributing when inexperienced users study the software and write documentation material.</p> <p><i>Conclusion:</i> To keep the overview it's necessary to focus on a few but always updated and focused documentation artefacts.</p>	<p><i>Goal:</i> Creating qualitatively high documentation is a laborious task where volunteers are usually rare. Still it is essential to find people in the community who create complete and well structured documentation.</p> <p><i>Conclusion:</i> The leaders should create incentives for community members to write documentation, e.g. by donating money, offering public acknowledgements or organizing publishers who print the documentation.</p>	<p><i>Goal:</i> To ensure the comprehensibility and thus the longevity of the software the source code has to be commented completely. Especially in frameworks a complete and updated software reference manual is important to give programmers support to develop their own applications.</p> <p><i>Conclusion:</i> The API of the software has to be documented completely; thus incentives for the core developers to accomplish this are necessary.</p>
<i>Release Management</i>	<p><i>Goal:</i> Frequent releases communicate the progress of the OSP to the public raising the attractiveness to participate in the project. New features and improvements of the software need to be published visibly for every release.</p> <p><i>Conclusion:</i> Release often but not too often implementing the patches submitted by contributors. The task of the release manager can also be accomplished by community members, so they have to be motivated to take over this responsibility.</p>	<p><i>Goal:</i> New versions of the software have to be released in an appropriate way taking into account the interests of the active and inactive user community.</p> <p><i>Conclusion:</i> Depending on the progress of the OSP and the size of the community a certain release management process has to be agreed upon defining, among others, feature freezes and development branches. It should be taken into consideration to release on a regular time basis.</p>	<p><i>Goal:</i> The stability and, if possible, the backwards compatibility of a new release are very important. To guard the customizations of user implementations a clear distinction between external and internal API is essential.</p> <p><i>Conclusion:</i> To guarantee the on-time shipping of a new release a feature freeze has to be enforced. Migration scripts may update customizations to higher releases. A clear API policy helps to distinguish between internal and external API.</p>
<i>Collaboration Platform</i>	<p><i>Goal:</i> A High ranking on collaboration platforms like SourceForge increases visibility of the OSP. On the other hand its own branded website is also important to give the project a certain individuality.</p> <p><i>Conclusion:</i> OSP leaders should actively use the services of collaboration platforms to raise the ranking of the project. Also the project should be listed on all well-known platforms.</p>	<p><i>Goal:</i> The responsibility of a collaboration platform is to provide modern, fast and reliable services for the work on the OSP. Also the development team requires a certain freedom to configure the platform for their needs to efficiently work on the software.</p> <p><i>Conclusion:</i> Depending on the stage of the OSP an appropriate free collaboration platform has to be chosen or a project-specific one has to be constructed.</p>	<p><i>Goal:</i> To improve the production level of the OSP the collaboration platform has to support the development process of the software as much as possible.</p> <p><i>Conclusion:</i> A state of the art and highly reliable revision control system is essential for the development process so the collaboration platform has to provide this.</p>
<i>Physical Meetings</i>	<p><i>Rationale:</i> Presentations of the OSP show the people behind the project and also are stimulating moments for visitors to try the software for the first time. Personal contacts to the community spreads confidence and motivation.</p> <p><i>Conclusion:</i> Organizing fun meetings and other social events raises the community spirit and the attraction of being part of the project. So there should be creative leaders who choose special places for sprints. In addition participating in open source exhibitions is a good way to meet potential contributors.</p>	<p><i>Rationale:</i> Personal relationships of contributors are intensified enabling better collaboration. Meetings are also a good way to accomplish knowledge transfer from experienced members to potential contributors. In addition, community meetings serve to take important decisions concerning the OSP and its organisation.</p> <p><i>Conclusion:</i> Conferences and small leadership meetings are the best way to improve collaboration inside the project. By organizing this fairplay is very important and also group dynamics must be taken into account.</p>	<p><i>Rationale:</i> The idea of a Sprint is to work intensively on the source code and advance the software by doing bug fixes, architectural improvements and implementation of new features.</p> <p><i>Conclusion:</i> Developing in the pair programming mode with a modern infrastructure supports the working environment of the Sprint. In addition, planning by setting specific goals for the event helps to stay focused on the assignment. Whether a Sprint is really necessary for the OSP depends on the size and structure of the community and the software.</p>

	<i>Recruiting</i>	<i>Collaboration</i>	<i>Production</i>
<i>Foundation</i>	<p><i>Rationale:</i> An OSP can benefit from the good reputation of a well-known foundation thus increasing the confidence of users. The association also offers a secure and reliable open source license, a correct credit system and the protection of the OSPs' brands. Additionally it may bundle the marketing power of the community and support collaboration among companies in the OSP.</p> <p><i>Conclusion:</i> The OSP should be part of a known foundation or initiate its own if there isn't an appropriate one.</p>	<p><i>Rationale:</i> A foundation has to bring stability into the project and to smooth the fluctuation of people. It also organizes the various tasks in the OSP, installs democratic structures and thus creates more transparency. By collecting donations the foundation acquires the ability to hire people if deemed necessary by the community. Especially important is the legal protection of the developers.</p> <p><i>Conclusion:</i> New associations should follow best practices of established and successful foundations.</p>	<p><i>Rationale:</i> Besides protecting the software developers against law suits the foundation also has to secure the copyright of the OSPs' source code. In addition it may canalize resources to support the most active core developers so they can concentrate on their work. The foundation is also responsible to provide a state-of-the-art collaborative infrastructure for the OSP.</p> <p><i>Conclusion:</i> The OSP leaders are responsible for improving the functioning of the foundation continuously.</p>
<i>Internationalisation</i>	<p><i>Rationale:</i> Going global with the OSP vastly increases the number of potential contributors. Additionally it offers an easy introductory opportunity for newcomers by letting them work on translations. Also knowing and visiting people from all over the world is attractive for newcomers.</p> <p><i>Conclusion:</i> To save time in later software adaptations an international community for the OSP should be planned from the start. Among others it requires all communication to take place in English.</p>	<p><i>Rationale:</i> The diversity of the community may help to overcome cultural phenomena like not-so-communicative people in the originating country. Also knowing of people working on the OSP all over the world at any time of the day has a highly motivational influence on all contributors.</p> <p><i>Conclusion:</i> To maintain a friendly atmosphere of communication conversations have to be polite and respectful. Also it may be necessary to answer questions about the different legal environments in each country.</p>	<p><i>Rationale:</i> The software should be usable in various different languages and cultural areas.</p> <p><i>Conclusion:</i> Internationalisation of the source code base involves the translation of the user interface and documentation of the software.</p>
<i>Spreading the Word</i>	<p><i>Goal:</i> General marketing activities to raise the visibility and improve the image are also important for OSPs. The goal is to achieve as much publicity as possible on various open source platforms, directories and news channels.</p> <p><i>Conclusion:</i> The OSP has to be published on all popular open source channels, especially those meant for the technologies of the project. Writing press releases facilitates reporting about the project so the probability of publications increases. Companies who own the project or offer services for it also should invest large effort into product marketing.</p>		
<i>Credit System</i>	<p><i>Goal:</i> Instead of paying money OSPs can offer raising the contributor's reputation. So the attractiveness of an OSP raises when participation in the community is acknowledged.</p> <p><i>Conclusion:</i> All contributions should be credited to the corresponding authors offering them a possibility to present themselves and assure their references.</p>		
<i>Communication Channels</i>		<p><i>Goal:</i> For efficient collaboration the optimum mix of communication channels is essential. Therefore, OSP leaders have to be familiar with strengths and weaknesses of each communication instrument to deploy the appropriate ones according to the community's demands.</p> <p><i>Conclusion:</i> The OSP leaders have to evaluate the influence of IRC, mailing lists, Blogs, websites and other communication means on community collaboration. Then they have to focus on suitable ones and adapt them according to size and progress of the project.</p>	

	<i>Recruiting</i>	<i>Collaboration</i>	<i>Production</i>
<i>Community Structure</i>		<p><i>Goal:</i> The investigated OSPs show a broad variety of stages of organisation. Therefore the appropriate structure has to be found for each project individually corresponding to its size and progress.</p> <p><i>Conclusion:</i> Anticipating the organisational structure of an OSP has mostly failed. So the leaders can only react to the changes in the environment and should not plan large organisational changes in the OSP without need.</p>	
<i>Task List</i>		<p><i>Goal:</i> The idea of a task list in an OSP is to motivate people to contribute to the project what is actually needed thus giving them a hint on how to start participating.</p> <p><i>Conclusion:</i> The leaders should try to introduce a task list or a list of open issues even though people might not have asked for it. It is important to offer small pieces of work in order not to overstrain motivated contributors.</p>	
<i>Feature and Code Quality</i>			<p><i>Goal:</i> Obviously software quality should be as high as possible. This holds particularly for open source projects where many different developers have to work with the available code.</p> <p><i>Conclusion:</i> The leaders must accept qualitatively high source code only to ensure the overall software quality. Therefore code ownership is to be rejected. Also the software should provide sufficiently useful features so the initial effort of a new developer to get into the project is justified by the correspondingly high benefit.</p>
<i>Design User Interface</i>			<p><i>Rationale:</i> Although the actual value of a software is derived from its functionality the graphical user interface still plays a major role in the handling and overall look and feel of the application.</p> <p><i>Conclusion:</i> To increase the attractiveness of the software, especially in highly competitive environments, it is essential to invest effort into the graphical user interface. This can be essential for the overall usability of the software.</p>
<i>Installation</i>			<p><i>Goal:</i> The installation process has to be perfectly organized concerning technical installation and documentation of every single step.</p> <p><i>Conclusion:</i> To test this procedure OSP leaders should investigate how users usually install the software. Feedback on how to improve this process can be very helpful. Also offering a demo installation gives an easy way to get into the program.</p>

Table 5: The Subject-Level Promotion Matrix of Community Building

5.3. Interpretations of Interview Responses

The following drafts of two hypotheses were neither expected nor explicitly investigated during the interviews. They are the result of interpretations of the interview responses describing certain tendencies in OSP communities. The first one is the assumption that mainly demand driven actions by OSP leaders will be readily accepted in the community. The second interpretation assumes the longevity of an OSP depends at least partly on the degree of dependence of community members on the progress of the OSP.

5.3.1. Demand Driven Actions

The triggering event of organisational actions by OSP leaders turns out to be mostly a response to a change of the environment in the community and not an anticipation of that change. Hinderink gives an example of such a reaction by explaining how the installation of a quality assurance process for control of the submitted extensions was initiated by the community: *“I’ve realized there is some regulatory system. The problem was recognized by the community itself and so collaboration on this issue started.”* He also experienced the opposite, trying to install organisational structures in advance: *“There were a lot of attempts at organising the community. Once it was tried to build teams for all different topics. And we must say this actually failed.”* The same happened in Plone where new mailing lists about specific topics were opened only when demand was high enough: *“Later on others were added, one for each specific product. But only step-by-step: when a topic appeared again and again a new list was opened.”* Also the detailed process of adding new features to Plone as it is in use today wasn’t an anticipated rule but a necessity at that moment of time: *“That was the time when companies had problems with projects when Plone didn’t work any more because of these side effects. So it was decided that no one is allowed to add a new feature without first writing a PLEP, a description of the feature.”* And concerning API policies to separate internal and external architecture there needs to be distinction between younger projects that still undergo lots of changes and older ones where a large community is behind as Rothfuss explains: *“So younger projects with not so many users usually don’t have these deprecation policies. It’s more important for older ones that are used extensively, like for example the Apache Web Server.”* And speaking about the organisation of Sprints and other events around an

OSP Wesdorp would organize something only when the community demands it: *“As it looks like for right now at least for Kupu 1.x we will never have a Sprint I think. I mean if people are interested, I’m interested too.”*

Such answers implicate there should be no organisational changes in an OSP unless there is an acute problem to be solved or a clear request by the community. So it may be assumed that mainly reactions, say demand-driven changes, are successful because a solution is wanted. Contrary to common management practice, anticipating problems may result in new problems. This phenomenon might be a consequence of the voluntary nature of open source communities because in commercial software engineering environments management decisions can be enforced where as in OSPs no one has authority above others.

Wesdorp summarizes this hypothesis by talking about his vision for Kupu: *“We don’t have a todo list for the community or a real direction in which we want to steer the community. We just let it flow and see where it ends. Currently this works quite well, but maybe later on we want to change that if something is going down with Kupu for some reason. [...] I hope it will last longer and I hope we can still grow, but I don’t have this roadmap where we are going to - which is probably a good thing too, I mean, we are really open to new ideas, open to new features. I hope people understand that and are willing to take that up.”*

5.3.2. Dependence on the Progress of the Project

The life expectancy of an OSP seems to be related to the degree of dependence by the OSP contributors on the progress of the project. Wechner states there are basically two different types of OSPs, those initiated and conducted for fun and on spare time usage and those where people actually make a living on, like e.g. Lenya: *“Here you see the difference between ‘hobby’ projects and those that are used in a commercial area. You will have some dependence on your customers and can’t just say we will change the API like that.”* Plone is another OSP, which is mainly used in professional environments: *“The Plone community is special. Many members have families and many have their own company offering Plone services. So in the beginning there was nobody who did it as a hobby or during his studies. They all used Plone and Plone services for a living.”* Then Bühlmann adds the important point concerning his and others’ strong personal interest in the progress of the project since it is needed to

finance a living: *“If I own a company offering Plone I do have a strong interest that it gets developed further. But one who does this as a hobby or a student who just likes it, doesn’t care about it any more once he finishes his studies and starts to work. He might even forget it and stop developing. People like this get lost, people who do open source development for fun. It does happen, of course, in many open source projects people write an application just for fun. But Plone really has a business orientation. We want to develop a system that is so good it can top commercial products.”* And there is a similar situation in the Cocoon community where members also don’t leave so quickly since their business depends on this particular OSP: *“Maybe it’s because many members of the Cocoon community also do business with it and have a company. So if something goes wrong you can’t just say ‘I’m leaving.’ because it’s not just a hobby but your business. I think that’s the reason why we take care to have a clean communication. I see other projects which are very different. They have these flame wars, nobody interferes and it gets worse and worse. This happens very seldom in Cocoon and if it does there’s always a quick reaction like ‘Here you can’t talk like this. Please come back to the right level.’ That’s very good.”* And obviously the situation is the same with the projects founded by a company because money was invested in development and marketing and won’t be given up as quickly as some student’s time.

These answers show the larger the contributors’ dependency is on the ongoing of the project, the more they undertake to continue the project and remain part of the community. Thus it can be assumed that projects where people earn a living on have better initial conditions and a higher probability to be continued in the long term than those where people work on it just in spare time.

5.4. Do’s and Don’ts for OSP Leaders

The answers of the interviewed OSP leaders often showed distinct repetitive suggestions of how such projects could be successfully managed. These propositions are summarized in the following seven do’s and don’ts for OSP leaders to give short and memorizable hints.

<i>Do it for yourself.</i>	<p><i>"As founder of a community you have to protect yourself. If you count on people's contributions and thankfulness you'll fail for sure. You have to do an open source project for yourself, not counting on outside rewards. If there are acknowledgements that's nice, but you shouldn't be depending on them."</i> (Hinderink)</p> <p>Compared to the number of people using a certain open source software only a small part of them will ever give feedback. So to stay motivated in an open source project one has to have a personal profit from it, either in a way of doing business, by the joy of programming or in any other kind of satisfaction.</p>
<i>Don't loose yourself in perfectionism.</i>	<p><i>"I believe it's important to release frequently, as Eric Raymond said. You shouldn't always say: 'Oh let's do this also and that and so on.' That way you never get to a release. Even if you fix something it's possible to have side effects, create a new bug and so on. So it's better to fix some things and then do a release. Then you see stuff that doesn't work so you fix it and release it again."</i> (Wechner)</p> <p>Most users of open source software only perceive the program in its released form so they never know about attempts at new features that didn't make it into a stable release. So OSP leaders have to release often and bravely and accept the responsibility for the well-functioning of the software thus creating bug fixes when errors occur.</p>
<i>Do accept others' ideas and work, too.</i>	<p><i>"I saw that Cocoon didn't fit our needs exactly but accepted that because it was better building on something that already had a community than doing my own thing."</i> (Wechner) <i>"[It's important] not trying to be too stubborn - that's probably a developer's thing, lots of developers are very stubborn about their project. You have to take your time to consider what people think because what people think is important."</i> (Wesdorp)</p> <p>All successful OSP initiators are very creative and committed persons who contribute a lot to the open source community. But talented developers are also often quite stubborn, wanting to program all on their own not looking at other peoples' code. So even if one has the gift of being a skilled OSP leader he should still consider appreciating the ideas and work of others.</p>
<i>Do communicate openly.</i>	<p><i>"You always had to say 'Write it in the mailing list. Write it in the mailing list. Write it in the mailing list.' People are not used to open communication, they usually communicate in a closed way."</i> (Wechner)</p> <p>During most of the year contributors of an OSP don't meet so they have to rely on electronic communication. This requires them to communicate in written language so regular practice is important to maintain a high level of communication in the community.</p>
<i>Don't speak of ideas but contribute solutions.</i>	<p><i>"We noticed that ideas are cheap but realization is costly. This means if somebody has a great idea we ask: 'Oh cool, so when do you start?' That's the usual question. It's much better if people come with at least a prototype they already programmed to demonstrate the concrete idea."</i> (Delacr��taz)</p> <p>Usually in OSPs there is no lack of ideas how to further develop the software and how to improve the collaboration. Ideas are worthless, however, if there is nobody who realizes them. So as OSP leader one has to be active in contributions, either in the form of code or in communication, in marketing services, in infrastructure work or other helpful activities.</p>
<i>Do behave nicely.</i>	<p><i>"I do think you can steer it [the success of a project] a bit as long as you make sure that you don't piss off new users, don't say nasty things about people and try to take your time to actually help people to get into the project. That way you can definitively stimulate getting a community growing."</i> (Wesdorp)</p> <p>Since a large part of the engagement in an OSP community is caused by intrinsic motives people principally want to care about the progress of the project and not about internal conflicts. Consequently it remains the responsibility of the OSP leaders to maintain a motivative communication culture in the project and to be nice themselves, even if a so-called flame war⁵⁸ should break out.</p>
<i>Do serious marketing.</i>	<p><i>"And of course you have to do a lot of marketing, make sure that you are known on the appropriate open source channels. Like for instance Freshmeat, mailing lists and so on. Of course you need an accessible and a bit of a clean website, screenshots and so on."</i> (Wesdorp)</p> <p>Usually, the absence of a large marketing machinery is one of the major disadvantages of OSPs. Large enterprises have the power to polish the image of a software even if it is technically inferior to open source alternatives. Therefore it must be one of the major goals of all community members to help with effective and truthful marketing for their software.</p>

Table 6: Seven Do's and Don'ts for OSP Leaders

⁵⁸ Definition flame war: When an online discussion degenerates into a series of personal attacks against the debators, rather than discussion of their positions. A heated exchange. Enzer (2005)

5.5. Future Research and Closing Comment

As shown in the beginning this paper largely follows the process of developing theory from case study research proposed by Eisenhardt. Although various suggestions on how to execute such qualitative research have been applied in this work some had to be abandoned because of the lack of time and resources. Especially to harden the conclusions mentioned in this chapter it would have been helpful to add quantitative data to the qualitative results, e.g. the number of downloads of each project during the last 12 months or the number of subscribers to the mailing lists of each OSP. Another possibility would have been to ask several leaders of the investigated OSPs to approve or disprove statements and thus to be able to corroborate the hypotheses. Other community members possibly would have introduced new perspectives on the project widening the perception of the OSP. Moreover, all the investigated projects belong to the software category of web applications and frameworks so it would have been interesting to analyse other kinds of OSPs, too, to check the general applicability of the conclusions.

Further research could lead to the definition of distinct success factors for OSPs resulting in a list of properties that successful projects need to possess, e.g. a certain size and diversity of the developer community or a distinct set of communication channels. So the procedure to select the projects to be investigated could be based upon such success criteria providing an objective rather than a convenient list of OSPs. Taking into account previously determined properties particularly suitable representatives of an OSP could be chosen and interviewed. The conversations could again treat the spanning and specific subjects in OSPs looking specifically at the three distinct levels of community building and extracting successful actions, which are expected to lead or have led to healthy community growth.

Concluding this paper the author expresses his hope that this writing has contributed a little bit to the future thriving and prosperity of the open source community and their great software projects. Any feedback on this research is warmly welcomed.

Appendix

To provide complete information about the conversations the entire interviews are listed in the following. This seemed important since not all statements could be cited in the paper and also because some readers might only be interested in particular OSPs. Again it needs to be remarked that although the translations were done with great care not all nuances could be transferred to the English language. Also all of the interview partners speak English very well and thus probably would have formulated answers differently if the interviews had been conducted in English.

Interview with Bernhard Bühlmann, Plone on 17 November, 2004

Please tell me about yourself, your education, and how you came to the Plone project.

I studied computer science at the University of Bern and also did a PhD thesis about 3D graphics there. We developed a C++ graphics framework which we already should have put open source. But somehow we didn't do it although there were no such 3D graphics frameworks around at the time. I also worked at IBM for about 7 years during my studies since the beginning of the nineties.

This was before the beginning of Linux.

Yes, in this time the first minix came out, the first free Unix distribution. I remember during my time at IBM a colleague of mine installed it on a PS 2 system. This was in the beginning time of Linux.

Did you hear about Richard Stallman back then?

No, I wasn't much interested in open source software. But we already used it at IBM installing it on different systems. So I got into contact with OSS. But then it wasn't yet in the strategy of IBM because they thought they wanted to earn money with Unix. They still believe this today I think, the AIX operating system, but now they turned over to Linux, too, because of marketing reasons.

Then I went to the computer services of the University of Bern working on a super-computing project. Later on I went to Zürich for two years to a start-up company Perspectix where we developed 3D graphics. After this I came back to Bern but - luckily - didn't find a job right away. So I got the idea of doing my own business in 2001. We focused on team platforms, on collaborative work over the Internet, in the beginning for construction projects. We started with Microsoft software, Sharepoint, a very good software. I installed a server with this software and rented it to customers using the ASP (Application Service Providing) business model. But then the customers' demands became more and more difficult to handle because I couldn't customize the software since the source code wasn't available. It was a powerful software but I couldn't adapt it to the customers' needs.

So I began to search for an alternative in the area of open source. I used to work with Python so I came to the Zope application server. I wrote my first Zope application which didn't work although I just typed it from a book. This wasn't a good experience. Nevertheless I felt it's a good platform.

So you actually did learning by doing?

Yes, there were one or two books so I looked up how it works and quickly realized the high potential of the Zope application server. One can realize things very quickly because lots of applications are already available and are configurable through the web. I started to develop a CMS based on Zope. Even today there's still a website working with it. But I real-

ized it doesn't work out well in the long run working on your own with your CMS. I again started to look for an appropriate software and tried a couple of Zope applications that already had a community to get support and experience of others.

What was the main criterion for your final choice?

Although I advanced very quickly with my own CMS and everything worked fine, bug fixing and testing new functionality was very time consuming. So product development got difficult. Also the architecture wasn't well thought-out because I had just started programming without much planning. It worked exactly as I wished but extensibility and maintenance weren't good so I looked for an alternative. Very soon I found the so called CMF, an additional product of Zope, a Content Management Framework. This included the foundation classes of a CMS so it was again a layer higher than Zope itself. Zope only serves components of applications but the CMF already offered content management, workflow procedures, a portal system. But the problem was that CMF was unusable from a visual and usability point of view. So I gave up on it because you couldn't use it out of the box. Two other guys, Alexander Limi and Alan Runyan who met via IRC, said the same. Limi said CMF looks bad and Runyan agreed although mentioning the concept would be really great. They joined with the intention to create a cool user interface for the framework. So Plone was born, about February 2002.

Quite soon the first release was made available. I downloaded it but found it was unusable because it was way too slow. But I realized the potential behind it. The nice layout was the key point of it, although I wasn't quite content with the standard layout of Plone. It can still be improved but basically it looked good and this was the main reason I decided to keep on working with Plone. The two initiators of Plone met for the first time in Charleroi, Belgium, at the Euro Python Conference 2002 where I also went. In a small room next to the conference they presented Plone during 45 minutes to about 20 persons. They presented the basic idea and the architecture behind it. Somehow, I don't know why, lots of people got attracted by the project. I believe the main reasons were that Zope already was an elaborate application server, CMF worked as a stable framework, and Plone had a good layout. So in other words the entire product had a high standard in every aspect.

Did anything comparable already exist at that time?

Yes, there was another product called CPS, Collaboration Portal Server of Nuxeo company from France. They did the same, develop their own portal based on CMF. But Plone, apparently, was better in the initial phase so more people got into Plone than into CPS.

Was CPS also open source?

Yes, I believe so. At least you can download the source code freely. But I don't know for sure if changes can be returned to the project.

Plone had a good release management from the beginning. Right from the start they had created a SourceForge project and let everyone do CVS commits into the project. So from the beginning they gave everyone the chance to participate in the project.

Are the two founders of Plone still in the project?

Yes, they are.

Are they both from Belgium?

No, Alexander Limi is from Norway and Alan Runyan is from Houston, Texas. They physically met for the first time in Belgium. Before that they only had contact via IRC.

IRC is important. It is actually the backbone of Plone. I'm seldom in the chat room but just yesterday I visited there because I had downloaded a software the day before. It included a file I didn't know the extension of. By coincidence Alan Runyan was in the chat so he could explain the problem: In the bundle I had downloaded a product was missing so it didn't work. He told me to add it to the bundle and upload it right away to SourceForge. So I did correct the error by entering the chat, getting the information, fixing the bug and uploading the correct bundle.

Do you also use SourceForge as your CVS server?

In the beginning, yes, but now they have a Subversion repository. They try to install their own infrastructure for versioning, project management and so on. Nevertheless lots of things are going via SourceForge.

Are mailing lists and news still on SourceForge?

I don't know where these things are. The problem of SourceForge is that in the past it wasn't working reliably all the time. There were times when check-in wasn't possible. The availability just wasn't good enough - no wonder with 70'000 software projects! This was very annoying for the Plone team so they said let's build our own infrastructure. But for a start I believe SourceForge is perfect.

What are your activities in the project? You also have commit access as you mentioned.

Well, I have to admit, I'm not the typical open source developer because I don't develop a lot. But in the beginning I did testing by putting one of the first Plone sites online. It was the website of a uni hockey team. They needed a new solution so I said I'll do it in Plone. I installed the website and thus could give feedback pointing out errors and so on. Like this I could help to improve the product by applying it to a real world project.

This means you could access the CVS repository from the beginning?

Yes, it was never a problem. They held the interesting position to be open to everyone. Still they had kind of a code review. For example if somebody checked in something that didn't work first of all they got a telling on the mailing list. And then these changes immediately got deleted. In this way a very open culture was developed where criticism is possible without generating a tense atmosphere.

So one could easily change things which worked out fine in the beginning because there were only about 30 developers. As the community grew it got more and more difficult so the release management became a big problem. For example the transition from Plone 1 to Plone 2 was very difficult because the developers wanted to implement too many features. So the product never got mature and release 2 was delayed. One company from Austria then said that they needed Plone 2 because one of their projects required features of release 2. So they began to develop their project on a beta version of Plone 2. But because there were so many changes all the time they had to rewrite their software every week. Finally, Jodok Batlog wrote a long email where he complained that he got sick of this because they had to start from scratch every single week. He demanded a professional release management. This email changed a lot. First I thought "Oops, now we've got trouble brewing!" But since the people knew each other per-

sonally no bad mood developed.

So here we see the importance of personally knowing each other. This I believe is the main success factor of Plone: people drink beer together. In December 2002 I went to Rotterdam to a Zope Sprint. A sprint is a meeting for developers where intense product development takes place. So in Rotterdam we were 40 developers - I'll show you some pictures. There I programmed together with Alan Runyan in a pair programming team. Like this I got to know him and learned a lot from him. I then asked him if we shouldn't organize a Plone Sprint, I would organize one in February 2003. So he promised to participate and since Limi attended this Sprint, too, I got him on the participants' list as well. I announced the Sprint and hardly a few days later all the 30 places were occupied. So in February 2003 we organized the first Plone Sprint of the world here in Bern. I knew Plone needed something like this; people had to get to know each other to improve working together via the Internet. So we really went drinking some beers. This was the beginning of a whole lot of developer meetings.

In May 2003 the next Sprint took place in Padova, Italy, and later in September 2003 the next one in a castle in Austria.

So you deliberately went to special places?

Yes, exactly. Here in Bern we were at the University which wasn't very special. It was a perfect place to do such a Sprint because we could install WLAN and had black boards to sketch ideas quickly. It was good but Switzerland is too expensive to organize international Sprints because of the side costs. After all, people have to pay everything by themselves, travel, accommodation and so on. In Italy we simply met in a booth during an IT exhibition. Instead of exhibits we set up some tables where we could work. It wasn't really good because it was too noisy but the important thing was to meet once again after only a few months.

In addition, others saw you at the exhibition.

Yes, of course. You talk about this and people are amazed that developers from all over the world meet there to program during three days. In the castle we worked in the living room of the owner. The dimensions of the room were about 10x10x8 meters and dinner we had at a long table like 50 knights. Those were very special, unique moments when programmers from all over the world met for one topic only. Some of them I'd already seen for the third time that year so I can say now that I know Plone people better than my neighbours here in Switzerland. So the Plone community is not an IRC or email community but a community of friends who see each other regularly. This leads to unique stories every time we meet, like if you go into a training camp with a sports club. For example at the first sprint I organized a traditional Swiss cheese fondue in the city. Limi ate way too much so he won't ever again eat fondue. For two days he suffered from stomach aches. To this day he always reminds me of this incident and that it was a bad idea to go out for fondue. I even acquired the nickname "Mr. Fondue." This is the kind of stories you tell over and over again.

This year in September we had the second Plone Conference (not a Sprint!) in Vienna. The first one was last year in New Orleans which I missed, unfortunately. In Vienna there were 300 people attending the conference, ten times more than at the first Sprint. So I would say the community grew ten times in but one and a half years. I don't know if it's an appropriate parameter to measure community size but at least it can be said it grew a lot. I even think it is one of the biggest open source conferences in existence about only one project. There were people from Japan, India and so on. So one guy from Bangalore asked if we still needed any development resources because he would suggest renting Plone programmers from Bangalore. So here we see what resulted from a meeting of two guys over IRC starting a new software and only two years later there are developers from Bangalore offering Plone services. I'd really say it's a big project now.

I saw the potential of Plone and realized there is no other platform offering such a quick web application development. This is the unique advantage of Plone: high velocity of development. No more than a few lines of code are needed to solve

a problem. We seldom write a module where we need more than 100 lines of code. Everything is connected with the framework so in the end you don't need to program that much.

E.g., for the website of Bern, did you write your own CMS or could you use Plone directly?

We used Plone as a basis and added own modules to it. Like customized entity types and so on. The interesting part of this project is that the city of Bern now started to develop in Plone on their own. We trained three programmers in Plone and now we work together with the city on the project "internet presence of Bern".

Is there also a relationship between the large Plone community and the decision of the city of Bern for Plone?

Of course they made the decision because of us. But the intention of 4teamwork always was to spread the word about Plone throughout the nation so we did lots of marketing. We founded SwissZope, an association of companies working with Zope. Unfortunately it didn't come out so well so it's a bad example of an open source community. In Switzerland there are now two Zope communities, but the country is actually too small for two communities. In the beginning there were people from Zürich who founded zope.ch. I wanted to help there but also intended to introduce business aspects like company presentations, a Plone service supplier directory and so on. But the people who founded zope.ch wanted to build a non-commercial community. We tried to negotiate but failed so we founded swisszope.ch. We also went to the Internet Expo with this label which resulted in new projects. People read some showcases on swisszope.ch and followed the link to our company. So it's not really a community now, only three companies are still on the portal.

Do you cooperate with other companies?

No, we don't fit together. It is difficult to cooperate with a company in Bern because we cover the same area of customers. Maybe it's me who is too much business oriented. We work together much better with a company from Nürnberg, Germany. There's no competition concerning customers there. Here in Bern it's delicate if you give your know-how to your competitors. In Switzerland the Plone community isn't good. There are Plone companies but without much contact. It might be a goal for next year to try to put some drive into the Swiss community. I believe Plone has advanced so far now that the demand for Plone projects has stabilized. In the beginning it was difficult to get new customers for Plone so we were happy when we were the only ones offering Plone services. But now we believe there should be as many companies as possible offering Plone. It's somewhat a chicken-egg problem. On the one hand you think "hey Plone is great, I'd better tell nobody about it so nobody besides us will offer services for it". But if you think like that only you'll end up staying the only company in Switzerland offering Plone. This, however, is bad for business. So it was always our goal to convince as many people as possible of the advantages of Plone because my intention is that Plone should become the standard CMS in the open source field. Our company does 100% Plone right now and doesn't touch anything else. It's quite a risk but on the other hand if you focus all your power chances are many customers decide for Plone. Once they choose Plone and it meets their expectations they won't switch again. Now in 2004 we are in a phase where many companies already have an internet presence but maybe not an intranet system or maybe they want to renew their website. So this is a chance for us to step in with Plone. Therefore our goal is to spread the word about Plone.

On one hand we would like that other companies offer Plone services but on the other hand there remains the question if we can exist like that. Somehow it's a strange situation. At least we're in a way better position than vendors of commercial CMS. For example Obtree got sold twice already, and they have 20 developers. They developed a good product but now Plone is better than Obtree. We've already had a couple of Obtree customers who've switched to Plone. They say Plone is much better and easier to handle. Right now there are about 150 developers worldwide advancing Plone 7 days

per week, 24 hours per day. This is a development power only big software companies like Microsoft, Novell or SAP can afford. For the middle range CMS provider it gets harder and harder to survive. Plone is advancing fast and at high quality so more and more people will choose it as their standard application. It's the same as with web servers: I suppose today you'd choose Apache. Of course there are other web servers, but Apache is standard, runs well so it's everyone's choice. Our goal is to have Zope and Plone become standards in the application server field of open source solutions.

Zope and Plone are written in Python. What are the differences between this programming language and, say, PHP or Java?

Python is a cool programming language and we have many developers in Plone who've come from the Java world. The disadvantage of Python is that it's not well known. At the moment Java and .net are number one, then comes PHP and finally comes Python, I think. But Python has some important advantages: It's a scripting language, so it's not necessary to bother about compilation. Also if there are 10 developers in Java writing an algorithm there will be 10 different solutions. If there are 10 Python developers their solutions will be very similar. You can read Python very easily and you'll quickly understand how a Python application works. The guy who developed Python tried to pick all the advantages of different languages like C++, Perl, Java and so on and combine them into Python. This is one of the reasons why Plone advances so fast: it's so easy to understand someone else's Python code! I believe problems can be solved with much less code lines in Python than in other programming languages. This is an interesting point.

How about documentation on the levels of Python, Zope and Plone?

Python is very well documented. There are several good books about it and it's not so difficult to learn. For Zope there are also a couple of books. Plone is the least documented of these three. There are large efforts to document Plone well, too. But it's a main problem and even a miracle that Plone has advanced that far without being well documented. Obviously Plone must be so good that people are willing to suffer learning it without good documentation. Sometimes I almost lost my nerve with it but when, on the other hand, I saw how bad other systems are I came back to Plone quickly, accepting the momentary lack of documentation.

Is this lack of documentation possibly a consequence of the sprints where people communicate orally?

Yes, during a Sprint you learn much more than by reading mailing lists. You can listen to talks and gain much more than through reading mailing lists. You can also program together with developers who really know it. In this way you learn much faster. It's the pair-programming idea, writing code in pairs.

What kind of mailing lists do you use?

In the beginning there was only one mailing list. It's an advantage to use only one mailing list instead of ten different ones about different topics with nobody posting anything on them. The one list in the beginning got more and more traffic and so it could be split into a developer and in a user mailing list. Later on others were added, one for each specific product. But only step-by-step: when a topic appeared again and again a new list was opened.

What is the average traffic at the moment?

Well it's so large that I can't read everything any more.

So there are more than 10 emails a day?

Yes, in the beginning you could keep up reading them but then the traffic exploded and you didn't have a chance to catch up reading them. You would need two hours a day just to read all the emails, and in addition you'd have to be in the IRC chat. There are people who do that, they live for Plone. It's become so broad. If you want to work with Plone in a company you must do that 100% otherwise you don't have a chance. Plone has become very large and complex so you

have to invest a lot of time. On the other hand you can develop extremely well-working solutions.

What would I as a novice have to do today to get into it?

First of all, download Plone, install it and start working with it! Of course, you should read books. In the past two months two Plone books got published explaining it very well. If we had had them in the beginning life would have been much easier for us. But mainly you'd just have to work with it and try to understand how it works!

Are there also tutorials on the internet?

Yes, meanwhile there are lots of different sources of documentation. We collect them on our website providing links to them.

What kind of documentation do you have?

Most important is the Plone manual describing, e.g. how to create new content types and so on. One important element are the archetypes. Archetypes are another framework based on the CMF. It's optimized for Plone. With archetypes one can create very complex content types.

Are archetypes an independent open source project?

Yes. Plone wouldn't have a chance without archetypes. They have become such an important additional product that you can't imagine being without them. With archetypes you can now draw an UML diagram using the UML editor and then create a Plone application with another script. This is a very radical thing.

Who would use this functionality?

We don't use it yet because we're not yet convinced of it. It's good if you need to create a prototype rapidly, but to transform it into an application you still need to code. Still, during one afternoon you can design a complex web application in the UML editor and then create a working Plone application out of that.

Is there a development environment for Plone like roundtrip software for UML design and update?

There is no such thing yet, but still you can add new functionality to the code by doing changes in the UML diagram and then you can generate the code again. The added method should be included then.

Nevertheless we got away from this because archetypes are already a very comfortable development environment. So we keep developing in Python.

What's the development environment at the moment?

You've got subversion for version control and a text editor, that's all at the moment.

What about auto completion and debugging?

We rarely use these. There are very good development environments like Boa Constructor. It's a development environment based completely on Python. I only saw it in a demo but didn't test it yet. It's a good environment but right now I work more efficiently with a regular text editor and a local Plone installation for testing. In addition, Zope itself has a debugger installed so if there is an error in Zope you see the stack and a line number where the error occurred. This way you save lots of time simply because of the error messages.

Don't you also want to check the state of the objects?

There is a debugger in Plone. You can start Zope in debug mode and set break points to step through the application. But we need this very rarely. It's usually not necessary because of particular mechanisms that prevent writing wrong code. For example in Python everything must be indented correctly and in Plone itself you've got a page template language that is validated against some XML scheme. So if there is an error you can't even save it. Consequently, everything that is saved is at least syntactically correct as XHTML. So lots of common errors are avoided. That's one of the reasons why development is so fast. Once it runs only logical errors remain.

How much of the current code of Plone has been written by the initiators?

I'd say about 80% of the code has been developed by the

core team consisting of about 30 to 40 persons. They are distributed all over the world like Brasil, USA, New York and Europe like Germany, Norway, Switzerland and so on.

What are the reasons it got distributed so quickly? What were the actions of the project coordinators?

From the beginning we had a Plone website on plone.org that was very open. For example you could log in and write some docu. There were no restrictions for persons who wanted to contribute.

How about quality assurance?

This was done by the core team. But they had the general opinion that everyone should be able to contribute and that they would check later if it's good or not. This was a factor. In addition the combination of people was favourable. They were not freaks but people of at least 30 years, i.e. not so very young.

Does the degree of experience have any importance?

Yes I believe so. Plone isn't a toy. Nothing against PHP but this language can be learned quickly by computer newcomers and they can create applications with it. But Plone is object oriented which PHP isn't really I believe, I don't know it very well. Anyway, Zope is an application server comparable to Lotus Notes or BEA Weblogic, it's a sophisticated framework. Plone was designed to develop company-wide portals and large internet websites. These jobs require a well founded education in computer science to be able to use Plone for developing applications. You need to know what happens behind the scenes.

What would you say is the average education of the Plone users?

The members of the core team are all very skilled software developers. Part of them has come from the Java world with about 10 years of experience in software engineering. This is obvious. The people working on Plone really knew what they were doing and how it should work. They are software architects who don't just start coding but first think about how to design the system to make it extendible in the future.

What do you think is the influence of this fact on community building?

The Plone community is special. Many members have families and many have their own company offering Plone services. So in the beginning there was nobody who did it as a hobby or during his studies. They all used Plone and Plone services for a living.

Do you think this helped to put additional drive into the whole thing?

Yes! If I own a company offering Plone I do have a strong interest that it gets developed further. But one who does this as a hobby or a student who just likes it, doesn't care about it any more once he finishes his studies and starts to work. He might even forget it and stop developing. People like this get lost, people who do open source development for fun. It does happen, of course, in many open source projects people write an application just for fun. But Plone really has a business orientation. We want to develop a system that is so good it can top commercial products.

What means of communication do you use and why?

Most important is the IRC channel. This was one of the most important elements in the beginning so people could talk to each other directly. The other one is the mailing list, it's still a central instrument although I'm not a big fan of it. In our customer projects mailing lists and forums don't work. Regular people who work in projects are not used to work with discussion forums. But in the Plone project the mailing list is important. Until today it's still the main communication channel. Then there's the chat. It's more like a meeting point where you get inside informations and where rumors are produced.

How many IRC channels do you use?

There are many channels but only one Plone chat you should use. There are companies who log the chat and read it afterwards because there you learn the most. Although informa-

tion can get lost the advantage of IRC is it's direct and fast and you can talk to each other as in a restaurant. On plone.org you can find a couple of IRC protocols of interesting chat sessions, just copy-pasted to the website. This way newcomers can visit the website and get a good introduction. Yesterday, e.g., Alan Runyan, one of the Plone initiators, was in the chat. One guy asked a very basic question and Alain took his time to answer the question. This really helps community building if you patiently answer the stupid beginner questions, too, even if you are the main developer. This was very important in the beginning of the project when there was not yet much documentation but people received help in the chat.

How is the presence of the core developers in the chat?

It was high in the beginning and it's still very high. Sometimes I get the feeling the core developers are in the chat almost 24h a day! I myself don't have enough time to participate much because we have Plone projects to do. We have to generate income for the company so we have to put priority on customer projects.

How about leadership in the OSP? For example the "friendly dictatorship" model?

So far there is not one single leader of the project. Take Alexander Limi, one of the founders. He would like to be the leader but can't because his character doesn't fit the community. He'd like to be the guru but that's not possible because of his way to solve problems. He sometimes acts like a dictator but it doesn't work. Alan Runyan, on the other side doesn't care about leadership at all. But actually he is the real leader because of his way to help everybody and listen to everyone. Whenever we meet he asks what can be done to improve Plone and I tell him all my problems. He accepts criticism and tries to improve the software so he has the highest degree of reputation in the Plone project just because of his way to be, to support and accepting criticism. That's how he reached this position.

What about his knowledge?

He does know a lot but today you can't know everything about Plone. There are so many additional products. Still he's the one with the greatest experience. Then there are many others who know a lot about one specific area like user management or content types. I'm more of a generalist so I build applications from finished components and so I know what components to take.

Are there different roles in the Plone community?

Yes, we designated heads of specific areas like documentation, release management, code review, subversion management. Yet it's a democratic process. The release manager would never say "Hey guys, next week the release comes out!" There are discussions on the mailing lists until a consensus is reached. So the one who writes the best mail has the greatest influence. Take this Jodok Batlog who wrote the mail about Plone release 2. He wrote quite aggressively but without any spelling errors and well formulated, so you realized here is someone who did think a lot about what he wrote.

What was the consequence of this email?

The consequence was a long discussion about release management followed by decisions. Within a month the release came out.

What are the important aspects of release management?

Mainly feature freeze. It's one of the most important rules to integrate new features only if they work completely so the architecture doesn't change all the time. In a Subversion you create branches, merge them again and tag them for a release, as in CVS.

Who is in charge of the release management?

Actually it's Andy McKay but meanwhile he's got a family so he doesn't have much time right now, so someone else does it at the moment. By the way Andy wrote the first book about Plone.

Is he also a developer?

Yes he is. We've introduced the Plone Enhancement Process. So if you want a new feature in Plone you first have to write a so-called PLEP where you describe what the feature has to do and how it can be realized. Today you're no longer allowed to implement new features in Plone directly. After you hand in the PLEP the release management team evaluates it and then we decide in which release this PLEP is to be implemented.

How was this structured process introduced?

Probably because we had such a big chaos. Everyone implemented new features and no one had an overview any more. The features also had side effects which led to dysfunctionality of the system. That was the time when companies had problems with projects when Plone didn't work any more because of these side effects. So it was decided that no one is allowed to add a new feature without first writing a PLEP, a description of the feature. In this way you could see what side effects could possibly occur and if necessary improvements could be made in the design phase already.

What about API changes? Was Plone 2 completely backwards compatible?

No, it got a break. The transition from Plone 1 to Plone 2 was a hard task. There were migration scripts that migrated most of the code. So projects using Plone 1 could migrate to Plone 2 without a change. It's a very big help to be able to migrate old applications to newer versions of the platform. For every release a migration script is written which let's you migrate your application to the next higher version of Plone. For example a script migrated applications from 1.0.4 to 1.0.5 and so on. There is a function that compares the version of your Plone site to the version of the current Plone installation. Then all the migration scripts are applied incrementally. It's an important point not to let the community down with update changes in their applications. Of course some work had to be done to update. But if somebody just takes the standard Plone installation he can migrate quite easily to newer Plone versions. If you do too much changes in Plone you've got problems. In the beginning we did this but now we try to use as much of standard Plone as possible. This is what's running and it has been installed a thousand times.

What about the name and other marketing aspects?

Right now there are important activities going on with the initiation of the Plone Foundation. It's like the Apache Software Foundation, an association that protects Plone and offers administrative help. Computer Associates have sponsored Plone and donated USD 100'000 to the Plone project enabling the hiring of somebody to lead the project. He has to see that everything works fine in the Plone project. He mainly coordinates the mailing list, motivates people to do certain tasks and so on.

Another aspect is that Plone gets registered now as an international trade mark so nobody can misuse Plone for his own purposes.

The third action is to establish a well based licensing model. It's a hot topic right now. They try to assign all the copyrights of Plone to the Plone Foundation. At the moment Plone is published under the GNU GPL. The Foundation is supposed to decide under what license Plone is to be published so they want to clarify the current state of right. The idea is to relieve the individual developer from responsibilities. So if a patent claim is opened, e.g. Microsoft says Plone infringed upon our patent XY, then the foundation behind Plone shall make it less likely that a company sues Plone. Before this change, e.g. Alan Runyan or Alexander Limi could have been sued because they might have broken a registered patent. But because they gave their rights to the Plone Foundation a company has to sue the foundation. They, however, can take an attorney much easier now and this might scare off potential adversaries. This instrument of power is supposed to provide security in the background for us as a company, too.

The foundation is also developing a marketing concept. Porter Novelli, a world wide marketing company, donates marketing services of a value of USD 20'000 a month by letting one of their marketing specialists work for the Plone project.

Additionally in Las Vegas there was the Computer Associates Conference in September of this year [2004] where the Plone project was presented. Alan Runyan was on stage talking in front of 3000 journalists in the room. This also promoted the project.

All these are reasons helping us to sell Plone as a solution. For example the city of Bern looked at the community behind the project, too. It helps if an open source project e.g. is downloaded over a million times. These are facts that can't be discussed away.

Are there roadmaps, goals for Plone? What could the project look like in 5 years?

There is a roadmap but to be honest I don't know it. You can look on the website. We trust the guys so we don't care much about it. Our influence is more indirectly by convincing the core developers of a new concept. For example telling Alan Runyan that we need an interface to Exchange. In this way we try to push people into a certain direction.

What's the importance of Plone interfaces to other applications?

Recently a Plone desktop has been published, a connection of Plone to Windows. It's a commercial product, one of the first that you have to pay for. Using this you can manage your Plone directories in your Windows Explorer like regular Windows directories. It's like working on a file server. In this way you can use Plone as a document management system, too, not only as a content management system.

In five years maybe Microsoft will have a product that includes all this functionality in its Windows system so Plone won't be needed any more! No, I think the next big step will be Zope 3. The first release just got published. Maybe Plone will be migrated to this new application server by the end of the year. The aim is more flexibility and still faster development of applications.

Where do you see the difference between the initialisation of Plone by these two individual programmers compared to having had a company behind the project?

For example Nuxeo published CPS as an open source project but not many people were attracted. In Plone there were only these two guys. They worked in a company but you had the feeling that they were working independently. I believe it's an important aspect that there is not one company behind the project but one or more individuals.

What do you think, when is the right moment to publish an open source project?

On one hand you should publish a software only when it's quality is sufficient. On the other hand it might also make sense to publish it early to let people give you feedback. It's difficult to say when is the right moment - it might be always and never. You need a lot of luck and the right momentum. It's as in soccer when Greece won the European Cup 2004: nobody would have guessed it. This can also happen in the open source area when the right momentum is present and the right people join the project and talk about it.

Do you think the name "Plone" has an influence, too?

No. I think it's not a good name. But it doesn't matter because a name only becomes good if the product behind it is well. I believe it doesn't have much influence. Therefore Plone never changed its name. There were intentions to rename Zope because it always appears in the end in an alphabetically ordered list and often gets confused with SOAP. But still the name is secondary. The important thing is that the name is internationally compatible.

Another aspect: I believe Switzerland is too small to start an open source project focussed on the Swiss market, only. There are too few people, developers and users. Switzerland is too small a country so you need to start your project internationally from scrap.

Last question: Are there only community members who program or are there also non-developers in the community?

For example at the sprint in Bern there were also non-developers. They didn't know how to program but wrote some documentation or did tests. These are also important team members. Or at the sprint in Austria there was this blind woman who just wanted to check how Plone could be controlled by blind people. She also did a presentation about how she used Plone. The computer told her where in the system she was but because it wasn't implemented well she lost track and complained about it. Right away developers corrected the software and now Plone is out-of-the-box controllable by blind people. This is what other companies offer for a couple hundred thousand Swiss francs, and Plone has it simply implemented. Such features make the difference. They aren't visible at first glance.

Interview with Boris Kraft, Magnolia on November 19, 2004

Please present yourself, your education and so on.

I studied computer science in Kaiserslautern, Saarbrücken and then Edinburgh in Scotland. Mostly I worked with simulation issues. I finished my studies in 1996. I started working in the internet security area and later on I got self-employed in the field of web applications, databases like Oracle and so on. Four years ago Pascal and I merged our companies. One and a half year ago we started with our own content management system. Originally we used a commercial software from Iceland, based on WebObjects because we did a lot of application development in WebObjects at that time. But it got more and more difficult to sell projects with licensing costs and we also noticed the project costs were usually higher than the licensing costs. So we asked ourselves why keep on working with licensing costs if they are a real problem? We tried to find some leasing solution with our partner in Iceland. But since the software was still proprietary and we didn't have access to the source code the customizing expenditure was higher than actually necessary since we didn't understand things that were not completely documented. Or to say it in the reverse: If a customer wants some customizing you don't need to program it as an extension but can do it directly in the system.

So we looked what kind of open source CMS were out there and couldn't find one that fitted our requirements: it must be Java-based. There were only Lenya and OpenCMS. It must

be standard based and good in usability because our customers really demand that. They don't care about XML, Cocoon and so on, for them the CMS simply has to be easy to use. For us developers it must be efficient in implementation so if we got a design we had to be able to realize it quickly.

Under these conditions we looked at the open source CMS market again and suddenly realized there was no such CMS out there. My statement at that time was still "We won't write our own CMS, it wouldn't make sense to write the 1001st CMS." On the other hand I knew that we had very good people at that time so I let Samir, the core developer of Magnolia CMS, start programming. Binary initiated the whole project and is also financing it still.

Why did you publish your new CMS under an open source license and not under a proprietary one?

During development we thought about making it proprietary but we also knew that in today's market if we published yet another proprietary CMS we wouldn't succeed. Today you need a huge amount of marketing and sales effort to promote your CMS. Just considering this perspective we couldn't afford these expenses. So one reason to choose an open source license was to get cheap marketing and a large amount of feedback.

On the other hand marketing alone doesn't help getting the investment back. So the reason for the first Magnolia release

was primarily for our company to be able to avoid licensing fees and implement customizations faster. Release two which got published this week was more for the community.

What use do you hope the community has for you?

The primary idea is to spread the word about Magnolia and its connection to obinary and simultaneously to show that we are able to handle such technologies. So we hope that people respond and say "Hey it's a great tool you've created! Can you do our implementation and help us in some parts?" This actually occurred a couple of times and we hope it will continue to grow in the future. Of course, how much really ends up as a paid project we'll see, but before we were just a small software company and now we're an internationally known software provider. This also helps in current customer presentations where we can show what we can do and the customers don't just have to take our word but can see what we've actually achieved.

Another goal of course is that we don't have to keep on developing the CMS by ourselves alone so we hope this will change in the future, too. Also we hope Magnolia helps us finding more skilled developers who are motivated by working on a well-known open source project. Such persons don't have to work on a project which nobody will see but can participate in a cool open source project and can also make a name for themselves in the open source community.

Did you get any feedback by the users?

Yes, lots of it via the mailing list. Mainly it's positive, especially concerning the GUI. But most of the people don't really react. Of the 40'000 downloads we had of Magnolia so far less than 1% have registered on the news list, only about 360 persons. On the developers' mailing list there are about 250 registered users while the number of Magnolia installations is about 4000 to 6000. We have sort of a ping mechanism - which can be turned off - that shows how many unique IPs are using Magnolia. In addition on our website you can see how many websites were realized with Magnolia but not by obinary, and you'll find no more than about 10 installations. So it's a very small number of installations that we get informed about.

What about the traffic on the mailing list?

It's varying a lot. There may be no emails for three days and then again 20 or 30 on one day, depending on the topics that are discussed. Installation issues are very small compared to what I read on other CMS mailing lists. We've got a demo installation which you can install with a double click and then show how it's all done.

What did you do to promote community building besides this?

I believe a community builds around a project when it's interesting enough and many people are going to use it. Astonishingly there are many people who started working with Magnolia release one and were immediately amazed by the beauty of the code. But most of them never said a word about it! Only when they finally posted a question on the mailing list could you tell that they knew exactly what they were talking about. For example Siemens Enterprise finished their website completely on their own and only got in contact with us because they had some problems in the end.

Once it got a Wiki about Magnolia people started to post a lot of information there. It's an important point concerning community building to lower the entrance barrier for the people.

What are the entry barriers in your project?

Everything can be a barrier. Just subscribing to the right mailing list might be one. It's not that easy in our project because I want people to first read through the Wiki and the mailing list archives before posting something on the mailing list.

What kind of communication infrastructure do you use?

Mainly we use the Wiki and the developers' mailing list for the people who want to get on Magnolia. The user list is used by the people who install Magnolia and have questions about this. Sometimes it's difficult for them to decide on which

mailing list they want to post something.

How many questions did you answer?

I wrote about 400 emails. A quarter of the entire traffic we had from November 15, 2003 to November 15, 2004, during the first release.

Did you have any specific reasons to do the release on that day?

Besides the fact that November 15 is my brother's birthday we released in November because company budgets are fixed around that time for the next year.

Concerning the mailing list, how can you manage that you don't have to answer all the questions by yourself?

You need a certain momentum which you only achieve if you're active in the beginning. As soon as you get a certain number of people involved they will also start answering emails. The small percentage in the beginning shocked me a lot; to see how few people actually helped although they had received an enterprise CMS for free! So I realized that in the end everybody still looks for himself. Now we've introduced a partnership program which gives you a gold partnership for USD 1000 and a platinum for USD 4000 a year.

The general problem is that people who know how to help don't have the time to stay on the mailing list all day and answer emails. The discussions need to be interesting enough for them to invest their time.

How do you motivate people to participate in the mailing list?

The most important thing is to show that something is going on. So if somebody asks a question and he doesn't get an answer during a couple of hours he'll be disappointed. Like this good people come back because it's also fun to communicate like that.

How can the community participate in Magnolia development, e.g. is there any task list for the community?

We want to open development with release 2 of Magnolia so we will install Subversion and continue to use Jira for bug tracking. So we hope the community can help in testing new releases and write bugs directly into the tracking system.

Additionally we are planning to post projects that describe possible extensions of Magnolia. In release 2 we offer these GUI controls that allow third party developers to write extensions in the same look and feel of the application. Later on we want to write sort of an Extension Manager that lets you install the Magnolia extensions with one click just by accessing some sort of an extension repository. Also we want to offer the possibility to write commercial extensions so you have to pay for certain ones, like in Eclipse.

How did people react to the fact that Magnolia was initiated by a company?

It's difficult to say. There were some that were happy they could just call us. There are also the hard core Apache guys that don't like the GPL/LGPL under which Magnolia is published.

What about your strategy of giving only few people commit access to the CVS?

We must make sure that external committers really know what they are doing and don't give us more work than they help us. Until they can commit on their own they have to send us patches after signing some committer's license agreement. This is important for us since we want to keep the possibility to change the license. In general we had a lot of licensing questions lately, it's a very complex topic and unfortunately there are only few people who really know about it thoroughly. Simply said we are sort of afraid that if we took an Apache license someone could just take our project away.

What's the influence of the type of open source license on community building?

It's difficult to say. I think most people don't care what license they use. At least it can't be that only the Apache license is good since most projects are published under the GPL license and still have great communities.

How much influence has the release management?

It's something you don't want to leave to the developers because otherwise the project never finishes. The release manager must be somewhat strict and decide when we're doing a new release. It's not typical for an open source project. Maybe this could change once we get more contributions from outside. But it's lots of work to do a new release. Just doing some bug fixes we can solve with patches, but e.g. including new features needs some documentation of these and description of their influence and so on. This is definitively something that the community can do in the future, too, building the installers and so on. The question is who wants to do that? Open source projects usually have bad documentation because developers don't like to write documentation but want to write cool code.

Would documentation be something that non-developers could do?

Yes, definitively. I didn't write any code of Magnolia but wrote the Template Quick Starter that describes how to write templates for Magnolia. It's not difficult at all, just making screen shots and describing how to create new users and so on. But it's still work and somebody has to do it.

What kind of documentation do you have?

Lots of marketing stuff like a features list and even a video walk through the process of how to create a new page. [...]

You use Sourceforge as download site. What do you think about such open source platforms in general?

For us branding and control of the infrastructure is important so if you use Sourceforge you'll always do some Sourceforge marketing. If you build up your own infrastructure you'll have much more freedom using your own preferred tools, like bug tracking and Wiki. Today we use Sourceforge as download site just to save bandwidth. One big advantage is that if you do a lot via Sourceforge you can rise quickly in the rating. For example the project Open Workflow where I'm still somewhat involved, gets into the top ten from time to time. John who initiated the project is currently the only active community member but he does a release and prerelease every week, writes news and so on. We didn't put our mailing list on Sourceforge because we don't want to be dependent and also we knew that e.g. the CVS had problems again and again. As I said branding is also important so people receive a nice looking website and not the one of Sourceforge.

What influence has the name of the project, in your opinion?

Magnolia is a name that can be pronounced in all countries, it's sufficiently unique for such a product but still known as a name by itself. It's a simple and easy to remember name.

Is there a possibility to rename the project?

Yes of course, Lenya did this when they were donated to the Apache Foundation.

What's the influence of such an open source association?

It's difficult to say. Apache does a lot of software but not all are well known projects.

What's the influence of standards?

Very early we started with this JSR-170 standard that defines how CMS have to access data in the content repository. At that time we didn't know if it would be published and if it would ever get any importance. Indeed there were people who found us because of our JSR-170 compliance so, apparently, we are interesting for a certain group of people.

What are the unique features of Magnolia?

Certainly the GUI, then Java and also the JSR-170.

Is there a first mover advantage?

Well, we did all the effort, the GUI, the JSR-170 stuff and marketing because there wasn't anything like that at the time. Now it makes no more sense to do it all over again but to join the community and help to advance Magnolia.

What can't be changed any more now that the project is published?

It's difficult to say. I wouldn't want to change its name because it's an asset of us. Neither could you change the technology like JSR-170. It's possible to extend Magnolia for a special interest group like "Magnolia for Group Collaboration", and also as a commercial solution. But there is still a large potential inside the CMS area. We would like to advance it especially in the enterprise CMS area because Magnolia already has some important features like clustering and separation of authoring and live environment. [...] With the module mechanism you can extend Magnolia so it's possible to build up a large community in the extension area. That makes more sense than having people in the core since it's quite finished and you would have to know all about it. It's the same as in other CMS like Communicate of Day or in TYPO3.

What could you improve today to let the community grow?

Our strategy is to do marketing and spread the word about Magnolia. For example last Wednesday we were on Heise and this had a lot of impact. Also newsletters are important like the one for Apple Developers where we can announce a new release, and websites like Serversite and JavaLobby.org. Developers' magazines would be very cool but it's quite hard to get to the right people. Also keyplayers are important like for us Tomato, a known design company of whom we could write a cite.

Concerning further community building I've talked to Daniel Hinderink, the guy who helped building the TYPO3 community. As compared to the TYPO3 organisation, he sees the possibilities of the central control of our company we have. [...]

We recently founded the Magnolia Organisation of which obinary shall be only one member among many others. Obinary will always be the initiator of the project but other companies shall have the same rights, too.

What architectural aspects of the project could help promoting community building?

Certainly the module idea. When this was introduced into the TYPO3 project the community actually exploded.

Interview with Bertrand Delacr  taz, Cocoon on November 24, 2004

What is your education and when did you hear about Cocoon for the first time?

I'm a HTL engineer in electronics because I didn't want to do computer science only. After graduating I completed additional studies in computer science. Since 1989 I'm self-employed working as programmer, consultant and teacher in the area of open source web technologies like Java, Cocoon and so on.

Cocoon started to catch my interest because of its XML publication properties. I had done some projects where we published data in the web with Java and XML in 1997 in the early days of Java. I got into contact with Cocoon quite naturally because it did just exactly this. So I followed Cocoon

until it was mature for productive use and worked with it in different projects.

What's the history of Cocoon?

Cocoon was initiated by Stefano Mazzocchi when he was still at university. He was already active for the Apache Foundation working on a module called JServ. He had to do the documentation of the website so he looked for a tool but didn't find anything. So he decided to develop Cocoon, about in 1998. In the beginning he worked a lot on his own or with only two or three collaborators. I believe in 1999, when more and more people joined, they wanted to rewrite Cocoon and implement better mechanisms like a sitemap and other concepts. So Stefano is the inventor of Cocoon and

later on more people joined development.

What was the idea in the beginning?

The vision was an XML-based publishing system that could publish content in different ways and forms.

Was there any comparable technology at that time?

Probably. But when we got a large job for the Swiss parliamentary services in 1997 we had to build our own system since Cocoon didn't exist yet and there wasn't anything else available that we could use. There were already solutions around for that kind of job but they were not sufficiently open or mature enough. So the concept of Cocoon was indeed innovative at the time.

What were the unique features of Cocoon?

The XML pipelines. You start with XML data and can add as many transformers as you want to build a pipeline. Every step can include its own XML parts and in the end you have a complete document.

Was it like that from the beginning?

Cocoon 1 was already like that but the pipeline information was contained in the document itself. This wasn't good because frequently you have documents from another source or you can't change the documents. The big innovation of Cocoon 2 was the sitemap. It tells which pipeline to use for a specific request of the client and this is quite flexible.

Is there a certain vision of the main purpose of this framework?

I'd say the vision develops with time. We're lots of people now in Cocoon, about 40 committers and about 20 of them are very active.

Are you also one of the active ones?

It depends. Over the year I'm very active during some months and then again not so much.

Back to the vision: The point is Cocoon is very open and we are just starting to call it a web application platform. This is very open. The core of Cocoon itself is very small but it includes lots of tools and libraries. So for someone Cocoon might only be a system to publish XML to HTML, another one builds a large web application with databases, XML feeds, including multi channel publishing to web, WAP, PDF, images and so on. So the vision is very open and it took quite a while to get one. I think we have it now. For three years now we have the Get Togethers, annual community meetings. Last year for example we decided to have one single form library, the classes which process the forms. Before there were three libraries but we decided to really focus to get a clear, common vision. We don't want to exclude the others but for the Cocoon project we have now Cocoon Forms. There are still people who do large applications with other libraries, it's not prohibited, but for the team there is now only one. The vision is to become an XML based application platform. This is very broad so we are always discussing where we are going, what we are doing, what has high priority, what is of low priority and so on. But at least we have the clear aim to keep the core of Cocoon as small as possible. Everything else is modular. This is like e.g. Linux where the kernel itself is not so big but there are many utilities and classes. Cocoon goes in the same direction, just acting as a kernel and then everyone can do with it what he wants.

What are the benefits of this strategy concerning community building?

I think the result will be a very open community. There are very active people in the team who work with very different technologies although they all work with Cocoon. We must be able to communicate because we have to agree on certain points. This is interesting because the people are very different and they are not doing the same work. There is a broad application spectrum of Cocoon so there is also a broad spectrum of community members.

Is there sufficient documentation?

No. Actually I got invited into the project because I started working on the documentation. In addition to the official

documentation we use a Wiki system. In the beginning I did a lot in the Wiki, helped to organize it and to set it up. Since that time we talk a lot about documentation. Our documentation isn't good compared to what you can do with Cocoon. We are not proud about that, it's a real problem. There are a couple of books about Cocoon since two or three years but there hasn't been a new one since about two years. So because there were a lot of changes in Cocoon recently the books aren't up to date any more. Hopefully new books get published soon. On the other hand writing a book is very tough because it remains a niche market and you can't hope to sell 10'000 books about Cocoon today. Economically it's difficult so we are looking for cheap ways to publish books or do professional documentation in some other form. Today's situation isn't satisfactory. Just a couple of weeks ago we discussed if every user could donate CHF 100 so we could employ some people to do a professional documentation. This would be great but it isn't easy to organize. There must be fair play and so on.

What influence does this situation have on community building? Is it rather positive because people have to concentrate more on the code or is it rather negative since people don't join the project because of poor documentation?

I'd say for Cocoon it's not that negative. Of course it's definitely negative to have bad documentation - by the way it's not bad but badly organized at the moment. There is good information but it's difficult to find.

What kind of information?

I mean e.g. references about components, how does a component work and so on. In principle everything is available but it's difficult to find.

Is there an introduction into the code of Cocoon?

There is but it's difficult because of the broad spectrum of Cocoon. For example if one writes a tutorial it always goes in a certain direction. So the next one says "Oh but I don't use Cocoon in this way, I can't really use this tutorial." We have to find a solution and I also think there will be one. For example a book about a certain case of usage describing exactly how this works and so on - always in context with the reference documentation of the components.

Back to the other question. Bad documentation is not that bad for Cocoon because we agreed Cocoon isn't a toy for "Joe User" but a tool for developers. Developers can be very productive with Cocoon but there is a lot of experience necessary because of its combination of various tools. I'd say roughly to be productive with Cocoon you need knowledge in Java, in XML and XSLT, in JavaScript - many "glue scripts" are written in JavaScript - and the build scripts to start the whole thing. This is really the scope of an experienced developer. So in that sense we thought it's better to have people fail early. People should realize soon if it's a useful tool for them or not. In the current situation without much introductory documentation there are people who say "Uff that's really nothing for me." Those that proceed really understand the code and for them Cocoon definitely fits. I wouldn't say that's ideal but it's not that bad because Cocoon isn't a tool for people who for example did ColdFusion only and don't know anything else. Or a bit of PHP and HTML but without real programming experience. Cocoon isn't for those people. From this perspective it's good that the level is high. Nevertheless it would be better to have better documentation - also being proud of the documentation isn't bad. But today that's not the case.

How much of the code of Cocoon is still by Stefano Mazzocchi?

Not much. Maybe the code of the Sitemap stems from one or two persons but generally the code is from many different people.

How does one get commit access?

The usual way is to get known on the mailing list, participate and do something good for the project. It doesn't necessarily have to be code. There are a couple of committers who don't write code at all or at least not much. We take them along because we believe they still contribute to the project. They are

e.g. “evangelists” or people who know how to write documentation and so on. It’s a minority but there are some. For example I didn’t write much code of Cocoon. I wrote some tutorials that contain some code, but nothing of the core. I fixed some bugs in the kernel but only a few. So it’s important that one participates in the mailing list and then sends Subversion code patches - some sort of correction or change in the code - to the committers. And if one person sends a sufficient number of patches the committers get tired of integrating all these patches and then say “It would be better if this person could integrate the code directly.” Also with the patches you can see the quality of the code, if the person knows how to write good code, does it fit into the project. Then the person is proposed to become a committer and then an election takes place. There are certain rules in the Apache Software Foundation. We always vote with +1 which means “yes” or -1 which means “no” or +0 which means “maybe yes but I’m not very positive.” And there are other rules that say how many +1 one must have at a minimum and how many -1 one can receive at a maximum. In the end we see if the person gets elected as committer. In Cocoon we only have few -1 because we often discuss these people in advance.

Is there an internal mailing list?

Yes, we’ve got an internal mailing list. It’s not really for the committers but for the PMC, the Project Management Committee. This is another element of Apache because every Apache project must have such an organisation. In Cocoon any committer can be member of the PMC. This is not necessary, every project can decide how they want to have it. So in the PMC we discuss about the applicant “What do you think, is this guy ready or do we have to wait a little longer to see more?” We try to evaluate all the criteria because we want the committers to stay in the project for a long time. This is important for us because it’s some work to introduce somebody to the project. Or of course it’s ok if they say “I can only work during 6 months but intensively.”

How valuable are people who stay in the project for a long time compared to those who just want to get an Apache Committer title?

It did happen a couple of times concerning the documentation: Some people came and say “Hey your documentation is nothing, I can do it way better.” So they develop large plans how they want to do that but after a few weeks we don’t hear anything any more. And that’s what we fear, people who come with very great plans and then nothing happens in the end - sometimes it does, but usually not. So we try to find people who make their way into the community gradually, learn how to deal with others, learn how to handle conflicts - of human or technical nature. We call this “the Apache way.” There are written rules and there are also some sort of politeness and methods that aren’t written down. That’s how we work.

What is the basic mentality?

Mainly it’s openness. Everything has to be done as open as possible. Sometimes it’s not possible because if somebody has a big contract with a company he can’t tell publicly for whom he is working. Maybe he has sort of an NDA, a Non Disclosure Agreement, which tells him not to betray the customer’s name. For example he works for a big company and they tell him not to say for whom or on what project he is working because it’s secret. So then you can’t disclose everything and have to say something like “I’ve got a large contract but can’t say what it is about. There I’m working with Cocoon.” So we try to handle this as open as possible. Everything we do is open. Everybody see one another’s code and so on. This has consequences on how one talks on the mailing list. On mailing lists you can communicate only through writing and it’s asynchronous. For many people English is not their native language and also the mentalities are different. We have people from Europe, America, Australia, South America, Russia, people from everywhere. So you have to take care how you say things. So when I think that somebody wrote something weird in his email I have to comment cautiously e.g. “*I believe* that’s wrong.” or “I’m a bit

worried about that.” Not “You are dumb and what you’ve written is stupid.” So often we realize we misunderstood it or he didn’t write what he meant because of his English. It has to do a lot with open communication, politeness - not meaning political correctness, just common politeness and respect for people.

In the Apache Foundation we are also cautious about business interests, because most of the committers do business with Cocoon. So we have to be aware that on one hand we work for the project and simultaneously we also work for our business. But business must not harm the project. This is important. We also make sure new committers really understand this.

In what areas can you still use new community members?

What I try to do is - because I also give classes in technical schools - is to find students who want to do their diploma thesis about an open source project. There are some great projects for these people like for example automatic testing. This is interesting to do and also very specific. In this way these students could make a name for themselves, as contributor or even as committer. If somebody writes a lot of code it will be noted somewhere - you get credits. We already have a lot of automatic tests but we can never have enough of them. Also e.g. the documentation is something where we need people. They’re harder to find than people writing code. Because usually if people need code they just write it. So when I’m working on some project and need a component then I’m just going to code it. Of course I also contribute it to Cocoon and so it and also my project get better. Maybe somebody even sees my code and thinks “Oh I can make that better.” so he improves my code. With documentation it’s different. Usually if somebody writes documentation it’s useless for him afterwards. He understands it now and doesn’t need it any more for himself. That’s why it’s much more difficult to find people who invest personal time into the documentation.

So there is actually no advantage for the documentation writer himself?

Yes exactly. When I write some code I can use it afterwards. But when I write a documentation I can’t use it any more because I’ve now learned it, so that’s it. So there’s a difficulty. Many open source projects that have good documentation got it sponsored or the project itself is sponsored so they can invest money into the documentation. The authors then earn money for writing the documentation.

I don’t think many people do contribute to open source projects just for fun. People are willing to give but also want to get something in return. So here you see it very clearly. I’ve no problem with that at all; one just has to find a balance.

So what can you get from writing documentation?

What you might get from writing documentation is a good reputation as writer, which might lead to writing magazine articles. Or as the author of the project documentation it might be interesting to write a book. For the project it’s definitively positive to have good documentation but for the writers it might be difficult to find incentives.

Maybe holding workshops or something.

Yes. For example I wrote a tutorial about Cocoon for a commercial workshop I held and after few months I contributed it to the project so others could also use it and improve on it. This can be a model, but it was just one case for me.

What tasks can be executed by non-core developers?

There are lots of things. Cocoon has a core and this might be more difficult to work on. It consists of a few hundred classes maybe, not very large but quite complicated to start with. There are maybe 10 committers who comfortably work on the core.

Stefano, too?

No he’s not very active on the code but rather concerning the vision and so on. Now he works at MIT on a semantic web project. He definitively uses Cocoon there but doesn’t write a lot of code for Cocoon currently. He did renew the build system about last year. Not that many lines of code but very important code.

Then there are the Blocks, modular parts of functionality. They are completely modular so everyone can write a new one.

What are these Blocks for?

For example the PDF formatting is one Block. It uses the external component FOP (Formatting Object Project). So the Block is actually some Java code that integrates FOP into Cocoon. This Block consists of only a few lines of code but you still have to write it for the integration of the different properties and so on. It's quite simple to write such a Block so people can participate easily.

Also writing tests could be something for non-core developers. Maybe it's even better if there are others who write the tests because they have a different view. So since Cocoon itself is very modular it's quite easy for people to contribute.

Is there a list of available or desired Blocks?

There is such a list. We use BugZilla to manage the issues so everyone can add a wish. But we also noticed that ideas are cheap but the realization is costly. This means if somebody has a great idea we ask: "Oh cool, so when do you start?" That's the usual question. It's much better if people come with at least a prototype they already programmed to demonstrate the concrete idea. Usually the committers don't have time available, they are all occupied. If they have spare time they usually work on their own ideas.

So it works only if people do the programming by themselves or at least try and then ask for help for the remaining 10%. Then we see that they're doing something. Maybe somebody can explain it to them and needs only five minutes for that. So people who have ideas but can't finish them on their own will get lots of help by the community. People who have just an idea will usually get the question "When do you start?"

Are these people usually external or internal to the community?

They can be both. Usually these are people who have a specific need. Maybe they programmed something which isn't optimized for Cocoon but if the idea is good others will take the code and improve it. It might end up in a completely new functionality.

Is this one way to get into the community?

For sure. Somebody who arrives with a specific idea and code to back it might be quickly invited.

What could you as core developers do to attract more such people?

Just make sure people receive help when they come. Even if we think it's rather a strange idea we try to make the people explain it. Only if we see it's already inside Cocoon we'd tell them "Oh we've already got that." Often they just don't know it already exists in Cocoon. People who come with code or specific examples quickly get into the community - if they have the same mentality. Frequently there are people who say, "This component is very bad. I've done something much better." When they start like this we might answer "But we already used this bad code in many applications and it works very well. So what do you think?" So we are somewhat critical, too.

Is there an explicit credit system?

Yes. We have a status file where all the changes are logged. So when I get a patch from somebody I as committer integrate it into Cocoon. Then I'd enter into the status file that I implemented the patch, which was created by this other person. That's for minor contributions. People who contribute larger parts are mentioned in the credits file. There you'll find, "part X is donated by company Y" or "developer A wrote code B." We take care to give people their credits.

What influence does this practice have on the community?

It's important to recognize people for their work. And for someone like me as a self-employed person it also has business value. This is my reference. People contribute so we have to give them something back. We can only give acknowledgement, neither money nor rights because with the Apache code everyone can do what he wants. Of course there's also the community and that's fun, but the other side

is very important.

What license do you use?

The Apache Software License. It's prescribed by the Apache Foundation. To simplify, most everything is allowed with Apache licensed code as long as credit is given to the Apache Software Foundation. It's a very open license; you can even sell your software. But you have to mention that the code comes from the Apache Software Foundation. That's all.

What do you think, would there be a difference if Cocoon were under the GPL or LGPL or BSD license?

Having a GPL or LGPL license would make a big difference, BSD not as much. The ASF license is friendly to companies so they can earn money with it. Also the committers are very well protected by the license. Nobody can sue a committer. If anything judicial happens everything is forwarded to the foundation. We are very well protected. If for example I made an error in my code and some user of my Apache Cocoon software has problems because of that he can't sue me since it's all managed by the foundation.

What influence does this have on people who want to participate? Do they actually think about that?

Not many people think about it but later on they might think it was good for them like that.

Is there a clear release management?

Yes, we have a release manager in the project. It's his job to do the releases but it's also documented how to do that so anyone could do this, technically. In addition, he has to sign the release digitally. A vote is required to do a release, according to the ASF rules.

Where do you publish releases?

There are mirrors. The Apache website has very good mirrors because it's also used to distribute the Apache Web Server which has much more downloads than we do. But we can use the same infrastructure. There are mirrors in Switzerland and elsewhere.

Can you tell more about the community? Are there other specific roles?

There are the committers, then the PMC - in Cocoon almost every committer is in the PMC since everyone can say if he wants to join or not. The PMC has a chairman, a director who has to report to the Apache Software Foundation Board. He has to send a report x times a year to the board on how everything goes in Cocoon, if there are technical or political problems, how we advance and so on. So the board gets some kind of an overview of the current state of the project. Then there is the release manager and that's about it, I think.

So there is no documentator, no bug fixer and so on?

No, the committers are responsible for the rest all together. Of course everyone is specialised somewhat but that's not official. Officially there is only the release manager and the PMC chairman.

What influence does the Apache Foundation have on the project? Imagine e.g. Cocoon were not an Apache project, what would Cocoon then look like?

The reputation is certainly good. People know that the Apache Foundation is here to stay, it is well known for many years now. Also our license is "clean" so people can do anything with the software and they will not be legally harmed as long they retain the credits and don't misuse the Apache label. For example they can't call their company "Apache Cocoon whatever." Also the logos are protected and such things. But for everything else one has lots of freedom to apply the Apache License without being sued. So the reputation is definitely important.

And there's also the infrastructure. There is a very good infrastructure since it gets paid by sponsors, companies and individuals. We also benefit from this. So it's certainly good to be part of the Apache Foundation.

It's a little bit more difficult today because the foundation has grown and there are many new projects now. It started out quickly as a very tiny organisation so we have to adapt to these changes. When more and more projects are included

one has to take care to retain a clear overview and an appropriate support of the infrastructure. Maybe it would become necessary to hire people to do that.

Is there anyone employed by the Cocoon project?

No, the Apache Foundation has no employees at all. The ApacheCon, the Apache conference, is organized by a company that gets a part of the money that is generated by the conference. Voluntary work wouldn't be appropriate here since it's an international conference. That's the model for the moment. Maybe if the foundation continues to grow we might need part-time employees like a secretary or something like that some years from now.

How do you communicate in the community? Do you have several mailing lists for example.

Yes, we have one mailing list for the core developers, one for the users and a closed one for the PMC. There was a docs-specific mailing list but it's not used anymore. We handle documentation issues on the main "developers" mailing list now.

Do you have IRC?

Yes, there is an IRC channel for Cocoon. But it's not used very much. There is a so-called First Friday meeting on IRC every month. It started about one year ago. As many of us as possible try to be there so it's very good if it works. But it doesn't work all the time depending on the level of occupation of each of us.

Are there also physical meetings?

Yes, there is an annual Get Together. This year we had it for the third time. This is very good. This year we had 130 participants, quite a lot for one single project. The first day was Hackathon, bug fixes and so on, the other day was conference and in the evening we had some social event.

Who organizes this?

It's organized by Orixo, a group of companies who all do business with Cocoon. They organize the meeting. The people are also part of the community so it's not just a business-only thing without any relation to the project. They are very sensitive about being part of the community and at the same time doing business with Cocoon. It must be fair play. And it works well.

How about entry fees?

It's very cheap, about 60 euros for the conference, just to cover the costs. I gave a lecture so I got there for free but didn't receive anything else.

Can you categorize the community somewhat?

Yes, there are people who are in Cocoon since the beginning. Others did just one specific project with Cocoon, for example generate large PDFs with Cocoon. Again others have a broad engagement since they work as consultants. Or there are companies who use Cocoon as their only tool. So yes, there are certainly categories. Also because it's modular.

Do you see people who for example start using Cocoon and later on work on the core?

This is the usual way. Cocoon is really a tool for developers. Usually after some months they get into the code and so they can also help to contribute. This is a big advantage of the project. The users of Cocoon must have almost the same competences as the developers which means they can quickly change from users to developers. In other open source projects that's not the case. For example I started Jfor some years ago, an XML converter to RTF, Rich Text Format, for word processors. There the user competences vary a lot. Users just want to generate documents so many of them don't know Java and thus can't help with the code. After several months there still was no community. It was much more difficult to grow a community because the people were different from those in Cocoon.

So you initiated Jfor yourself?

Yes, it was for a customer of mine who agreed to make it an open source project. It runs, we didn't change the code for one year now, it's stable. It's limited, you can't do everything

you'd like with it but it works well. It's published on Sourceforge.

What influence does the name "Cocoon" have for the project?

Mainly that it's well known now after some years. You can't tell from the name "Cocoon" that it has anything to do with XML and web application. People know more or less what it is about. Sometimes they have a wrong idea but at least they know it.

Could you comment on the future of Cocoon, in one or five years, say?

The functionality is quite stable now. We're rather in a consolidation phase because last year not that many new features were added. There is one major change coming concerning the configuration of the functional Blocks. Now you have to compile them but we would like to do this more dynamically at run time. So you would just need to download the Blocks and plug them into Cocoon. This will be a big internal change in the future. Otherwise I believe we've used the 2.1 release for almost one year. It's stable and you can do a lot with it. I think it might even stay like this for the next couple of years. Maybe 2.2 will include the new Block system. Technically there will be not much change.

We speak about documentation almost every week so I guess we will improve in this area. We also see many new names on the users' mailing list, names we don't know. So this means it grows and more and more people use it.

And can the members help each other?

Yes the users' mailing list is meant for that. The users shall help each other. Many developers are not on that list. And still it works. Some developers do look at the user list. But sometimes we just leave a question. It's better the users help each other so they increase their knowledge and we save time to concentrate on developers' issues. I believe we have about 1000 people on the user list and 500 on the developer list. And about 40 have commit access.

What entry barriers are still there?

I think the most important is to say exactly what Cocoon is and what it is not, so people would know quickly if Cocoon is something for them or not. With the current documentation it might take some time for a novice to realize if it's something useful for him or not. If we want to enlarge our community we have to simplify this. On the other hand we might not want the community too large because we also have to support it. It has to grow at a certain rate so we can still bear the growth.

I forgot something about the community structure. In Cocoon there are just the committers but in the Apache Software Foundation there are also the members. If you are a committer in a project and the members see that you're not just interested in your own project but also in the Apache Foundation in general you might get nominated to become a member. There are about 1000 Apache committers worldwide in all projects and about 130 of them are members. And then there is the Apache Board, the directors, six or eight people who are elected each year.

Are there any additional comments you would like to add concerning community building?

I think respectful communication is very important for us. If a conflict happens on the mailing list, for example people get angry and disrespectful, we interfere immediately. I believe the Cocoon community is very good. I'm there since about 2000 and I think since then only two people left the community because of disagreements. That's not much compared to the about 100 daily emails on the developers' mailing list. Of course we don't agree all the time but as long we can discuss openly and with respect it's ok. But if for example somebody says something wrong another might write back "Please apologize, what you said was not right." It has happened a couple of times so far. After some incidents people start acting respectful automatically. So when I realize I wrote something wrong I reply myself: "Oh I'm sorry, I didn't mean it like that." I believe this is very positive. Maybe it's because many members of the Cocoon community also

do business with it and have a company. So if something goes wrong you can't just say "I'm leaving." because it's not just a hobby but your business. I think that's the reason why we take care to have a clean communication. I see other projects which are very different. They have these flame wars,

nobody interferes and it gets worse and worse. This happens very seldom in Cocoon and if it does there's always a quick reaction like "Here you can't talk like this. Please come back to the right level." That's very good.

Interview with Guido Wesdorp, Kupu on 25 November, 2004

First I would like to ask you about your education and how you got into the Kupu project.

My education is not really relevant. I did some High School but that's about it. It was during the .com-Bubble I decided there was money in computers. I was always into computers like playing games, making music and so on. So when I saw that people are making money using computers it still interested me a lot. I decided maybe now was the good time for me to teach myself more about them and get a job in that direction. And that worked. So I worked for a couple of web developed companies first as a HTML designer, then I started hacking JavaScript and then PHP, later Python. That's how I came to work for Infrae, the company I work for now. They're doing Zope development mainly and open source development. That's how I got to know the open source world. I really liked it and stayed with Infrae for a while. They needed a What-You-See-Is-What-You-Get editor. At that point there were some possibilities and we tried some. Most were either very hard to implement or didn't work on the browsers we had in mind like IE and Mozilla. We use Mozilla all the time. After some point when we tried some proprietary and some open source ones we decided to build one for ourselves. Or at least adopt an existing project. That was Epöz at that moment. We worked on Epöz for a while. Then Mike Jablonski said he did want to develop it further for himself. That's why we formed the project called Kupu. That's what Kupu is right now.

So Kupu is actually the descendent of another open source project?

Yes, right. There was a small editor written by Mike Jablonski who does a lot in the Zope and Plone world. For us it was an ideal project to step into but it didn't have all the features we wanted.

For example?

It hardly had any image support. You could add a URL to an image but you couldn't browse to it. It wasn't very configurable and it was Zope-only. This was something we decided wasn't a good marketing aspect of the editor so we decided to change that, too. It used pretty old JavaScript techniques and so those things we wanted to change. We asked Mike Jablonski "Would you mind if we joined development?" Then he sort of told us he didn't plan on developing it further by himself. So that's why we decided to develop it ourselves completely or take over the project control. And later on Mike Jablonski decided he did want to develop it a bit further because people still had bug fixes and that sort of thing. That's how it all started. Then it became Kupu.

When was this?

This was about one and a half years ago [2003].

Now you are several persons, Paul Everitt, Philipp von Weitershausen and you.

Yes, there are currently a couple more. It started out actually, I think, Infrae asked Paul Everitt to work on the project. At some point he had some problem with design and blabla. Then I joined him in developing Kupu. That's when the first Epöz release got built. So at that point it was only Paul Everitt and me. And Philipp von Weitershausen joined at about that moment. He didn't do much JavaScript programming but did more the Plone integration and some other Python backend stuff. Those three are still in the team so I'm still one of the main developers. Paul Everitt sometimes does some stuff, too, and Philipp von Weitershausen still does some administration. Some additional developers now like

Duncan Booth who works in some other projects, he works on Plone integration. I think currently he does more checks than me. And there are a couple of people who work on separate implementations such as Rolf Kulemann for Lenya and some other people. So the group is still growing and I hope that it will continue.

Yes right, we would love to implement it into our PHP5 framework. I wrote to you before.

Yes you wrote, if you have any questions just let me know!

That's probably going to happen. Can you actually say there are different jobs in your project? Or can you say that everyone works on every part?

Everyone is allowed to work on every part but as usual everyone has his own favourite parts to work on and some stuff he benefits from. I think there isn't really someone in the project that controls it and says "This should be done and this should be done." Everyone just works on the thing that he thinks is important for the project or for his personal benefit at this point. And so far it works quite well. Obviously we're still a quite small group but it works nicely.

Do you share one common vision or goal?

Not really, it's pretty anarchistic in that sense. So it's really who wants something to be done he does it. And so far it didn't turn into clashes which is sort of surprising. Still it seems to work quite well. In the end it's probably still me who will say it. Usually some feature starts out on a mailing list, like someone asking "Hey is this possible?" And if so, why didn't we do it? So we gather discussion where the feature is discussed and sort of forked out. And if we all agree it's a good feature someone will write it. But it's not always like that. Sometimes people just start writing features, add them and it works out. We do have the advantage in Kupu that it's quite much large so the chances of features clashing, even though they can't coexist in the same setup, are quite small.

When was Kupu published the first time? Like on the OSCOM Website.

We didn't go to OSCOM at first. The original Epöz project was run on zope.org because it was a Zope-only thing. So we decided to do the first releases on zope.org, too. And it wasn't until we went to Switzerland for some reason. I think there was a Sprint or some discussion about editors on client side that can help open source content management systems. That's when I got to discuss with Philipp von Weitershausen by the way. We had the Epöz issue at that point. That's when we decided to change the license to a better known open source license. So it's now BSD style rather than the Zope Public License it used to be.

Why did you do this?

Mainly because Zope Public License, it's liberal and all that but it's not too well known. So if people hear Zope Public License... We wanted to get rid of the association between of what is now called Kupu and Zope because Epöz used to be a Zope only thing and at that point we already decided that we wanted to have it working - because it was quite easy to get it working - on different systems. So that's why we decided to get rid of the ZPL and to put in a plain one, a better known one. It came to a C license. It's also easier for instance for the project we're working on at Infrae, Silva. It's also to be a D license so it cannot use something like the GPL, it has to be something more broad. BSD is just very plain and simple. So we decided that and also we will be moving to OSCOM and we decided on a new name. So the first Epöz release was on zope.org and Kupu, I think it was Kupu 1.0.3 - because we

continued the version numbers we started out with Epöz 1.0.1 and 1.0.2 on zope.org I think - and I think it was Kupu 1.0.3 on oscom.org the first release.

Why did you do all these changes?

Mainly because we were still in Mike Jablonski's area. I mean Mike Jablonski had somewhat promised "OK I will not develop it any more, you can have the project." When he decided to continue anyway we sort of felt sorry for him. It was sort of a nasty situation because people didn't know what package to download anymore. "Was it Epöz 0.x or Epöz 1.x or Epöz ng or all the new generation?" It was really confusing so we decided we didn't want that anymore. And also it was bad for an open source project name if it had different forks and that sort of thing. That's why we decided on a new name and a new image. That worked rather well. Occasionally we still hear people searching for the Epöz 1.x version and "Where can I find it?" But it was way less painless than I expected.

What are some technical aspects that have an influence to the topic of community building?

Kupu uses a basic widget that's provided by the browsers. That's called Content Editable for Internet Explorer and Midas for Mozilla. That was an Internet Explorer only thing and at some point the Mozilla guys decided that's a very useful widget. There was no standard but everyone was using Internet Explorer any ways so they decided it was a nice idea to pick that up, too, and use it in their system. Unfortunately it's not entirely one-to-one mapped but it's usable now, at least for basic editing such as making words bold and that sort of thing. The real core of Kupu is some browser built-in thing and then there is a large gloss around that which controls that and puts some additional function on top of that to make it easier to get like more custom element types - the widget only supports a basic set of element types. That class provides some methods to make it easier to add stuff like tables for instance and images. There is also a set of tools which are basically plug-ins. Kupu works with those plug-ins even for the core tools. The basic tool set is already done by using the plug-in architecture. And then there is a set of helper classes to control the selection for instance and a large set of browser abstractions because there are still a lot of browser differences.

What do you think is the influence of plug-ins concerning community building?

I think it can be very important. Right now we unfortunately don't focus on plug-ins that much. But I'm trying to shift from using the plug-in architecture not only to add core features but also trying to turn the core features into plug-ins and try to exploit the plug-in architecture by making it easier to add third party plug-ins and extensions. But right now I think we still do too much in the core which is not that much different from actually writing the plug-in. It would be quite easy to grab the stuff from the core and release it separately. Maybe we do that at one point because it's nice that you don't have to care too much about all kinds of extensions and like niche areas if you do a core release. So it makes maintenance easier for sure.

Would those help people to get into the project without knowing everything?

That's what I hope. I mean Kupu is definitely a framework in the sense that you need not building your own libraries for your application but you put small scripts into existing applications. In that sense you need to know a lot about the framework. For instance Zope. You can't do anything inside Zope without knowing at least half of the application's API. I think that counts a bit for Kupu, too. Although we tried to keep the plug-in API simple and comprehensible, I think we kind of succeeded in that. I don't think it's really hard to write Kupu extensions. Maybe it's even easier to write extensions than to configure it. That's probably not a good thing.

How about documentation? Do you have a tutorial to write such plug-ins?

Yes, it's definitely geared towards developers so it's not very elaborate and doesn't explain in detail how to do each and

everything. You are obviously working in a JavaScript environment which is really corny and there is quite a lot of stuff to be told to actually to get something to work. And the documentation doesn't contain any side notes how to build something in JavaScript and so. But I think it's decent in the sense that it's short and compact but it explains what you need to know. For an open source project we're sort of okay I reckon.

What do you think, what would change if you had a more elaborate, more detailed documentation?

Yes I think it would help a bit although I think would still be... I mean you can't explain everything in your documentation so you'll always get complaints from people that they don't understand this and that and we should have explained that in documentation. We have that now and I think we will always have that. I wonder how much of a difference it will make. But sure, if you have a bit more explanatory documentation it will probably help in development. Although I do have to add since we have a mailing list and a lot of active developers... I don't think it's hard to get the information you want by asking on the mailing list. That definitely makes a difference.

Where could you use now the support of the community members? What tasks could non-developers do and what tasks could not-core-developers accomplish?

I think people who aren't developers don't have that much stuff to do in Kupu. I mean for the users there could be some documentation written. There is some documentation actually but it's specific to Silva and there is no Kupu author documentation. Well it's really straightforward. It's a quite familiar interface that looks a lot like any old commercial word processor so I wonder if there is much to do in that sense. But for developers... I think you don't have to be a core developer to work with Kupu. If you pick up a document "About extending Kupu" you can have something working in a couple of hours. That can be very useful to the core too. And actually we have some people just occasionally drop by to fix small issues and small CSS thing and that sort of thing. Those people are very useful too. I think it's nice about the Kupu project. Usually if you have an open source project it's a core group of developers and not much people around it but in Kupu there are a lot of different stages of development by different people going on. And that's nice, that's also encouraging.

Who has now commit access? What would I as a newcomer have to do to get commit access?

I think the only thing you have to do is showing that you are enthusiastic about changing something and preferably showing that you are not going to mess up anything. Currently we are on Codespeak, a server that is run by a friend of mine. It has kind of a nice setup there that he has this SVN [Subversion] repository and some other stuff that is shared by all the users. So if you are a developer on one project you also have commit access to all other projects. And of course this means you have to be sort of trusted to get access. But right now we don't have really high standards to get commit access. If you mention on the mailing list you want to commit something chances are that you'll have commit access now or later. In that sense I think we have currently 20 or 25 developers in the project.

How is the release management organized?

That's usually done by one or two people. Usually by Philipp and me. Currently there is a release which is worked mostly by Duncan Booth. I think he'll have more of a saying in this one than Philipp and me do. But usually it is done by Philipp and me. It's just easy: We say one or two weeks in advance "Please check in your changes, we do a branch and do a release of that branch. And that's the branch..." Nothing fancy.

So you don't have a clear feature freeze thing?

We try to do that with branches and of course we have sort of a bad release candidate if we need to... So we do have sort of a stage where there is a feature freeze. We are not allowed to add any features unless it's unlike something breaks.

Besides the release management are there any other tasks in your community? Can you structure the community somewhat? For example is there somebody responsible for the website or for any other thing?

No, in that sense we are also sort of anarchistic. I think now three people have access to the website and if we see that something is wrong we just change that. Usually the website is pretty static actually. So we have the release and make notes about that and that's usually done by Philipp and me, too. So actually I think in a sense Philipp and me are like the spokes people/managers of the project. But it's really very loose. So if anybody wants to do something they are definitively invited to do so.

So you are a quite open community?

Yes, yes, that's what we try to be!

Is this also part of your philosophy?

Maybe part of my personal feeling. Personally I'm a hacker that likes to work on new projects and I'd love it if these projects could gather a life of their own. That's more or less what I'm trying to do with Kupu. It's not that I want to step out of it completely at some point but it would be nice if most of the work and thinking is done by someone else. Currently I think we are succeeding in that quite good. I think it's always important that there is someone who is really taking care of the project in a sense that he always keeps an eye out if things are not really going wrong and that sort of thing. And there is quite a large probability that I'll always be that person and that people somewhat depend on me. There should be always someone who answers emails on mailing lists that don't get answered and that sort of thing. I don't see someone else picking that up easily. But that's only a minor part of all the work that needs to be done. Currently I do some check-ins every now and then, specifically for Silva and Infrae, but most of the features are written by other people, that's really cool.

Why would you like to have a large community?

I don't know why. It's good for the project obviously. More people use it and more people develop it. It depends whether it's not going to be chaotic. You don't want something chaotic. But as long as it is managed a bit of a proper way I think it's nice to have a large community of people thinking about and discussing it. And it's just fun in a way to see something growing, that's probably what I personally like about the project.

Where are still barriers to prevent your community from growing? What needs still to be done by you as the coordinator to enlarge the community?

As I said the mailing list is important. People keeping communications going, people helping new users and new integrators and new developers, that's very important. Usually if you are a developer of a team you answer to the emails you find interesting or important. And that means that some of the emails will not be answered. And I think it is important that those get answered, too. So that's something I still do a lot for the community. And of course I manage the mailing list and website, that's still a lot of work. And every now and then I try to sort of veto features and help people out find their way in the API. And of course I still do some development from time to time.

Can you sort of categorize the people in your community?

Yes, I think probably most of the people are integrating Kupu in their project and benefit from either extending or changing it a bit. Or they are trying to do special configurations. There are usually only developers, hardly any users. We do get some user complaints but these are mostly from the developers, too, because they are usually the spokes people of the ones that actually use the editor.

Could somebody who doesn't know how to program design new buttons for example?

I think you need to know JavaScript and a bit of the Kupu API. As I said currently configuration is still a bit hard, it's mainly done in JavaScript and that's why you at least need to know a bit of JavaScript to get your stuff working. Also for

the behaviour behind the buttons you need to know a bit about the framework. Usually if you add a button you add a tool too, so usually a button gets a very simple plug-in back end sort of thing.

Do you coordinate development, for example with a task list?

No, not at all. We have an issue tracker and if people post issues to that we can sort of assign people to that. But I don't think that is managed either, people just check out the issue tracker once in a while and say "OK, I can do that and that" and tackle the issues. Just before a release I usually crawl through the remaining issues and try to get them fixed. That works quite well, usually people do their best to pick up the work that is necessary to get a cool stable release, that's cool too.

Do you have written down the names of these people, something like a credit system?

Yes, we have a credits text file in the package. I don't think it mentions all the developers. Maybe I should correct that. It has been a while since I last did changes in that. But apart from that it's really sort of anarchistic, really open, really anyone can join and do their thing. And as long as the releases are stable it's also OK to mess about with the trunk a bit if you want. I do have to add that most of the people who are sort of new either work on a branch obviously or work on integration stuff in their own directory, so it doesn't happen very often that the core is broken or anything.

Is this a typical way to get into the project?

If I get to speak to the people who get into the project usually they are already quite far because they already picked up a project at home - and the sources are quite small still, about 15'000 lines of code, so it's quite understandable. So it's not too hard to get into the project, so usually the people to whom I speak on the mailing list are either indeed asking for small issues because they are configuring or integrating something and don't get it to work. Or they have already played around with the API a bit and really know their thing and want some help. And sometimes I do have to point out that "Hey, wait, what you are trying to do is already covered by some helper library or something."

So this is also one of your tasks as a project coordinator or initiator to give information that is only known by the someone who knows the project entirely?

I think that it's both because I'm the project manager so to speak - I find that hard to say about myself because I'm not really a one-person-thing, it's really the community that manages it itself, and that's very cool. I really like that about Kupu. But it's also... Since I wrote most of the API I know the API better than anyone else I'd say - maybe Duncan nowadays knows as much as I do. So Duncan and me are usually the people who help out with API questions.

About communication: Do you interact mostly with the mailing list?

Not only. Our IRC channel is always quite active and a lot of people pop by at IRC before visiting the mailing list. It's just easy to get an instant response, it's an easier way to communicate. And then they choose following the mailing list or the IRC channel, too.

Is there any log of the IRC channel?

No, but I think you can log it by the way, I think there are different options to do so. You can either do a client log by yourself depending on the client you use. Or you can even ask some logger to join the channel, I don't know much about that. But currently our conversations are not logged indeed so it does happen that people ask the same questions twice but then we will answer twice, it's not that bad.

Did you already have a physical meeting like a Sprint or something?

Not specifically for Kupu. I don't know if that's ever going to happen, too. It's people are just far away. I think that Kupu is already quite mature that in a sense that - it can always use new features and extensions but... Maybe for a complete revision. At some point I would like to rewrite Kupu from

scratch because I don't like certain parts about the content editable parts - it's impossible to change those. But that's really way away from now so I can't do any more assumptions about that. As it looks like for right now at least for Kupu 1.x we will never have a Sprint I think. I mean if people are interested, I'm interested too. I think development, as it goes right now, goes quite fast. Not like these huge, monolithic projects that just need a Sprint to get the first head start or so.

Is this a certain characteristic of large projects?

I was sort of afraid that you would continue on that one! It was just some statement. But let me consider this before I give some stupid remarks.

You are an expert, you have experience.

Well expert, I've been only programming for six or seven years and I work in an open source company for only three years or so, I'm not really an expert. But I know something about the community, at least I should.

So maybe, I think so. The sprints I've seen so far are usually for larger projects. When I think about a Plone Sprint for instance it's just to get stuff done for a project that is already long running. It's not by definition for a monolithic start-up or something. I was maybe a bit overdoing it.

How important are personal relationships among Kupu developers?

I don't think that they are too important. I mean it's nice, especially for Paul Everitt and me it has been nice to discuss some issues one-to-one, just for the start-up. We made a lot of progress there. But I think most of the people that are currently developing Kupu have never met each other. I'm probably the one who knows most of the other people. But it's not like important in a sense "Hey we met so we can work better with each other."

Why is there a difference compared to Plone where personal relationships are very important?

I'm not sure but maybe a difference is that Plone is way larger so you can actually help with large parties to do a lot of hacking and have a good time and also make it sort of co-ordinated because I reckon in a project as large as Plone it's way harder to coordinate the development team. It's definitely good to know with whom you are working.

Are you registered on SourceForge or any other open source platform?

No, we used to be on SourceForge but partially because of the slowness of communication we quit. When you post something to the mailing list it takes a couple of hours before it arrives, CVS access is slow. That's understandable since SourceForge is a large project. We were looking for something more speedy. Partially because of that and partially we liked the idea we decided to go for Codespeak. It's a bit more open than SourceForge. As I said if you have check in rights on one project you have the rights on all projects. And also you share this website. It's a rather cool project, it's fast and has new tools and good infrastructure.

Do you still have an account on SourceForge?

We used to use the Epox account and I think this is still used by Mike Jablonski.

Could there be any marking reason to have an account on SourceForge?

Yes, I can imagine there is. People look around for an open source project and can find your project easier. But for us the advantages didn't count up for the disadvantages.

Is there any influence being an official OSCOM [International Association for Open Source Content Management] project?

Yes, at least because it's a cross pollination thing. We haven't been too active on that point but I do know that a lot of OSCOM people at least know about Kupu, want to integrate it into their system or already have it done. There is a bit more communications going on. We have a couple of shared mailing lists, not really active though. It's a bit useful at least, I think we still could improve it though. I would like it

to be a bit more active.

What would you like to change?

Currently we are indeed an OSCOM project but the Kupu team is not really active in the OSCOM community. It would be nice if that changed a bit. But that also requires some efforts on my side but to be honest I'm very busy though.

How is the feedback of the users?

From users I don't get much feedback but from developers and integrators I often get complaints about it being quite hard to integrate as I said it. It's sort of hard to configure because you don't have a configuration file where you can turn things on and off. You really have to hack a bit of JavaScript. But I think that is improving and apart of that feedback is very positive. A lot of people are impressed of what Kupu does. And I still get a lot of feedback of people saying "Wow, I didn't know this was even possible in JavaScript!" I think the feedback is generally very good.

Besides configuration stuff, what do people demand?

We have a lot of requests for internationalisation. I currently wrote a JavaScript library for internationalisation so we can use that. And we had some requests for accessibility. And the rest are usually just some small issues like if I press enter I get a break in Mozilla and a paragraph in Internet Explorer. And of course we have the sort of visionary developers who every now and then think of a really cool new feature and want to add it.

Concerning marketing: How do people find your project?

I think a lot of people find us by just googling and reading new pages. Also we try to do some marketing from the company Infracore I work for. Of course OSCOM helps there and I think Codespeak helps a bit there too because a lot of developers who are interested in new and innovative projects go to Codespeak. Although I have to add that most of them will probably be more innovative and a bit more Python developers than people who are just using a What-You-See-Is-What-You-Get editor. Zope.org helps also, we still post some news items to zope.org. And of course also to news channels and Freshmeat. We do only post items on Freshmeat so we sort of use it as an announcement channel.

What about the name of the project "Kupu"?

We couldn't use "Word" so we took the Maori version. And it sounded nice, we were looking for a short name. Actually it was invented by a co-worker of mine. After a long vote on the mailing list we decided that would be the best name.

What do you think is the influence of the name?

I don't know. It's hard to tell how much influence it has. On the one hand it's just a name, on the other hand it's the string - especially currently with Google and such - if you have an unique, well findable name it's definitely important. There is some importance but it's hard to tell.

Could you still rename the project?

I think it wouldn't be that wise. Obviously we could but we had to start a lot of things over and it would not be understood by the users.

In general, how are people attracted by the Kupu project?

I would say it's definitely the feature set because it's pretty feature complete compared to other open source ones. The license obviously, it got the most liberal license you can have. But also the way we try to tackle things: It is completely object oriented, it is very clean code, it has a nice abstraction layer. I think it's partially the appearance, what it looks like and what it feels like for users. But definitely also the API. I think the code is definitely a strong point in Kupu. And what we try to do is leverage modern techniques of a browser. Rather than using the old style JavaScript coding we are trying to use new features of the JavaScript language and also newer features of browsers to get a better user experience. For instance the PUT, we use HTTP PUT rather HTTP POST. You can use POST too if you want to but that's definitely something that appeals to any user.

Are there any comparable open source projects like yours?

Yes, you've got the bitflux editor but this is a way more XML editor that tries to be more geared to really editing XML schemes and that sort of thing. It's a very cool system by the way, too. And there is HTMLArea and so on...

A large difference is between Kupu and the other ones is that Kupu is quite new compared to the other ones so it uses some modern techniques.

What is the difference that Kupu was initiated by an individual developer compared to a company acting as the initiator?

Kupu was first developed by Mike Jablonski but I would consider that this was a blue print for that what Kupu is now. Then it was picked up by a commercial company that makes open source software but makes money with it. I think this was quite important to help with marketing and that sort of thing. Just to give it a real push. I probably wouldn't have done it if it wasn't for the time I could have spent at Infracore. Probably the same counts for Paul Everitt, I think. I'm not sure of course.

Why didn't you call it Infracore editor?

From the beginning on it was clear that Infracore didn't want to market this as their editor. They wanted it to be an open source project. Actually part of the reason that I started the open source project was because Infracore explicitly didn't want to maintain a What-You-See-Is-What-You-Get editor themselves because it was a huge amount of work. I liked the idea so much writing a What-You-See-Is-What-You-Get editor that I decided to propose to do the initial work and a lot of the maintenance work in my spare time. And to do the Kupu work that is actually interesting for Silva during Infracore hours. Infracore has written Silva and some customers wanted a What-You-See-Is-What-You-Get editor for that so that's why we went looking for a What-You-See-Is-What-You-Get editor for the first place. Kupu is only partially a company sponsored project but the company backup was definitely what made it possible to have got a feature complete editor. So I think I spent about half the time at the company and half the time at home. Kupu wouldn't be what it is now if it weren't for either one of those parts of time. It's really a sort of a bit weird mix. When I was at work the Infracore people were pushing me to get the Infracore work done and while I was at home I looked at Kupu from a completely different perspective, from a manager of an open source project. While I was at work I looked at it from an integrator's perspective. It was much fun to be these two persons at one time, also in relation to the company. I think Infracore is a great company that I can actually work for Infracore.

If the project was only from the company, would this have made a difference concerning community building?

Yes, I think so. I'm not sure, it depends how Infracore would have tackled this. If I look at other projects at Infracore - and I can't really imagine why they do it like that - but usually we don't have much outside developers. People can send patches and that sort of thing but that's not a community-like sort of thing. While at Kupu I really try to emphasize that community thing by letting as much people work on it as possible. But if you have a product which you try to sell you have to take more care having it stable and accessible and consistent all the time. From a community perspective of Kupu we can sort of decide to do strange, crazy things at times and even release that and if it's really bad we can subtract it from the feature set.

What is the advantage of having you as a spare time project coordinator and Infracore as a project sponsor of Kupu?

Concerning me I think it's an advantage because I can look at it from a pet project kind of view. I can just decide "Hey I think it needs that." and can spend like 20 hours on that specific thing even though no one is asking for that. For the other hand Infracore did really provide us with a lot of time and a lot of drive and a lot of testing. That's the advantage of a large company. You have customers, you have people you get feedback from them instantly, you have also a drive to have certain features done because the customers ask for that. It's really a nice symbiosis there.

What kind of people would you like to have in your project?

I think I'm quite happy with the people we have right now. Of course I would be happy for just about anyone joining. As long as they try to be productive I definitely stimulate everyone to join if they can. It's nice to be talking about features and directions to go. A group with really completely different people that still have the same goal. That's just cool to watch that. But we don't have a todo list for the community or a real direction in which we want to steer the community. We just let it flow and see where it ends. Currently this works quite well, but maybe later on we want to change that if something is going down with Kupu for some reason.

And what kind of people aren't interesting for you?

That is a very hard one... There seems to be a group of people that do not really understand that if they benefit from something it would be nice if they contribute something. It's not that I wouldn't want that people as users but we can't really use them in the community. I think that counts for any community, that's just a general rather than a personal issue specific to Kupu.

I would say we could use everyone, please join!

What would you like to have Kupu look like in 1 or 5 years? What is the vision of your project?

As I said I would like to rewrite the whole Kupu but this may not be in one year and not in five years either. We are just aimed for stability and hope for new browser features. Now we are sort of at a verge of what browsers can do I reckon. Obviously there can be a lot of other features too. I don't have this clear roadmap. I do think we can improve a lot. Obviously world domination would be nice. I hope it will last longer and I hope we can still grow, but I don't have this roadmap where we are going to - which is probably a good thing too, I mean, we are really open to new ideas, open to new features. I hope people understand that and are willing to take that up.

It's one of the goals of Kupu to leverage new browser features as soon as possible. But unfortunately we are stuck with Internet Explorer because currently it looks like development there is almost dead.

And I think there are still a lot of things possible with JavaScript. I'm currently working on another project too that does WebDAV browser and a lot of other neat stuff. I think we can still do a lot of cool stuff, too, even if we are stuck in Internet Explorer.

So you are also working in another open source project?

That project is a pet project of mine that I'm currently the only developer of. I'm going to release that and I hope probably in December. It's going to be a JavaScript WebDAV client which does sort of all the content management tasks completely from the client. So the only thing you need is a WebDAV capable server and then you just can login to that and create new directories and new files and that sort of thing. It's going to be released in the GPL so the idea is after the 0.1 release has been done people can join - maybe I can build up a community too, we'll see. As I said I like starting up new projects but at some point I like to see them getting picked up by others too.

What should you bear in mind building up a new community?

I think the easiest way - although I'm not talking as I had much experience or something because actually the only project I helped starting out was Kupu and that already had a bit of a start. I think it's very hard to do the initial ground work from a community. But as soon as you do some releases and try to get yourself known in a lot of mailing lists and make sure that you help anyone that wants to join the project. It takes a lot of effort but I think it's not that hard. As long as you let people know that you are open for development and have an interesting project it should go from itself.

What does it mean "if you have an interesting project?"

That's hard to say. It's so hard to say what people find interesting. That's one of the things with open source, you can find something interesting yourself and you may have a lot of

benefit of it. But if other people don't find it interesting you'll not find any developers. I don't know, it's really hard to say. It's also something dependent on trends. It's very hard to say if a project is going to be successful or not. I do think you can steer it a bit as long as you make sure that you don't piss off new users, don't say nasty things about people and try to take your time to actually help people to get into the project. That way you can definitively stimulate getting a community growing. And of course you have to do a lot of marketing, make sure that you are known on the appropriate open source channels. Like for instance Freshmeat, mailing lists and so on. Of course you need an accessible and a bit of a clean website, screenshots and so on.

Do you know any additional "do's" and "don'ts" concerning community building?

Don't try to piss people off. Even if someone is complaining about all sort of things on the mailing list, don't ever make fun of people, take people seriously and take your time. And don't be the one who starts a flame war. And do take your time. That's really it, it needs a lot of time, a lot of patience, a lot of openness. And not trying to be too stubborn - that's probably a developer's thing, lots of developers are very stubborn about their project. You have to take your time to consider what people think because what people think is important.

Interview with Michael Wechner, Lenya on November 31, 2004

What is your education and how did you get into Lenya?

Originally I studied theoretical physics at the ETH Zürich and after that I worked for three years at the Max Planck Institute in Düsseldorf. There I got into contact with the internet and HTML for the first time. [...] I wanted to publish a science journal with a global design so the layout and the content had to be split apart. Then I began to write a CMS, first with Perl because it was very simple to implement on the server side. Then I learned about Java Servlets and XML and so on. I also worked at the NZZ, Neue Zürcher Zeitung, where I got into all the cross media publishing stuff. [...] After some time I heard about Cocoon and downloaded it. [...] Later on I contacted Stefano Mazzocchi to ask if the pipe lines of Cocoon could process binary data, too, but he answered this would never be possible. That was about the end of the communication because I thought this was important. [...] So I started to develop my own CMS called XPS, Extensible Publishing System. [...] With my company Wyona we developed our open source CMS, basically for the NZZ but from the beginning published under the Apache License. By coincidence I got into contact with Giacomo Pati who was involved in the Cocoon project and also worked in Switzerland. So I had again contact with this community and personally met some developers. I saw that Cocoon didn't fit our needs exactly but accepted that because it was better building on something that already had a community than doing my own thing. So in 2002 we migrated the XPS - which was then renamed Wyona CMS - to Cocoon. And in 2003 we donated the CMS to the Apache Foundation.

Why?

At that time the Apache Foundation had decided they wanted more top level projects - like the Apache Web Server or the Java project. [...] Stefano wanted to make Cocoon a top level project and have the CMS and other Cocoon based projects as sub projects. He liked our software, because we had integrated other stuff, so he asked us to donate it to the Apache Software Foundation. For us this made sense because it was open source anyway and also because of visibility reason.

What were the consequences when you joined the Apache Foundation?

The community didn't grow enormously. We already had about 150 people on the mailing list when we joined the Apache Foundation. How many we are now, I don't know. Nothing changed radically, maybe because in the beginning we were an incubation project, one of the first ones. Apache didn't want to become a second Sourceforge so they introduced the incubation process. In this way they can see e.g. if the community works. One common problem with donated projects is that they are often driven by a mono culture - all the developers are from one company only or there is only one developer. For the long term it's important that the projects have a diverse community. So the projects who come out of the incubation time may not have more than 50% of the developers in one single company. Lenya - as our project is named now - achieved this and we even became a top level project this summer.

So how many community members do you have now?

I think there are about 10 to 15 committers.

How did you do community building in the beginning?

We didn't do anything special, just answering mails and letting people contribute. One thing that we did consciously was to open up our development. Initially, the developers came only from our company so we had a lot of internal communication. So to make sure that the outside world was informed about what was going on we consciously decided to communicate via the public mailing list only. So now we practically don't communicate about Lenya internally any more. This is very important because otherwise it would seem like we did something for our private business cut off from the outside world. I've heard of projects where this became a major problem - even in Apache projects. For example the Xalan project, the XSLT processing, was created by Lotus, IBM. So one office was located just in front of the other one and after they went out for lunch together they announced we will do this and that. They are really nice guys, the problem was just that the discussion didn't take place in the open. So if you come from a company or any other closed entity it's important that the communication gets shifted to the outside. You need to learn to distinguish between issues that only concern the open source project and thus must be discussed in public and between issues that are company specific, which you don't want to communicate to the outside world. After all, everyone on the mailing list has his own interests and that's ok. A problem arises only if you push your interests in ways which usually aren't possible for everyone in a community like consciously leaving others outside of the decision process. In the long term that's bad for the community.

How do you open up communication?

You have to make your people aware that if they want to discuss something they have to do it via the mailing list. That's not very easy. I believe in our project it's working this way now but it wasn't easy to achieve. You always had to say "Write it in the mailing list. Write it in the mailing list. Write it in the mailing list." People are not used to open communication, they usually communicate in a closed way. You can give them incentives, however. For example give them a way to become visible. But there are people who don't want that or don't need that. It's important that they know that the project needs this open communication, otherwise it dies and this is bad for the company.

In all the Apache projects there are many companies participating, so you see very well - because you know the guys from the conferences or because you watched them on the mailing lists - when they made a decision before communicating the issue.

What else is important to remember if a project is initiated by a company?

Another important issue is to distinguish between company time and spare time. We had to learn that when we do changes in Lenya for a customer's project of course it's ok to do that during working hours. But if somebody wants to code

a features for Lenya without relation to a customer project then it's better done in spare time. Otherwise a mixing of work and leisure time occurs which isn't good. At least this is my opinion.

Again another issue is the visibility of the company in the open source project. For example we as a company dissociated ourselves from the project almost completely. Others know that we belong to this company only from our email addresses or because they know us personally. Most of the people just see the project and the Apache Foundation and any connection between the original company and the projects gets lost with time.

What about the Apache branding?

This has a large influence because today Apache is a real brand. Nowadays you don't have to explain to an IT manager why it's good or bad to use the Apache Web Server because over 50% of all the websites are hosted with the Apache Web Server. This confidence spills over to the other projects. Of course it's not sure that you can sell your project just because of this but at least it's a door opener, you don't have to do much selling any more. They still ask how many committers do we have and if we will become a top level project. [...] But with an Apache project you can make money because it just sells better than others. [...]

What influence does a company have on its open source project?

Lenya is quite neutral because it's an Apache project. Of course a lot of the committers are from our company so we can influence the project that way. Take e.g., backwards compatibility: If somebody breaks backwards compatibility in a stable branch then all our customers won't be happy any more; and not only our customers but all the people who use the software. I will think of all our customers who will need an update so it's bad for us and all other installations of the CMS. And yet, I won't go and say "We don't do that." It's the way of communication. We don't say: "We are Wyona and because we don't want to have that we don't do it." You can't communicate like this, but you can say "I know customers who will have a problem with this change." And it won't be different with others.

Here you see the difference between "hobby" projects and those that are used in a commercial area. You will have some dependence on your customers and can't just say we will change the API like that. In software engineering it happens that after a while you realize you should have done things differently. If you are a single developer you can say, even if others depend on it: "I don't care, I want to do it my way!" and restructure your entire API. But if you are in a commercial environment where people build on top of your software then you have to decide if you want that people continue to build on your software or if you want to do your own project. [...] There is still the possibility to create another branch. For example with Lenya we did this, one branch which is backwards compatible and another where we try new things and thus it isn't backwards compatible.

What about release management?

We don't really have a good release management. We don't have a release manager who really feels responsible for that in general.

What means release management in your project?

To see that everybody keeps the code freeze and doesn't check in after this. [...] I believe it's important to release frequently, as Eric Raymond said. You shouldn't always say: "Oh let's do this also and that and so on." That way you never get to a release. Even if you fix something it's possible to have side effects, create a new bug and so on. So it's better to fix some things and then do a release. Then you see stuff that doesn't work so you fix it and release it again. Somebody also proposed to do regular releases, for example every month. I like this because it's clear when you release.

And in what steps do you suggest to advance monthly?

[...] In 0.0.1 steps, like having Lenya 1.3.1, the next month 1.3.2 and so on. At the moment we only do maintenance releases like bug fixing and so on. Sometimes we also transport

new functionality from the development branch to the stable branch if it works well. [...]

When you started, was there another open source CMS based on XML publishing?

No, only recently some new CMS have appeared, for example Daisy and Hippo which are based on Cocoon. [...]

What documentation do you have?

We use Javadoc to document the API.

How important is API documentation?

In a framework it's very important because developers have to know how to use the components.

Also there is documentation for the integrators, how to implement the CMS and so on. End user documentation practically doesn't exist - because Lenya is so easy to use. [...]

How about the community, does it participate in development?

Yes, a lot. Unfortunately many things never get published in the community. Sometimes I see that it's just the top of an iceberg that gets back to the project. For example just some days ago somebody said something about DocBook so I asked him what he did with it. He answered he had built sort of a document management system for his company based on Lenya.

How could you get these people to contribute their extensions back to the project?

It's difficult. You shouldn't embrace people too much and invite them too openly because they might be shocked and run away and never show up again. Still I asked him if he would like to check the code into the project and after a while he answered yes, he could do that. [...]

What would I have to do to get commit access to your project?

First you'd have to get onto the mailing list, ask questions, fix bugs, contribute good code. All this for about one year or so. The older a project becomes the longer it takes to get commit access. On the other hand the mentality of the Apache Foundation also has changed over the last years. If somebody is around for about one year and regularly contributes something it's ok. Actually there is practically no committer who harms a project. They are usually very careful checking in code. In the worst case we can always rebuke the people about their behaviour. We also have a commit mailing list where we get an email whenever there is a commit.

Is there any structure inside the project community?

Not really. People just do what they like to and are able to do. [...] There isn't a code ownership either. In effect we try to avoid this on purpose.

What is the consequence of code ownership?

You would become dependent on people. Code owners could prohibit changes in their code. We want to be able to throw out code if there is better one.

Are there any non-developers participating in the project?

Yes there are. One guy has a documentation commit access and isn't really a developer but rather a consultant. Others of the University of Zürich comment as end users about usability issues and things like that. [...] Also for example if there are bugs they report that. [...]

Do you use IRC?

Yes, usually there are about five to ten people online there. But I'm not a big friend of IRC. Generally I think IRC is a good thing but the problem is that IRC is synchronous, not asynchronous.

You could log it.

Yes but there would be too much nonsense on there. [...] Also the advantage with email is you can answer questions or remarks, in IRC the person might be not be there any more. But IRC is very good e.g. for asking simple starter questions which have been discussed already on the mailing list. It's also a good way to just feel the mood in the community. IRC generates something like a community feeling.

Do you have any fixed meetings on the chat?

No, people just log in and talk. There was a time when people started to take decisions in the IRC. So I wrote a couple of critical emails because I believe if decisions are made in the IRC many others - who are not online or don't have time at this very moment - are excluded from the decision making process. Of course it's somewhat specific considering my personal habit of not being in the IRC often - partly because it distracts me and I can't work very well any more. Others don't have a problem with having the IRC open and doing other stuff next to it.

Do you use any other communication channels?

Maybe Blogs. There is no Lenya Blog nor Planet Lenya. Nevertheless I believe it would be good to have a Planet Lenya, the aggregation of all Blogs that have the category Lenya. For example Henry of Midgard, another open source CMS, is very much convinced of that. He says since there is Planet Midgard he doesn't have to write many mailing list emails any more because the essence is written down in the Blogs. So there is for example some mail thread in the mailing list and the final decision gets blogged by the members like a resume.

So in summary I'd say IRC is for very quick and simple questions and answers, the mailing list is to discuss issues in detail with the entire community and the Blogs are in the end the consolidation of all the discussions and decisions, which are published. And the aggregated Blogs in the Planet XY show the different opinions of the developers. [...]

Of course Blogs are not perfect. They are just the opinion of one single person. You can comment on the Blog entries but not everybody will see these comments.

Do you have any guidelines or behaviour codes for the mailing list?

No, just the standard Apache mailing list guidelines. Communicating via the mailing list is an advantage for the efficiency but it's also good for the culture in the project. To be friendly, and humble... Well, sometimes I think you still have to say some harsh words if you are really annoyed. For example recently somebody checked in code which he didn't test before. So I wrote a nasty comment because I had said it politely many times before but nothing changed. I just think one must not commit anything before testing it. Well, and then I received many critical comments from all over the place about giving a bad example, destroying community and so on. OK, I will apologize for not saying it friendly enough but the fact remains that nothing had changed before and I got very angry about that. So I believe sometimes you're allowed to smash your fist on the table. [...] I think it's also important to show ones feelings and not being humble all the time. On the other hand "humble" might also mean something different. Like not being all too self-confident and not rejecting criticism. To ask for the opinion of others, to be thankful for the work of others and also to praise them.

What other leadership skills are important for open source project owners?

Besides some amount of this good humbleness it's important to communicate your vision clearly - and to give arguments why you have this specific vision. For example explaining why one believes a trend has arrived and how to react to that. The other skill is to be able to let go and let others do the work in their own way. [...] This is part of being open and letting people come into the project.

And also the long term perspective is important. And to represent the values and culture of the community.

Nevertheless I believe in strong leadership. This doesn't mean shouting: "Shut up everybody, I'm talking." This is a misunderstanding of strong leadership. It means to be very clear but still open for consensus, bring others to consensus, and without compromise draw a line to those who aren't willing to. Because if we want to be such and such a community everyone has to cooperate and if anyone doesn't he'll have to leave. [...]

Do you have any physical meetings for Lenya only?

Yes, there was one about every year so far. It would be great to have more of them.

What influence do they have?

They're good to advance the project and let it run. They're motivating and giving an impulse for the community. On the other hand there is also a threat. When you meet all those people, spend time together and go out and so, then some sort of an inner circle develops. So you have to be aware that such an inner circle doesn't start making decisions and fencing off the outside. But I don't think this is the case with Lenya.

How do you mean this?

For example your boss likes to go out and drink so you join him and slowly you become part of this inner circle. Another guy who is really great, might not like to drink that much so all of a sudden he is an outsider. It's always the same thing. In open source communities, too, there are people who are more respected than others. Like in Animal Farm: "All animals are equal but some are more equal." So it's important that those people who have the power in the community take care that this doesn't happen. This is another aspect of that strong leadership. [...]

So you'd say personal relationships are important?

Yes, not to say it's good or not, it's just a fact. As everywhere connections help. [...] At least, in a community the effect of such relations is much smaller than in a company. It's a vicious circle where only those who are stronger and have connections rise up and not those who are actually better. Maybe they're just not good in developing these relations.

How would you define a good community?

One that is honestly friendly. Diversity helps to guarantee stability by including different perspectives. And the community must be able to find consensus and at the same time define priorities. And of course it should be able to create something and not to get stuck in discussions and let the project stand still. [...]

Interview with Daniel Hinderink, TYPO3 on December 3, 2004

What is your education and how did you get in contact with TYPO3 in the first place?

I studied economics and political science and did postgraduate studies in marketing at the London School of Economics. For a long time I worked as project manager for different organisations specifically in the area of NGOs. I got into general contact with open source software because in my studies I had to do some statistical work about currency exchange rates. This involved enormous calculating power because the exchange rates had to be calculated over several years minute by minute. Therefore we worked in a computation centre in Munich in a Unix environment. It was common to work with LaTeX and also with Linux, when available. So I grew into this in a natural way. TYPO3 became a topic for me years

later because I had to manage a relaunch of a consumer's internet website. They did most of their external communication with this channel and had several static platforms hosted on one single system. They also had eight permanent authors so it was quite a large project that could not continue with static pages only. I had to propose a new solution on a certain budget. It was big enough to accommodate mid-priced commercial CMS. During my search for a CMS I came across TYPO3. In many ways it was a completely different software at that time compared to now.

When was this?

In fall 2001. It looked quite different then, the user interface was different from the one used today. The classic backend

was much harder to use. Also the logo was another one and the typo3.com was very difficult to navigate. [...]

When was it published for the first time?

In 2000, at about the time of Kasper's wedding so it's easy to remember for him, sometime in the summer. It was already a mature system, when I came along. So in the end I had made a selection of three different CMS and presented them to the customer. I had already done some tests with TYPO3 because I believed I should really know what it meant to implement it. Moreover, there was no local service provider for TYPO3 at that time that could be hired. I worked together with two freelancers from Munich who already had some experience with it. Finally we got started with the project. That's when I became infected with TYPO3. I realized the amazing advantages of this system compared to the then available commercial alternatives. I was generally convinced that all new media projects should be implemented or migrated to CMS's as soon as possible. Having some years of experience already I knew that large websites get out of control when they are built statically. I felt the demand for a new design and TYPO3 seemed to be the best candidate.

Did you know of other open source CMS?

Yes, we had already used other ones, Contenido, Zope and others. TYPO3 seemed to be the most flexible one with the best user interface. Those were strong factors. Also customizing was possible.

How did you participate in the beginning?

First I sent an email to Kasper saying, that I was very interested in the project and had such and such an education with this and this experience. I offered to write up the advantages of the system from a newcomer's perspective and also to write sort of a starting tutorial. He reacted quite sceptically at first and actually wasn't very much interested in it. I was sort of disappointed because I had thought by doing this I would get his support. He supported me nevertheless but was not very motivated about it. I kept on going and tried out things on my own. The community, very small at that time, also supported me a lot, which was very helpful. I liked the system and its environment.

In the following year 2002 I was already involved a lot and wanted to take an active part to find out where to go with TYPO3 in the future. I participated in ongoing discussions, especially the one about the introduction of a modular concept.

Can you briefly describe the history of the project?

Kasper and a fellow student founded the internet start-up company SuperFish in Copenhagen in about 1996. They developed good partnerships with marketing agencies and other companies in Denmark. [...] After a while they began to realize writing their own CMS would improve website creation. So they started development in 1997. [...] Because of different visions about the company Kasper and his partner split in 1999. Kasper took his software along and worked on TYPO3 for an entire year. Then in July 2000 he published the first version 1.5.

Why right at that moment?

Maybe partly because of his wedding and to get it off his chest after working alone for such a long time. And he probably just felt it was ready. He also published it also because he's a believing Christian and thought the software might be useful to others. Doing it open source was some sort of self expression. He didn't expect much of it. But then others came along and started to use TYPO3. Among the first ones was Rene Fritz, one of co-authors of the TYPO3 book. The first public release was TYPO3 1.5 and until release 2 you'll find some very interesting installation procedures which needed some sportive ambitions to succeed.

How did he publish it in the beginning?

He opened up a new website and first announced it at Hotscripts. It didn't start like an explosion but grew continuously. Only in the last two and a half years it did grow dramatically.

What was so special about TYPO3 then?

I believe it was the sophisticated user interface and the availability of a large range of functionality like a simple address book and a small shopping system. There were about ten plugins that came along with the system. Compared to other PHP projects at that time the code quality was very high. Maybe the worst example is PHPNuke which started out small but many people joined and then lost control over quality. [...] TYPO3 was different because Kasper had already prepared a lot before publishing it. It was much more complete than other PHP projects. Kasper was actually rather a designer than a programmer at SuperFish.

What is his education?

After school Kasper started to study electro technics but quit after a while. [...] As mentioned before, with Superfish he was originally in creation and only out of necessity got involved with PHP. So all the visual things made a big difference to him, icons and so on. Very early there was Emile from Holland who created these icons about in 2001.

What was the influence of a single person initialising the open source project?

I believe there are more advantages than disadvantages to this fact so far. Kasper earned lots of sympathy by not doing business with TYPO3 for a long time. It psychologically lowered the entry bar for participation with the project a lot. My assumption is that community contributions are much harder to get for a project where a company is doing business with – just because people feel like helping the competitor or supporting somebody who has business interests. In the case of Kasper he as an individual still benefited from the success but he also had invested a lot into it. There's a much more personal relationship to the project for everyone involved if there is an individual programmer at the centre.

What does he live of today?

He has always made a living with customer projects. In addition there is a certain amount of donations, not much but at least some. [...]

So back to the initial question of having a single person as an originator. One advantage was this lower entry bar for the community. Another one is the issue of quality assurance. Kasper provided a very complete project – maybe this is also one reason why no one ever had the intention to do a fork so far. It wouldn't be that smart anyway because there are lots of extensions available today which would have to be made compatible to a new system. [...]

Also it's sort of a transparency advantage if there is only one person responsible because in this way the project gets something like a pivotal point. In a company it might not be that transparent. [...] Personal wishes of the project owner like "I always wanted the community to communicate in a friendly tone." also give continuity to the project. So it stays more personal. The annual snowboard tour is another psychological factor binding the community together, because it's not just a conference, but a fun event. [...]

Of course the single person project does have disadvantages as well. A major one is the non-scalability of Kasper – an expression adopted from the Linux kernel community "Linus doesn't scale." The creation of the extension repository was one attempt to decrease the importance of this problem so Kasper can now concentrate on the central issues again. It was successful although for example the typo3.org domain and the documentation were things that Kasper initiated himself. It needs to be distributed more among different people. One major effort to do so is the recent foundation of the TYPO3 Association with its residence in Switzerland. The idea of this association is to distribute different tasks like quality assurance of the extensions and the service providers.

So is there still some tendency to move from a single-person project to an institution-based project?

Yes, in parts. Kasper will still be the head of this institution. So what it will do is up to him and the others inside this organisation. From my point of view there was also some Glasnost idea behind it because transparency and openness are also issues. The association is supposed to have a beneficiary effect for everybody. It won't go into competition with the

agencies and offering service to customers – only to community members. [...]

Are there certain properties of TYPO3 which had to be thought of at the point of initialisation and can't be changed any more?

If you initialize an open source project today – like Magnolia – the initiators and also the developer community have more experience in what communities may look like and how communication might work.

One important thing in the beginning is to think about what license to take. It's not good to do that in a haste. You have to consider precisely what kind of contribution shall come from the community, where do I want to be able to control things and where shall the forces play freely. A license can partly communicate and organize this. You can write your own or you can take an existing one certified by the FSF. The licenses organize at different levels. For example the GPL controls only few things so there must be other instruments installed to enable collaboration. All the marketing advantages of open source software also have their price. The widely distributed development bears the risk that contributions can't be put together again or that powers fritter away. So it's important to think about collaboration and control – which in the end is the construction plan of a community. The problem is once you've decided e.g. for the GPL and later you change to something like the Zope License, many community members might get angry about it and feel exploited by the central company. There are implications for your business model. If you publish under GPL, a community will be likely to build up that helps on many different levels and control is only possible as quality assurance and consequently your business opportunities will be much narrower. Basically there remains the possibility to offer the traditional services like documentation, education and so on.

Most other things can still be changed quite easily later on. Marketing for example is about the same as in every other project. It's good to have it done well, but in the end it's the product quality that counts. [...]

One other thing that should be thought of in the beginning is the basic API. I don't know of projects that didn't lose people when they did major changes in the API. For example imagemagick is such an example. It's a great software but the history of the interface is weird so they've got a bad reputation from that. [...]

What's the license of TYPO3?

GPL.

What are the typical properties of TYPO3? Could you also comment on the programming language PHP, please.

The graphical user interface is one of the features that users like most about TYPO3. Authors, e.g. have the possibility to edit texts inside the front end, so-called "edit while you surf." In addition the precisely configurable user interface of the back end is very popular with customers as it saves time and money when training editors and authors alike. And last but not least the broad availability of functionality is a good argument for TYPO3. Many of the extensions are of very good quality and ready for production – because they weren't developed based on guesses what the users might need but were created based on customer demands. Development is quite easy, mostly because it's in PHP and the extension manager enables a simple and efficient handling of the extensions.

What do you think is the influence of this extension manager for community building?

I believe it's the central factor. Being able to add new functionality at this level without touching the core and also making deployment very easily – with the help of the kickstarter – is very unique. For developers and agencies the most efficient way of customizing is actually the only possibility to distinguish themselves because many service providers offer TYPO3 today.

What are the other properties of TYPO3 that promote community building?

The thorough documentation of the core API and of the different kinds of extending possibilities of TYPO3 had several

impacts. This was important for community building. Dominic Brander presented a nice diagram in Zürich (at the Open Source Content Management Association – OSCOM Conference 2004) showing the workflow for testing your plug-in inside the system. It's a complete IDE (Integrated Development Environment) that also influences project communication.

People work with TYPO3 in many different ways. There are designers who create the templates, developers who write TYPOScript only or some newbies who just do a mapping based on HTML pages. This is a good platform for people with different working methods. TYPO3 can be used in many different ways. Some use it for creating a pre-press system, others manage the website of their organisation. This variety of complexity is very important for the community.

How is development organized?

It's not very formalized. It basically runs like that: Kasper works on all issues only he can implement because they involve core technology. Extensions very close to the kernel are mostly realized by people in the sphere around Kasper. For example the workflow system is done by Christian Jul Jensen, a close friend of Kasper. There are others who offered to help, for example Martin Kutschker who wanted to take care of the typing system, especially UTF-8-compatibility throughout. He got active in several discussions on the developer list and proposed solutions. So Kasper showed him the classes involved and allowed him to take over.

So there is a certain kind of specialisation?

Yes, it works like that. Nobody does everything. Actually the only one who is active in all the parts is Kasper. All the others have found a certain topic and care about that.

How much of the current code of TYPO3 has been written by Kasper?

Difficult to say. Concerning the core I'd estimate at least 90%.

Who has commit access?

Only few people can commit. There are two who are responsible for the release management, Michael Stucki from Basel and Ingmar Schlecht from Hamburg. One of them cares about the Linux distribution and the other one about the Windows stuff. When they receive bug reports by the bug tracker they follow a specific workflow. First they estimate the impact of the problem, whether Kasper has to take a look at it or if it's just a minor issue. Then a solution is proposed and if it works they can commit the bug-fix themselves.

There is also a second SourceForge project.

What do you mean by this?

There are two SourceForge projects about TYPO3.

Do you also do CVS with SourceForge?

Yes. In the second project some parts have been outsourced, e.g. the Database Abstraction Layer, DBAL and other things. These are extensions with a large impact onto the system, having large consequences for it. And there is a maintainer for each of these areas. So the core and these major projects have been split up into two SourceForge accounts.

Do you intend to publish the entire TYPO3 in this project again?

No, just the extensions. Depending on the kind of extension. For example the DAM, the Digital Asset Management, is published in the extension repository only. It consists of different extensions and one of them also replaces some of the core classes. It's created by Rene Fritz, the first one who started to work on the core after Kasper, so he has some credits for doing such central things.

So if I'd like to do core development, how could I get commit access?

There is a very concise description on the Apache Software Foundation website concerning meritocracy. The same holds true for TYPO3. There is still quite some social mobility inside the community. It's not an old boy's club, many people joined only recently but already have considerable authority concerning the project. All entered about the same way:

They proposed improvements and realized them – so they all convinced the people who came and contributed before them by their achievements. One can't expect Kasper to grant access to someone just because he's asking for it, jeopardizing the entire quality. If somebody makes a proposal, demonstrates how it works and what the advantages are, and he is likely, if his solution is logical and fits the overall plan, that he'll get access if he is doing so continuously. Nevertheless, the overall control over the project will remain with Kasper.

How can I program an extension and check it into the repository?

That's no problem at all. Download TYPO3, select the extension manager in the backend, click onto the option "Make new extension" and then code your new extension. When you're done, you can register at typo.org to get an extension key. Then you're able to upload your personal extension into the TYPO3 extension repository. There is no restriction for publication at all, the only limit is the size of the file and even this you can override by asking Kasper or Robert Lemke, responsible for typo3.org, to increase the limit of your extension. So the hurdle is very low, everyone can participate. And you can actually do almost anything. Like extending core classes or overwriting functionalities. The only difference is it's done on a higher level so nothing will be gone or can be destroyed in the core.

What is the influence of this concerning continuous development?

I'm sure that if for some reason Kasper wouldn't be able to do something really necessary in the core somebody will do it as an extension. The advantage is something can be done very quickly. Of course that's not always good since resources might get lost or because people don't want to discuss or stay with the half finished solution. But for customer projects and short term solutions it's ideal to work like that. There's an issue about collaboration. For example if you see how many new extensions there are and how small their differences are you can see how important collaboration actually would be. Not to program five weak ones but to do a single one and plan and discuss what its features shall be. It's something that has to be improved.

I thought this would be a danger of the extension repository but I've realized there is some regulatory system. The problem was recognized by the community itself and so collaboration on this issue started. I would not say it works perfectly all the time but in some cases it does. For example agencies who do professional work for TYPO3 are interested in collaboration because it's a serious business for them. It will improve, the quality isn't as good as it could be but it will be better in the future at least in the basic technologies where companies can't differentiate. Of course there are problems; collaboration is always difficult, especially if you don't know the others personally. So such an extension repository gets spammed. For example recently I saw somebody who definitively meant to do good when he took about ten extensions, modified them minimally, changed their names and icons and committed them again. So these ten extensions weren't necessary, it was rather a finger practice which should not have been published.

These are issues a community has to solve. One reaction is the planned extension review process. There shall be a group of people who'll do a review with strong criteria and rate the extensions. Additionally there will be a user rating so there are two complementary elements which together form some sort of a quality seal. That's the response to the problem we have thought up at present. But it's not easy to implement because highly qualified programmers are necessary for the rating and these guys usually have better things to do than reviewing other peoples' code.

What kind of documentation do you have and what is its influence concerning community building?

Documentation was an important topic from the beginning. Kasper invested a lot of his time into documentation which is definitely unusual for a developer. Whenever he published something new he included a documentation. So there is a reference manual and various different tutorials which be-

came larger with time. There is a starter tutorial and others about template programming. And there is an "Inside TYPO3" document which describes how the API works.

Who wrote them?

All I mentioned so far have been written by Kasper. There is also a tutorial by Kasper and Robert Lemke about content typing. Of the references the TypoScript Reference TSRef is the most important one. It lists all the keywords of the scripting language. This has always been a core document from which a lot of information could be extracted. So on typo3.org there is a documentation repository with all the files, maybe a couple of hundred. There are not only those by Kasper but also by other people, mostly extension developers. You can and should deliver a documentation with each extension based upon an OpenOffice template. These are uploaded into the repository and transformed into HTML for display and thus a small manual website is created. Additionally the page can be viewed as PDF or again as an OpenOffice document.

The documentation always had a large impact on community building. TYPO3 has documentation above average concerning the sheer number of documents. And there was always lots of discussion about the quality. I believe it's quite high although there isn't a guide or a document with a general overview of all the components so far. So far two books have been published and two more will appear in the beginning of next year. And I believe there will be more in the next years because there is no publisher who doesn't want to have a book about TYPO3. We actually get mails from publishers who offer this to us – even we as authors are asked if we didn't want to write another book for another publisher. The one we wrote is surprisingly quite successful. The feedback of the community and agencies is generally positive and they also believe it helps the project. And it's good because this was the reason we did it – since you don't earn money with that.

Next to books there is also something else which is very valuable we have: videos. There are lots of them created by Kasper and others. And they are useful indeed. If you don't get it by watching a video, you'll probably never get it. I just looked into the documentation repository: there are about 381 files totalling 4186 pages – so that's quite a lot.

What would you say is the most important kind of documentation for a starting project?

The installation manual. There is no more important document and there are always lots of questions about it. Concerning documentation I believe you have to take it really serious as an important task. You shouldn't think the mailing list archive is enough. The knowledge in there must be transferred into FAQs, must be rephrased and made searchable. Then you see the weaknesses of your documentation. And you must be aware, even in a project like TYPO3 that has an enormous amount of documentation, of the questions are still asked. So you have to think about what you did wrong and correct these issues. We found out it was probably a bad idea not to create, say four main documents, each of them covering one perspective. One would be the installation manual, another one about template programming, thirdly one about administration and configuration, a fourth one about extension programming and maybe a few more. So for every important area there should be one central document. We should have done this in advance, thinking about what parts of knowledge are necessary to work with TYPO3, making one single document for each, and that's it. Instead we had a proliferating number of documents. All the tutorials and extras can't replace a manual, it's just something different.

What is the influence of the type of open source license on community building?

I can't really judge it from a birds eye perspective. I always thought it has consequences if somebody chooses the GPL compared to somebody writing his own license. Users would think there must be something strange about the project if it has its own license. But I think I was wrong. Most people actually using open source software are developers so they just get the code of the software and that's it. They don't care at all what's written in the licenses. There are cases where the

license is very restrictive but these usually aren't real open source licenses either. Jahia is such an example. I believe if you use it commercially then a licensing fee has to be paid. I wouldn't call that an open source license because it's a handicap for distribution, no doubt. But with GPL it doesn't matter. Everybody knows it. If you take an Apache license or write your own one like Zope or Apple doesn't have a large effect on the spreading of the software. I always thought it would be different but I can't see the effect now.

For every license there are questions. Strangely we get a lot of questions about the GPL and discussions. Once we tried to answer them with the help of an IP lawyer's office in Munich who gave a lot of responses. But this really had the opposite effect because afterwards there were even more questions about legal issues. It's just difficult for most of the people to think in these juridical terms – which is just normal. But also if you look at the GPL closely there are a lot of relations to other legal terms like the copyright protection which of course is still intact. A license can't omit all the other law constructs saying "With open source you can do what you want." So there are always questions about details. I get at least two of such questions a day like "May I delete the name of TYPO3?" Or "somebody installed TYPO3 and deleted the copyright note in the login screen. Is this allowed?" We once had a discussion with Alain Cox and he asked Richard Stallman. In one point the GPL is somewhat misleading. So Alain Cox asked Richard Stallman and they had to admit to be a bit clearer in version 3 of the GPL. "We actually didn't think of web applications."

These are points where you leave the developers corner and get into legal issues. That's difficult because there are also different geographical areas with different juridical environments. For example in Germany the regulatory frameworks are simply different from those in the USA or other countries. These are all topics you have to be aware of. It's not just done saying "I'll put a GPL sticker on it and then you can do what you want." I believe it's good to look at other licenses, even self-written ones – even if in some case the FSF (Free Software Foundation) doesn't accept the license. But when it comes to the acceptance of your software, in the end people don't care much about the FSF. They just get the software no matter under what open source license.

What kind of organisational entities do you have and what influence do they have concerning community building?

There were a lot of attempts at organising the community. Once it was tried to build teams for all different topics. And we must admit this failed miserably. Many people reported for the positions but in the end they didn't have the time to adopt the topics. The only thing that really exists is some loosely defined entourage of people around Kasper. Depending how tight you draw the circle there are more or less people inside. Looking at the intensity of communication there are about four or five persons including Kasper and me. It's going to change now somewhat because of the founding of the TYPO3 Association. By defining responsibilities it gets clearer. Also project groups with certain tasks will be formed. Nevertheless, just by formalizing things it's not guaranteed that everything will work. The main problem remains: people have to invest time. So one of the ideas of the association is to pay for certain work where no volunteers can be found.

What's the main idea of the association?

The main idea is to raise money to pay the core developers of TYPO3 to continue their work independently of actual client projects of their own. So to contribute something you just become a member of the association and pay your annual fee. Membership is possible for individuals and companies. [...]

What's the right moment to found such an organisation?

I don't know. It's actually too early to say for TYPO3 because it's not yet founded so I don't know the consequences yet. I believe we could and should have founded it earlier – but this I'm saying because I hope the association will solve certain problems. So if time shows I'm wrong my answer is no longer valid.

Are there other problems to solve?

As I said one issue is "Glasnost." It shall make structures more transparent, for example how we decide about the priority of projects. We only have one Kasper and one Jule so on what project shall they work on first? Such things will be discussed openly in a group of well-known people and they will also decide about it. In this way everybody benefits because it's better than to decide about such things on a secret mailing list as it happens at the moment. We don't have any advantages from organising these things in a small circle of people. The opposite is true: If people know what's going on and what issues are discussed they can see where they want to participate.

Do you also see any risks in founding such an institution?

If you do such a thing there will always be criticism. People might say such an organisation just consumes resources and produces overhead – writing a protocol instead of coding. Another topic of criticism might be the existence of a head of board. Maybe that's just too formal for some. On the other hand there will be the possibility of election for the first time; that's an advantage. I believe it's good to have some democratic principles and to receive confirmation for one's work, to see if what I'm doing is good or not. [...] Also it wouldn't be easy to disband the association. Because once you have 100 members but you realize you can't do with the association what you wanted to achieve, it gets difficult to close the association again when its members are very happy about it.

Concerning the organisational structure of an open source project, what points of coordination are really important to have?

I believe it's important to have somebody who thinks and communicates orderly, somebody who thinks in advance what the website needs – because this is somewhat the heart of the community. What information do I need to communicate to the community? And it's important to take this really seriously. Looking at a project like Bricolage the guy is doing this really well. Although I don't participate in the project I see the guy – it's more or less a one man show – always makes a release and writes an email when he does anything new. He's doing this consequently and brings it to a point, just writing what's new and what are the tasks. I believe it's enormously important to take communication really seriously. So if somebody as a developer is not able to do this – which is absolutely ok – he must find somebody who can do this and also likes to do it. Otherwise the whole thing doesn't serve much, it just stays on the technical level and doesn't get distributed. Neither will there be professionalising concerning building a community which helps and works together. The less information you make available in a bundled form the less people will be able to collaborate.

What is the role of the release manager in this context?

It's extremely important concerning quality assurance. It doesn't work without him. Everybody could just publish something that doesn't work. [...] You only realize the advantage of an open source system when all the bug fixing and other contributions of the community gets integrated.

What are the tasks of the release management?

Quality assurance, communicating what's new and maintenance of the change log.

Do you have an explicit credit system?

No.

How does the community communicate?

We use mailing lists which at the same time are news groups, too. There are lots of them and there's also a lot of traffic – I believe over one million emails a month. Most of the readers aren't subscribers but just go to the news groups. There are two main mailing lists, the English mailing list and the developers' mailing list. Additionally there are installation, marketing and user groups' mailing lists – all countries and many cities have their own. [...] Everything that has some significance will go via the developers' mailing list. First aid and political matters are discussed in the English mailing list, for example questions about licensing. There is a German list

but it's not an official one. And there is also an announcement list where a few persons – Kasper, the two release managers and I – can send news. I send announcements about new versions, press releases and so on. The release managers only do the bug fixing announcements. Once there was a community newsletter but unfortunately the editorial team did not get integrated well enough. We'll have to see if it gets revived. I think once the association is created there will be new reporting duties introduced. Then there are some very specific mailing lists, for example the founders of the association use a closed one.

Do you have physical meetings?

Yes of course. First there are small events for the people around Kasper, about once a year. This year we were only six persons when we met in Lüneburg. Last year there was a larger meeting because of a specific task. It's not a public meeting, people get invited.

And then there is the snowboard tour. Also there will be the first conference next year. We don't know yet when, either May or September. It's in Karlsruhe, the first international TYPO3 conference!

What do you expect from this event?

We hope to relieve the snowboard tour. The need for a conference grew a lot because of the success of the project. So many people come to the snowboard tour hoping to find a conference which is not possible to offer.

What were the consequences of the first snowboard tour?

The greatest effect, as Kaspers says, is "adding faces to emails." A new level of trust arises between people who previously met only on the mailing list. After these snowboard tours new projects get started and collaboration as a whole is better. For example on the last snowboard tour there was the discussion about translating the typo3.com website into German. The issue showed up again in the marketing mailing list so an agency in Berlin called about 20 other agencies and collected between 100-300 euros from each of them. Thus they could pay a translator and the site got translated very quickly. There were long discussions before but nothing had changed.

There is a difference if you met once and drank a beer together. It's just easier to team up.

Do you also see some disadvantages of such meetings?

No, I don't think so. [...]

How about infrastructure? You have a website and you use CVS from SourceForge. Where do you host the website?

There was always a company, Netfielders at Düsseldorf which hosted the websites and mailing lists. Then there is a Danish company (sunsite.dk) and SourceForge for downloads.

What are the advantages and disadvantages of a platform like SourceForge?

Well there is a lot of competition. There are also Tigris and others. The main disadvantage of SourceForge is its slowness and the lack of customization. It still covers a lot of the needs and also many people search there for a system. [...]

What is the definition of a healthy community?

I think the tone is very important. There must be people around who take care of the way how people deal with each other. Communication has to stay factual, with humour but not too much. People have to follow the rules like first using the FAQ and searching for an answer in the mailing list archives. Since the community is mostly defined by the mailing list the things important for mailing lists are also important for a healthy community.

It's also important to raise the level of communication on the mailing list so it gets interesting for people with lots of knowledge to contribute. [...]

Where did the community, excluding Kasper, contribute most to the project?

Definitely in the extensions. There were some donations but most of the activity is in the extension area. Maybe we don't communicate clearly enough how to participate or the tasks

are too big.

Do you use a task or todo list?

Yes. And people actually sign up for certain tasks so they get the information where to start. Sometimes they do it, sometimes they don't. Mostly it's done out of a mood. Somebody likes the system and wants to contribute something but in the end it gets drowned in the daily business.

As founder of a community you have to protect yourself. If you count on people's contributions and thankfulness you'll fail for sure. You have to do an open source project for yourself, not counting on outside rewards. If there are acknowledgements that's nice, but you shouldn't be depending on them.

Where can non-developers contribute?

The most important task is communication. There is always a need to write texts, to order existing resources, do ratings and so on. Working on the website typo3.org for the developers or typo3.com for business people, for example examining mailing lists and extracting FAQs out of them or ordering the documentation and eliminating errors. Such a project consists largely of communication so everybody can contribute.

What characteristics do such people have to have?

To work independently is almost the most important capability such people need to have. You don't sit together in an office and usually you can't meet; so people have to be able to work on their own. [...] Kasper calls this "self-starting fireworks": somebody makes a proposal, writes down the details and is still open for criticism. Others look at the draft, propose improvements and the initiator implements them or defends his own propositions. In the end he has to be motivated to do everything all by himself without expecting Kasper to be extremely thankful or some other positive attention of the community.

What are the leadership skills of Kasper which made the community grow so big?

I think it's not really leadership which caused it to grow so big. From my perspective Kasper has two extraordinary skills. The first one is the impressive ability to concentrate. You can discuss with Kasper from eight o'clock in the morning until ten o'clock in the evening without getting the impression of him getting tired. [...] Thus he produces things above the average and with consequences. So maybe somebody proposes something, Kasper works on it in silence for weeks and then shows up with a result anyone else would have produced in no less than three months. I believe this is the secret of his success.

So it's not his way of communication?

No, it's mainly his work. It's the same in about every open source project. If the product is not good, you can communicate as much as you want, it doesn't help. Of course both must be present. If you want to grow you have to communicate well. But you can't sell bad stuff.

Are there people you don't want to have in the community?

Yes, there are always such people. It's almost the nature of an open source project to attract freeriders. There are many people who use it, earn money with it but don't contribute anything. That's usually also the majority. And there are times when you get frustrated about this fact. It's especially not funny when the other one acts relatively aggressively. For example an agency realizes a lot of projects with TYPO3 and creates new functionality along with them. And still after getting the investment back they are not willing to publish the additional code e.g. as an extension. [...] There are a lot of agencies who save hundreds of thousands on licensing fees but still don't donate anything. In some respect I also think that's just stupid. The involvement in such a system is also some sort of long term egotism. You want the project to go on existing so it's very stupid not to do anything for it. Everybody can say "I know TYPO3 and offer services for it." because there is no control about it. For example at a booth at the SYSTEMS trade fair somebody once came and told us they were looking for an agency to implement TYPO3 for them. Only the third one they contracted could meet their ex-

expectations although all of them claimed to be working with TYPO3. These are things that a community has to organize. Others have installed such a control of service quality from the beginning.

Where do you see entry barriers to TYPO3 today?

You have to get into it quite deeply because a lot is available already. I believe it's much easier to get into a project where not much has happened so far. There are still things to do on a relatively low level. It's different in TYPO3. Probably no matter what you want to do there are always at least attempts or beginnings to solve the same problem by others. It's really difficult to have a new idea. So if you have a new idea that's great, but if not, you need to at least look at the existing attempts and work with them. So participation needs more.

Concerning marketing: How do people find the project?

TYPO3 is quite well represented in the press. Then it's present in all kinds of directories like SourceForge, Hotscripts, CMS Matrix of OSCOM and so on. These play important roles. But the major part is probably done by the agencies doing marketing in search of new customers. I think that's the marketing channel number one.

How can new projects make themselves known?

There are always these show case events like LinuxTag or LOTS in Switzerland. These things are always very valuable. Also it's great if you get onto one of the magazine CD-ROMs of c't or something like that. It's always good to do press related work. I don't know how important such directories are to start. I believe it usually goes via magazines and events.

What's the vision of your project? How can it still evolve?

That's difficult to say. I think there are about as many visions as there are community members and also the expectations are different. So I can only talk for myself, but I believe I know approximately what Kasper wishes.

What does he wish?

He wishes money and autonomy to concentrate on development. But Kasper also wants to keep his way of working, especially this intuitive and creative part. He's not an engineer who first makes exact plans and then builds everything according to them. He's really creative, thinks about an issue and then gets into some flush of work. This is something that has to be protected. I look at it as a challenge for the community to preserve an environment where TYPO3 keeps the continuity of development and also uses and protects the talents of Kasper. These are the things that have to be synchronized – and it's not very easy. The community has to offer a nutritive soil for this. [...]

On the technical side TYPO3 has to agree to standards better. And maybe one day the project can even define a standard. Let's say for example the extension repository where the core

of TYPO3 itself is already an extension. Commercial vendors of CMS say the standard functionality of such systems will some day just follow a standard based on some open source software. So it would be great if this compatibility would be drawn much further, for example if TYPO3 would support the JSR-170 standard. On the other hand it might support the Portlet standard on the frontend side. There the ideas and concept in TYPO3 are about on the same level as other known CMS so it's a large potential for TYPO3 as for other CMS for optimization. In maybe five years the intelligence might go way beyond today's CMS. [...]

What made TYPO3 so successful and is also true for other open source projects?

A good product. A consequent and serious communication on the level of marketing, documentation and coordination. [...] I know of other people in other communities who looked specifically at the extension system and wanted to copy it. Other things they didn't intend to copy. I don't think when Kasper created TYPO3 he imagined what has come out in the end. He just didn't dare thinking of such a big success. Maybe this was a mistake. Maybe if you're doing such a project you sometimes have to indulge in the luxury of imagining what the end should look like if the project is successful.

What are the do's and don'ts for project managers like you and Kasper?

You have to obey some rules you set yourself. At least this is what Kasper is doing. There is Kasper's ideal of "completeness." He doesn't just throw something at the world. He develops something, writes documentation and does lots of other things and only when all the dimensions of the package are covered and it can be presented, he actually publishes it. I believe this is extremely important. This is what makes frameworks doomed so often. As Patrice Bertrand said at OSCOM, frameworks actually are almost always the consequence of the developer not daring to say what the users actually wanted. So this way of completing things, offering quality on all levels is important to do and maintain to create some stability in the spirit of a community.

An important don't is not to take things too personally. You can follow a personal style like being friendly and polite. But if things are growing it's rather a handicap if there are family pictures on the website. It's especially an issue for individuals.

Another do: If you want to use open source as a vehicle to make money you have to state this clearly. You have to say exactly whereby and how you want to earn money. In areas where this is not the case and you intend to work together and build a community you also have to communicate this clearly. It's commercial where it is labelled and non-commercial where it is not. If you mix these up you'll lose credibility and respect.

Interview with Bård Farstad, eZ publish, on 3 December, 2004

First I'd like to ask you about yourself; what is your education, how you got to eZ publish and what's your function there right now.

I have a degree in software engineering. Actually while we were studying we started a company, me, my brother and a friend of mine. So we started in 1999. That was actually my first job. My role is software developer and I'm one of the main designers behind eZ publish, me and Amos. I also led development of eZ publish for the past five years. Now I've got another job: I'm in charge of the open source relations as we call it. So I'm working a lot now with the community and basically getting the message out to the people.

Can you tell something about the history of the project?

We started in 1999 and eZ publish started just around then. I was very interested in web development and Amos was interested in GUI development. We did C++ GUI applications in QT; that's what we actually started out with. But we got a customer for a web project in the early days and we thought

that this was a lot of fun. So we decided to build a CMS and do it open source. From the beginning we have programmed eZ publish as a general CMS, not for one customer only but as general as possible. The first version was more of a CMS to publish articles, products and so on. But it was kind of hard to customize. So in 2001 or 2002 we started the rewriting process. When we started eZ publish 3 we dumped all the code - about 200'000 lines of PHP code we just flushed down the drain. We started the project from scratch again with a different goal. We wanted to make a web application framework to produce general web applications and we wanted to have the CMS on top of that. That is the eZ publish 3 series you have today. It is now in version 3.5 and in terms of numbers it has been downloaded more than 1.1 million times from our website. Of course it has been distributed on other sites and on CD-ROM. At least this are the numbers we have.

Actually it has been two projects: eZ publish 1 and 2 are the first generation, and then eZ publish 3 is the second genera-

tion which is much more general. It's an application development framework for web applications.

How did the community evolve?

We had quite a good feedback from the community all the time. A bit special about the eZ publish community is that we have a very professional community. You don't find this in many other projects. When you go to our forums you'll see most of the users actually are working in companies doing eZ publish. They base all their living on customizing, installing and integrating eZ publish for their customers. We have a large community but it is also a very professional community. It is not so much one person just doing it in his spare time, it's rather companies that are involved in our community. That's quite different from other communities.

What was the reason you published it under an open source license?

When we started the company we didn't know what products we were going to make. We had the philosophy that the software we make should be open and it should be easy and it should be stable. With open we mean follow open standards if they existed. We use open source. That has been one of the main corner stones of our company since we started so it was totally natural to go open source. That was kind of the business we are in. We didn't know that we're going to do a CMS.

How did you publish it?

We started out a bit like any other project. We also had a page on Sourceforge but that hasn't been maintained for many years. It started out on Freshmeat and some other sites. One of the first sites we also published it was Hotscripts. If you go there and look at the rating system... There are about 20'000 projects on Hotscripts. For about the last three or four years eZ publish has been the most popular project, the most viewed, the most downloaded and the most clicked on. That has been one of the channels where we became known. I think they were Freshmeat, Hotscripts and some other smaller site.

What's the relevance of these platforms concerning community building?

I don't think it has too much to do with community building. They help with marketing and make announcements about new releases. But the community comes from other things like the forum and comments on articles and so on. That's where you create a community. Also we have a conference every year so I actually meet people in real life. That's where the community is, at least the hard core.

The whole point of those channels is to get the word out. [...] Freshmeat is not very specific, the users there are not looking specifically for web applications, especially not CMS. If you have more targeted sites like for example Hotscripts then you reach the audience in a better way. [...]

What are the advantages and disadvantages that your project was initiated by a company?

The advantage is you get consistency. People are dedicated for development over time. You are working towards a goal and you work very focussed on that goal. That is the main advantage for a user. Also if a company is behind the project you get the possibility of getting the professional services and support you need, especially if you base your business on this solution. And you get the comfort to know in a month there will be a new release containing those features. You know what's going on. If there isn't a company behind it things are only accomplished if somebody wants to have them done, otherwise they don't get done. That's the philosophy behind most of the open source projects. So if you do it, it gets done, if you don't do it, it doesn't get done.

So users can tell you what they want and you develop it?

Yes, basically. Of course you can't send an email saying we want this system and expect us to make it. But what we do before every release we ask what are the features you'd like to see in the next version, what are the things we should work on. We take all that feedback and we see what are the most requested features. We do that in more detail every year at

the conference. There we discuss the next releases and what we should work on. Specifically, what are solutions we could implement, and so on. That is the real power of the community because you get so much feedback on things you could do. If there is a bug they'll let you know immediately. Disadvantages for a company I don't know. It must be that a company needs to make money for a living, of course. But I don't necessarily see that as a disadvantage. It just means there are professional options available. It is open source so you have the freedom of choice.

Concerning community building is there a difference if a developer sees a CMS from a company compared to a CMS from an individual programmer?

I think we and MySQL are about in the same situation because we have the full copyright of the software. As an external developer you can't say I want to join the project and change this and that, because we do have all the copyright, because we have the dual licensing policy. This can be a disadvantage. Still we have external developers working on eZ publish as well but then they have to give the copyright to us. So to address that problem we have made eZ publish to be a framework. So it is designed to be extended. Normally you can just write an extension or a plug-in or a module that solves your problem. That's what third party developers mostly work on. Of course it is not exactly the same as a traditional pure open source project since we have full control over the software, all the commits and so on.

Concerning initialisation again: What can still be changed in the project and what can't?

We can change anything we want. We are in a position to do that. We have already done it once, from version 2 to version 3 we changed everything and actually just kept the name. [...] Usually we just improve things. We have issues and we work on them for say the next four months. [...]

Which technical characteristics promote community building?

As I mentioned the plug-ins and the extension system are of course part of it. In about 20 or 30 lines of PHP code you have an extension to eZ publish like for example a template operator or a small module. This promotes third-party involvement in the community. [...]

What's the influence of the programming language?

We chose to use PHP. It is a language which is very simple to use and the learning curve of PHP is quite low. Also looking at the worldwide hosting providers it's the most used language as well. As far as I know it is the most used programming language for web applications. That's good because it means you have a lot of potential users and contributors for your community. If we had used another programming language which is more obscure, harder to learn and not so widespread, that would have changed the whole potential for the user base. So PHP is a very good language for this purpose, at least this is our experience.

Do you know of any other factors concerning the software which promote community building?

We had a community project going on for writing documentation. This is a large part when you have a large project. We had a lot of contributions in the documentation area. Users wrote a lot of documentation for eZ publish. [...] You just can log in on our webpage and change the documentation, it's Wiki style. That is a point which helps building the community because then people can contribute without even writing code. I already mentioned it: bug reporting is also one of the best feedbacks you can get from the community, it also involves people. All this together builds the community: the forums, the bug reporting, the documentation and contributing code to the public extension repository.

How do you coordinate software development? Do you do everything in your company or do you allow people to have commit access?

Most of the developers come from eZ systems. But we have given commit access to some developers. This doesn't mean they can change everything. We have a mailing list where

you need the feature to get approved if you are an external developer. Internally we work on a project basis. We have [bond] release which are about every four or five months. We write a specification what should be in that release so we have a development process, testing and a release cycle. So we do software development on a formal level and try to coordinate this with the best of open source as well. [...] We do allow external commits of small functionality during the development cycle. [...]

How much code is from eZ systems compared to contributions from outside?

It would be 99.9% from eZ systems. In terms of code, almost everything is written by eZ systems. But most of the bug reporting and the activities on the forum are external. If you ask MySQL they'll say the same. It's very valuable. So if a user reports a bug that is reproducible it saves us a lot of time both finding the bug and fixing it. We can just follow the steps and fix it. That is the biggest value you get from the community. It's not just the code that has value.

If I'd like to get commit access to the SVN repository how do I have to proceed?

The first thing you need to do is to send us an email saying who you are and what you intend to do. Then we'll ask if you have used eZ publish before so you need to show us you know the system. If we feel confident about that we'll open an account for you. But you have to sign a contract saying all the code you commit will be copyrighted to the company. That is the way we do some of our business, selling professional licenses on the code. So we need the copyright of all the code. [...]

Besides writing documentation and reporting bugs, are there other tasks done by the community?

You can help spread the word which is part of every community. And our system is also multilingual so you can do translations. [...] We've gotten about 20 translations, all of which are contributed by the community except the Norwegian and the English one. [...] Most of the community activity happens in the forum. New members can ask questions and advanced members can answer them and help people. [...] You can write extensions, of course. [...]

What do you have to consider when you plan on building an international community? What do you get from this? What's special about it?

First you need to speak a common language, normally that's English. [...] So by having an English community some people won't join because they don't speak the language. [...] To solve the problem, you could create an English, a German, a French forum and so on. But this means some people will just discuss in the German forum so you'll lose those users for the international forum. It's a two headed horse. We haven't yet opened any language-specific forum other than in English. So all communication in our community is in English. [...]

What else besides communication do you have to consider regarding internationalisation?

[...] The software has to be general enough to be able to be translated and localized into other languages like Chinese and Arabic. This is specifically important for CMS. So the design of the software is critical if you want to have an international community. It doesn't help if they can discuss the software on the forums if they can't use it in their native language.

How many international members do you have in your community?

What I can tell you: We have 17'000 registered users in our forum. We have visitors from all over the world, about 150 countries every month. [...] We have paying customers in 54 countries. [...] There is only a small part of the traffic, about 4 %, coming from Norway. US commercial is the biggest. Then we have the network, like .net. But most users come from Europe. [...]

What's the special thing you get from this international community compared to a Norwegian-only one?

You'll definitely get the feedback on use cases and set-ups which is not normal for our customers in Norway. [...] So you get feedback from many different cultures.

What influence does documentation have concerning community building?

[...] There exists already a book about eZ publish and more are in the pipeline. Documentation is crucial. People take responsibility. [...]

Concerning the dual licensing model: how does it work and what's the influence on community building?

It's very simple: eZ publish is GPL, so basically you can do whatever you want as long you follow the GPL. But you cannot make distributions, sell it, or make changes and not give back the changes as open source. So for example if you are a company doing system integration and you develop a component which you want to sell then you can buy a professional license and then you can make business on top of eZ publish. But it is exactly the same software, we haven't held back any features in the GPL version. The only difference is, if you have a professional license you can rebrand it and resell it without concerning the restrictions of the GPL. I don't know how it influences the community. But I know it attracts companies who want to earn money on the solution. And most of those companies join the community as well. That's what we see, there are many professionals who work with eZ publish. This is also part of our profile, we want to make other businesses successful with their products based on eZ publish. I don't think it hurts the community building at all. It just gets new members into your community.

Did you ever get negative feedback about the dual license strategy?

No, nothing. I mean we are 100% GPL. But we also have another license. This means you can choose, basically. And if you choose GPL you don't even think about the professional license. It is basically to give the users another option because professional users need this option to make money from it.

Do you offer some sort of partnership for other companies?

Yes, it's part of our business strategy. On our website we have a partner program for different partners. We have solution partners, you might call them consultancy partners who build and integrate websites based on eZ publish. Then we have software partners. Typically these are the ones with the professional software license because they create and sell software based on eZ publish. And of course we have hosting partners. [...]

Are there any individuals who use eZ publish?

Yes, we do have some people who use it for their personal use. These are mostly technical people. They must not be very technical, just being able to install and customize it. [...]

Concerning the organisational structure of the project, are there other ways to promote community building apart from the partnership program?

[...] One thing we're working on now is to get user groups started. Currently we're starting a user group here in Oslo. We are promoting users who want to start an eZ publish user group. Also we offer to organize third party talks so for example if they want somebody from eZ systems to talk at their user group meeting we can probably arrange that. We can also help out in promotion and hosting of their user group website. There are some user groups on the way. [...]

The first step is to increase the number of people who know about eZ publish. So we write articles and have them published in different forums and magazines. Also we are active at the different open source conferences and go to trade shows. Basically we travel the world to meet people. [...]

How is the release management organized?

It's just a standard development process. We make up a feature list and then we spend the next four months developing these features. When they are implemented, then there is a feature freeze, testing and then you release it. [...] If a third party has a feature he wants then we'll implement it if it's not completely out of the way of what we're doing. We just have

the main goal and that's what we're working on during four or five months. [...]

What's the influence of version numbering?

We have major releases but this happened only 3 times. One was the total rewrite of the system. For the dot releases we can have large changes. Like from 3.4 to 3.5 we changed the whole administration interface. Between such version steps you see added and improved features but from 3.4.1 to 3.4.2 you only see bug fixes and security fixes and translations. When we get to 4.0 then we'll have full PHP 5 support because there isn't any at the moment.

Do you use some sort of a task list where the community members can see what they could do?

No, we actually don't do that. We do have a bug list where you can come with suggestions, bugs and enhancements. We don't have a public task list but of course an internal one. It's because 99.9% of the code is written inside our company. Of course it's something we could do to improve the community building. That's an idea!

Do you have some sort of an explicit credit system where you write down the names of external developers who helped with the code?

Yes, we actually do that. Let's take the documentation for example because that is where we get most of the contributions. If you go into the documentation and click on a page you'll have on the right hand side a list of all the users that have contributed to that page. [...] In the code we have the change log and a credits file.

What's the influence of mentioning the names?

For example the translations page is a very visible page. You see a map of the whole world and a list of all the translations. And of course the names of the persons are listed so they get credit for what they have done including a link to their web page. I think this is important. As a contributor you feel like my contribution is appreciated.

Do you have developers who write extensions and check them back in somewhere?

Yes, we call it contributions because we have many different kinds of extensions like workflows, template extensions, data types and applications. So we have like different categories. We have several extensions in our contributions area. There all the extensions get their own page with comments and descriptions and of course again the name of the author so he gets the credit.

Do those developers need to purchase the professional license?

No, that's totally GPL.

So only if you want to sell the extension you have to buy the CMS?

Yes, most of the extensions are GPL. But we also have a partner shop where all the partners can present their commercial extensions. Typically it's an integration with an ERP system or other larger things.

So you give your partners an opportunity to promote their extensions?

Yes, we offer our partners to sell their extensions over our website. That's our philosophy. You as a partner should be able to make money because this is what you basically want to do as a company. That's what we're trying to do. At the same time we have the openness of the GPL version and the community. We want to support both worlds.

How do you communicate? You said you have the conference. Do you have other physical meetings?

Yes, there is one official eZ publish conference in Norway. And we are at many other conferences many times a year like the PHP conference, the OSCOM and others. So we meet a lot of people all the time. Now that we're starting an office in Germany we will most likely start a German conference as well.

What's the importance of such personal relationships?

That really makes a difference when you discuss something

in a personal forum. I mean I did this for several years and suddenly you meet them in person - that is totally different to discuss after you've met them. Of course you don't need to do that but it really helps.

How do you communicate digitally? What communication channels do you use?

We mostly communicate through our forum and through email. We also use Jabber and IRC.

Do you have different mailing lists?

We have a few mailing lists. We have a developer mailing list, a partner mailing list - of course you need to be a partner to be member of that. We actually try to make most of our communication in our forum on the web page. That's our primary channel.

Concerning infrastructure. Do you host everything on your own servers or do you use any platform like SourceForge or Freshmeat for development?

We host everything ourselves. We need this because of the freedom. Once you grow up over a certain level you just need this freedom to control everything the way you want. [...] There are limitations, especially if you have lots of traffic and many downloads and so on. Still it is good as supplement or an addition. For us, for example SourceForge just wouldn't work. Especially not since we're also a professional company and sell services. This is not allowed there.

How would you define a good and healthy community?

Basically if you can ask a question and get a good, relevant answer then you have a good community. And also if you can discuss future features and improvements. Another point is to have the same people not only for one week but for several years. That's what we have, members who've been around for three or four years.

What other properties should the community members have?

I like all the community members. Of course we have all different types of members in our community. Some are just testing it for a few days and trying to install it for the first time. Others are experienced and maybe have been there since the start. It's not that we would prefer one in front of the other. [...]

Are there actions of community members you don't like?

Well, people who are not nice. But we've had very little problems with that. If for example somebody says something bad in the forum about the software we don't go in there and change it. We don't censor the forum. Basically people who don't follow the etiquette, for example having 1000 exclamation marks in their subject. Then I think I really don't want to answer this message. Sometimes they just don't know how to behave. So that's what we have moderators for who get rid of all the exclamation marks. [...] We have external moderators, too. They get their own icon and get promoted as moderators in our forum.

Are there any entry barriers to get involved with the eZ publish community?

Of course you need to get the software running. If you want to join, just write to the forum. Of course we also have a lot of passive members. But it's easy, if you want to answer questions in the forum just go ahead. So basically you need to have a usage for the software and then you need to create an account on our website which requires an email address.

What leadership properties are helpful for community building?

You need to be interested in helping people out, to get them started. We have the philosophy "If you recruit one person to the forum he will probably recruit one or two more." So it's really worth the time. [...] But it's actually not us who are the most active there. In a community, people like to take responsibility. [...] That's the power of the community. [...] I don't think it's really connected to a specific person at all. [...] Of course the most active members are all native English speakers.

What's the influence of knowing English very well?

That's important because people understand more easily what you are saying. [...] I don't think it's that important, you just need a certain level. You don't need too much language skills to join the community at all. [...]

Why is your project so successful?

It has much to do with how we made the software but also has something to do with the timing. [...] When we started in

1999/2000 everybody went bankrupt. That's the time when people look for alternative solutions, when they can't afford a high end solution. Open source is just a natural place to look because the price tag is zero. [...] The price tag does not necessarily reflect the features and quality of the software. So it has to do with two things, timing and how we made software. I believe these are the key factors of our success. [...]

Interview with Gregor Rothfuss, Xaraya on December 1 and 7, 2004

What is your education and how did you get into Xaraya (or PostNuke as it was called then)?

I studied business administration and computer science at the University of Zürich. Besides this I had some websites and was looking for a content management system for them about in 2001. First I found PHPNuke but it wasn't that good. Some people improved it and founded PostNuke. At that time I was abroad in an internship so I had time to participate in PostNuke. I developed some stuff I needed and did bug fixes. It quickly got very popular so I didn't develop much more but helped others to get into the project. I worked more in the background talking about on what features to focus and so on.

What made you decide for PostNuke?

It's got many features and a good community. Before PHPNuke I tried another one called PHPWebsite, I think. But this never advanced even when I sent patches. In PostNuke there was much more going on.

In what other open source projects do you participate?

In Apache Lenya because I also work at Wyona, the company that initiated Lenya. And besides this and OSCOM (Open Source Content Management Association) I don't have much time left. I used to work for some sort of ViewCVS once. But I realized that it's lots of work so I had to quit some activities.

What do you do in Xaraya at the moment?

Actually nothing. But just recently I had a look at it again and it's changed a lot. One thing that made me decide not to work for Xaraya so much any more was that I believed the project had some sort of a perfectionism problem. For almost two years they're working on a release 1.0 that hasn't come out yet. The last mile is some sort of a death walk that never ends.

So how would you suggest to proceed?

Long ago I've realized that there is a "time to market" for every product, some sort of a time window where there is a chance to succeed. After this there will be other projects that do similar things and bind their community to them. So when I saw others in the community were more interested in doing it perfectly rather than having a large community I got demotivated.

Are there really things that release 1.0 still needs or is just the version number not yet 1.0?

I mean there are always things to improve but I believe by now it's just hair splitting.

So why doesn't just somebody do these last few things and publish it?

Because this list of last few things always changes. The bar is always put higher and higher and improvements raise the needs for again other improvements.

So there wasn't a feature freeze?

In a way there was but nobody wanted to stand in the way of somebody who was doing something - like in many other open source projects. If some people are checking in new features all the time it's very difficult to say stop. But these features have to be tested again and have to be mentioned in the documentation.

In general, what is the influence of release management on an open source project?

Mozilla and Firefox, for example, have a very good release management. They announce the dates of their releases and tell the community that what's in there on these dates will be in the release and everything else will not. That's the way to avoid discussions about which features to include and which not. This always leads to delays. And projects which seldom do a release are not viewed as healthy because from the outside it seems like nothing happens.

What release cycles would you suggest for a project like Xaraya?

Maybe every two months about 0.1 higher. Well, this might be too much, maybe only 0.0.1 further. Doesn't actually matter how much. Some projects have different branches. I believe it's good to have a maintenance release maybe every two months. Every month would be perfect but people do this besides many other things so they don't have time for it continuously.

Could there also be problems if you'd release too often?

Yes, definitely. Because you wouldn't get enough feedback and also there's always some overhead to do a release. So if there is too little time between releases to develop new features or do bug fixing the delta will be too small. Additionally, it's hard for people who use the software productively because they would see that their release is out of date all the time and that they might not get any support for it any more.

How about API changes? What do you have to keep in mind before doing one?

It's a controversial issue because there are opposing interests. Users usually never want API changes but developers often would like to label something as deprecated quite quickly. For example users of Log4j even protest if something is changed after two years.

One problem in open source projects compared to closed source software is the difficulty to distinguish between external and internal API. Many projects don't define exactly which parts of the software others shouldn't take for granted. This makes it hard to do changes. Developers might think of certain parts as internal features and assume nobody will build upon them. But since people can see inside the code and it's not well defined - especially in web applications where it's difficult to distinguish - problems will result.

How would you suggest to prevent this?

Some projects have a policy where they define how much time has to pass until something is called deprecated. When these promises are kept they also get the confidence of the users. The other step is to document with JavaDoc or PHPDoc which methods are internal or external. The external methods are under the control of the policy and the others are not. This is quite simple for APIs. With other things like naming conventions or specific files it's more difficult to prevent conflicts.

So you would suggest to fix this in writing, too?

Well, writing down too much would kill everything. Nobody would want an ISO 9001 process for this. So younger projects with not so many users usually don't have these deprecation policies. It's more important for older ones that are used extensively, like for example the Apache Web Server. For years now everything is clear there. These are signs of the maturity of a project.

Would you say in projects like the Apache Web Server there

is no more interest to build new features or to let the community grow?

It's interesting you are saying this because release 2.2 is just about to come out. There won't be such radical changes as from version 1.3 to 2.0 but minor ones for sure. For example the filter API which is also used for PHP will be more flexible and other modules are rewritten. So you can say projects like the Apache Web Server are very stable because they implement a specification which by itself is very stable so there won't be big surprises. Also this project is some sort of infrastructure software. In other types of software like book-keeping or desktop publishing it's not that well defined what the spectrum of functionality should include. So there seems to happen more than in projects like the Apache Web Server.

So back to Xaraya, why was it initiated?

Xaraya was created because PostNuke grew too fast and was too successful. The developers actually got flooded by user requests. Many users weren't used to open source and demanded support from the developers.

What kind of community has evolved there?

A community can advance with a certain rate of growth only. The problem was that there were more and more users who demanded something but the number of people who were really active in the project stayed the same. So the ratio got worse. To train people is way more difficult than just to download and install the software and ask questions. A project which wants to be successful but isn't prepared to handle thousands of requests will be drowned by them. The same would happen if all the Apache Web Server users would start asking the developers: the project would be ruined.

What led to the separation of Xaraya from PostNuke?

[...] Most of the developers always had the vision of rewriting PHPNuke from scratch. So when this flooding of new users culminated in rumours that the developers conspired against the users all of the developers but one left the project. Privately they started development of Xaraya and in addition to the code they formed organisational structures in order not to end as in PostNuke. E.g. Bitkeeper was introduced, a version control system that's also used for the development of the Linux kernel, similar to CVS but with much better features. We then designed our structures based on those of the Apache Software Foundation because this is a mature organisation that found out what worked and what not. So we founded the Digital Development Foundation that keeps the copyrights of Xaraya and should be the long term supporting institution of the project.

What kind of foundation is this?

In the beginning we wondered if an existing foundation would be willing to adopt our new project. For example at PostNuke there was one guy who had reserved the domain postnuke.com privately so he held everyone else like hostages because he could do with the domain what he wanted. This was one reason why we founded this non-profit organisation, to prevent such concentration of control in a single person. Many other open source projects have done this, too, e.g. the GNOME Foundation or Mozilla.org.

What is the main advantage of such foundations?

First of all they protect the developers against lawsuits. The basic idea is if there are any legal issues not one single developer without resources has to protect himself but since the copyrights of the developers are transferred to the foundation it takes care of all these things.

Secondly there are many practical advantages like e.g. if somebody wants to sponsor the project it's much easier to donate money to the foundation but to one single person. Also it's better because you can deduct your donations from your taxable income.

In general a foundation just gives more stability to a project. People come and go, but the institution stays. So for example at the moment I don't have much interest in Xaraya anymore but the association still stays the same and people can go on working. Like this, a foundation reduces the risk for all parti-

cipants.

How is development organized in Xaraya?

This also changed somewhat compared to my time at PostNuke. In those days if somebody came saying he wanted to be a developer he immediately got commit access to the CVS repository. But this became just too costly because many people never left the initial state. They just wanted to be members of the club and weren't willing to learn. This led to the situation that people like me were mostly occupied training new committers but in the end they left the project. It meant wasting one's time for everyone. Then we introduced the meritocracy model, again as in the Apache Foundation. So you had to be voted into the project. An established committer had to nominate somebody and then the others looked at the contribution of this nominee.

What positive or negative influence can such a voting process have on the community?

One problem in open source projects is it's common to do everything as open as possible, like over the mailing list and so on. On one hand this prevents rumours like "Oh this is a club where everything runs behind closed doors. The code is available but all the discussions are secret and I can't join them." On the other hand it can be quite difficult to live with. For example when I'm a committer and somebody gets nominated to become one and I have doubts about this person, saying this on the public mailing list would be like hitting this person into the face. Although my doubts might be justified, they still shouldn't be announced all over the place. So the voting is done mostly via a private mailing list and only the result gets published. You don't want to attack these persons personally but still you would like to be able to express your thoughts uninhibitedly. Someone e.g. might not yet have contributed enough to the project or doesn't know how to behave in this community. If this happens internally the decision might be to wait a little longer and this person never knows he's watched until it's time.

What are the rules, how do people have to behave in the project?

This depends on the project. Usually you don't want to have people on an ego trip or running amok as soon they get commit access. It's described quite well at the Apache website. Under /dev there are a lot of descriptions. The motto is "the community is more important than the code". And the community is also more important than single developers no matter how much they've contributed to the code. Otherwise you would have included a point of failure where one single person can destroy the entire project. It even happened to the Apache Foundation. In the Avalon project somebody mobbed all others out of the project. This is very unhealthy.

Why is the community more important than the code?

Because only the community can support projects in the long run. If a single person codes a lot but harms the community by his actions it's not a healthy project. The situation might change and he suddenly might not be interested in the project any longer. So it was great that he contributed so much but it's bad he destroyed the fertile soil for others to join. Open source projects are not one-man shows.

What does this fertile soil have to look like to let a community grow?

Definitively important is that it doesn't get too elite. The mood should be "If I endeavour I can become committer one day." Also it's important that criticism concentrates on ideas and code but not on persons. This has a deterring influence. [...] In the end the code is important.

Also you have to remember that for many people English is not their primary language so if somebody writes something it might be misunderstood on the other side of the planet.

Now a question about the programming language. PostNuke and Xaraya are written in PHP, Apache Lenya is a Java project. What are the differences between these communities and what influence might the programming language have?

If you look at the number of projects you'll find many more PHP projects than Java projects. Java isn't that easy to get

into and see results as quickly as in PHP so it might not be that interesting for single developers. Usually, Java projects are used in large scale applications so you find only few Java based message boards and so on. So there are also different kinds of people who are interested in these different software areas. Easy accessibility is the positive aspect of PHP and other scripting languages. On the other hand this may be a problem as you've seen in PHPNuke or PostNuke. Because the bar is much lower people quickly think they know all about software engineering. This leads to poor code and a bad position of the project itself. In PHP, VisualBasic and Perl it can be observed that the criteria of a well written software increase continuously. For example with PHP5 the difference to Java has shrunk again so if somebody really knows how to program PHP5 he shouldn't use shortcuts like global variables and all this stuff. This means VisualBasic.net is much stricter than the old VisualBasic, they tried to move the entire architecture. Time will show if they will be successful. On the other hand there are efforts to integrate Java more into the scripting world. There are examples of JSRs for this, like Jython or Groovy. These are implementations of scripting languages inside the JVM (Java Virtual Machine). Or there are some starting points in Cocoon where there is Flowscript which is in JavaScript. I believe this grows together more and more so the differences between the languages won't play a big role any longer. [...]

You were in the PostNuke community and saw all the things that went wrong. What are the properties of a healthy community?

Well, sometimes I'm a bit puzzled about what happens on the mailing lists even in the projects where I'm in at the moment, like starting rumours and so on. What I'd like to say is that all projects have these troubles from time to time. But the more frequent such problems occur, the harder it gets to participate in such a project and the less fun it is - which is still a large part of the motivation for participation in an open source project where you work in your spare time. Of course it's different if you get paid for your work.

In the beginning one reason why I participated in open source projects was that I wanted to get away from large enterprise politics. Well, it turned out that this was somewhat an illusion because in open source projects, too, there is lots of politics going on. If I'd work on a project that not many people are interested in, I wouldn't be motivated enough. I also want to learn something from others in an open source project so the way how we interact must go along with this somewhat.

What should the key players in an open source project, e.g. release managers, do to grow a healthy community?

I believe there is not one single recipe for this. I remember I was looking for such instructions in my own master's thesis but didn't find any. I believe different open source leaders also practice different styles. For example Linus Torvalds is strict but he also allows lots of freedom in his community. Some say he has a certain kind of natural laziness so he doesn't need to make decisions himself but lets the community decide when it's really necessary. He believes that a single person just isn't intelligent enough to see all the different facets of an issue. Others believe in very strict organisational structures. I think it also depends on the size and the degree of maturity of a project. For example in a project where the structure still changes a lot a different style of leadership is necessary. In that phase developers are needed who can code and publish quickly for others to improve. But in a project like the Apache Web Server such hacking around is not welcome because there are millions of users and stability is most important. Negatively said this might not attract so many people because you need talented people who aim for stability and continuity. If you were looking for new features and surprises all the time this wouldn't be the right project.

What do you have to keep in mind if you want to start an open source project today?

It's difficult to find a niche today. Actually there are two aspects: On one side it's a waste of resources if you do your own thing although there is already something similar avail-

able. On the other side people also have an ego so they prefer to do their own stuff - maybe just for fun, to try something out, which is absolutely ok. As I already mentioned once it's much easier to become known if you initiate your own project instead of being just one small wheel e.g. in the Linux kernel. For somebody who just wants to try new ideas it doesn't matter if he joins a project in the right stage or if he does his own stuff. If somebody wants to start a business with his software it's important to see some success so the initiators are able to swim along with the wave and sell their services.

What would be a good starting point for a new project? What factors have to be present?

It's always better if there is already code available instead of just ideas. In SourceForge there are many ideas that never catch on. If some code is already available it's much easier to find people because usually nobody has time to study ideas and plans. You can't submit patches for ideas. There are so many other things to do, so it must be attractive. For example I'd invest a lot of time into documentation although you have to write it over and over again. It's something I never really did, but I believe you'd be astonished what influence good documentation can have. Documentation just lowers the entry bar and gives others the chance to collaborate. Although nobody likes writing documentation it probably results in a better 'return on investment'.

What is the right moment to publish your new open source project and start looking for contributors?

The infrastructure must be ready - today this is no longer a problem with SourceForge or other solutions. And there should be some tasks available which somebody new can start tackling. Just simple things to improve like a bug tracker with some ideas. Otherwise, if somebody finds your project and the only task you offer is to build a new architecture - well, where would you start? You don't know the project yet.

What kind of small tasks would be optimal to get into the project?

Some extensions where somebody can submit patches and has a chance to start working. It's actually a good feedback loop: You start working with it, see some things to improve and send a patch to the committer, who looks at it, says it's great and commits it to the project. So you'll think "Oh that's nice!" This reassures you in the learning process.

What about publishing a task list on the website, would you think this might help attracting people or would it rather repel people?

I think it helps. Sometimes people just need a little push. Of course you can say people could look on their own. But many like it when they get a proposition or an option of doing something. So if a project has some administrative guy who likes taking care of the bug tracker already in the early stages, it might help if he assigns issues to certain people of whom he thinks they might be able to do them. And then they can comment on that. Of course you can't force anybody to do something but you can give incentives and say he or she fits very well for this job and would be welcome in the project.

You said before there should be an administrator in the project. What other roles would you define in an open source project?

Certainly someone will be the promoter. He doesn't code much but he's active in many different communities. He will do the marketing for a project and will make sure it's mentioned all over the place. Of course good coders are needed and people who like to interact with the community and help new users with their problems. And then people are needed who look after the infrastructure like CVS and so on. This might be delegated to some service provider like SourceForge. Also people must be found who do releases and maybe also a project leader is needed who tries to control the general direction of the project. Also documentation must be done by somebody.

What are the advantages and disadvantages of platforms like SourceForge?

Their advantage is that they make it easy to create a new project - but this is also a disadvantage because many immature projects will get listed. It would be cool if they had more structure. There are many projects that have a release called e.g. 0.0.0.1.alpha23-c1... So somebody should filter and look if the projects are really good for something or not. Also I'd say SourceForge isn't that innovative any more. Their bug tracker is quite old and so far they offer CVS only. Hopefully they'll offer better tools soon. Most projects that are successful and finish their childhood will leave SourceForge after a while. It also happened to PostNuke because the infrastructure didn't bear the demand.

Is it better to build your own infrastructure or to go to such a platform?

There are people who like doing such infrastructure things, but many don't. Especially in the beginning it's practical to go to such a platform. Often it's underestimated what's actually necessary to build a good infrastructure. It might mean a severe distraction from the actual work. I rather dissuade against it. Even if I've criticised SourceForge I didn't want to say that they don't offer a great service. I believe such an environment is exactly the right thing for certain stages of a project. And not all projects ever outgrow SourceForge's capacity.

What do you think about explicit credit systems?

There are different opinions. Some believe they need this absolutely. But Apache for example thinks it wouldn't fit their goal "The community is more important than individuals." So they have a credits file but it's not said who did which part. This practice is meant to prevent, say, a company from accusing specific programmers for their code. I wouldn't list credits inside the code. Another, more marketing way is to do weekly interviews with the developers and show them on the project website. KDE is such a project. Generally, promotion shouldn't take place in the code. It might even hold off people contributing when they see that 95% of the code comes from one single person.

What's the influence of extensibility of projects, e.g. an extension manager?

I believe it's important. If you abstract it somewhat, it's a modularization of the code. It's one way for a community to grow. PHP for example is an Apache module. Just imagine PHP had to be developed inside the Apache Web Server - unrealistic! The same holds for the Linux kernel and its drivers for new hardware. Modularization is good because it makes large projects tightly structured.

What is the influence of the open source license? Which licenses support community building, which ones might have a deterrent effect? Are certain licenses better for specific kinds of projects?

I believe for smaller projects the GPL is not bad because it would generate a bad mood if somebody just took the code and didn't return anything. The GPL sort of forces community building for legal reasons because people have to give back their code changes of GPL projects. That's different with the Apache License. Somewhat exaggerated, GPL communities are sort of forced communities and Apache communities are voluntary ones. Of course this leads to other problems. There are many companies that use Apache code but never return anything. So in an Apache project it's more difficult to get the people to return their code back into the project because you can't use legal means. So you must find other ways to persuade them to contribute their development work, for example by offering them to maintain their own fork or by convincing them of their contribution to the good reputation of the project. But if you want larger companies to adopt an open source project the GPL is not the right thing. We worked with customers who explicitly didn't allow the use of the GPL or LGPL because they were afraid of the loss of control over their intellectual property. If you want your software to be used as widely as possible you have to publish it under the Apache License.

And if you want to build a community you'd use the GPL?

A user community. I believe it's more difficult to build a business upon GPL code because the easiest way to make money with open source software is to keep the changes for yourself.

How about the phenomenon of never hearing anything from most of the users of your software? Obviously including some errors wouldn't be a good way to get reactions. Do you know other ways of how project owners can get more useful feedback from their users?

Especially large companies don't know how to interact with a community. But they know how to work with a producer. So usually it goes over a separate division that cares about producers. I assume this is also a business opportunity for small companies to act as an intermediate. Often these large companies are willing to return their code but they don't know how to do this in the proper way. There might also be doubts in the community about a well-known enterprise. For example if IBM would participate in an open source project by maybe employing developers for the project, there arises the question about their interests and motivation for this contribution. So it might be better to go via a third party to avoid prejudices in the community concerning the large enterprise.

Can the community of the project do anything about this or is it at the discretion of these large enterprises?

The project can create incentives for people to get in touch with the community. It's difficult. Commercial suppliers may provide new features if you register. In well-known projects like PostNuke it's helpful for small contributing companies to get listed in a directory as an implementation partner showing their competence in this area. Of course it's always somewhat dangerous to hit the right level. A possible issue might be how much influence a company might acquire on the project. Maybe it's an unsolved problem. So if I see a question actually comes from a company I sometimes write them a private email to get more information about their project.

What is the difference if a project is initiated by a company compared to initialisation by an individual developer? What are the advantages and disadvantages?

Projects that get started by a company have a clear goal. Their motivation is not to try out some new concepts but to solve certain problems. For companies it is harder to get the confidence of the community than for individuals. A company can provide resources like hiring developers to advance a project very quickly. On the other hand there are often conflicts with commercial interests. The company wants to go along this direction with the project and the community wants to go into another one. Some projects are just too large for individuals. For example there are several web services projects that are sponsored by large enterprises. The idea is to get a reference implementation which is nothing that a developer would do in spare time.

How could non-developers help an open source project?

I believe there must be the same level of incentives for them as there is for developers. It's necessary to have people who are motivated to work on the documentation. It's important not to create a two class community and to communicate that this kind of contribution is as important as coding. So it's important to value feedback and tests of the users. In this way, a self-dynamic cycle starts letting people realize that their contributions and skills are appreciated by the community. In a certain aspect it can be more complex to do these things. If for example the code doesn't compile it's quite clear what to do. But if the usability is bad, it's not so clear what should be improved. Such issues are much less visible so people are needed who point out the problems.

What kind of people is not welcome in an open source project? Which properties don't help to advance a project?

If a developer puts his personal interests above those of the community. Also just submitting ideas without showing signs of effort to realize them isn't valuable. The problem is not to get more ideas but to transfer them from a planning

phase to an implementation phase.

What are the advantages of building an international community? What are the threats of doing this?

If a software is easy to install sooner or later people on the mailing list will demand the translation into different languages. It's quite easy to accomplish and contributing translations is also a good opportunity for people to get into the project. If you plan this from the beginning it's a serious advantage. Often it's underestimated how many people don't know English very well or feel too embarrassed to participate in the mailing list. Also in certain parts of the world it's not common to participate in a discussion, for example in Asia.

What could be the advantage of an international community?

If the project is used in a certain geographical area only competition becomes high. If the project is used internationally the market also increases. In addition it's really great to wake up in the morning and receive a bunch of commit messages because people on the other side of the planet continued to develop the project. It also broadens the project and makes it useful for more than one application.

How will the market of open source developers evolve? Will there be more and more projects in the future or will there be a consolidation to a couple of large projects?

I believe there will be an increasing number of projects but also more and more large projects. There will be more open source developers because the entry bar to become one is much lower nowadays. With web applications it's easy to start. When GCC and emacs were written a kind of very highly skilled developer was needed. This implies there are different levels of skill of open source developers. Some studies conclude that open source developers are better developers since they're interested in the thing. It's also easier to employ them because one can look at the things an open source developer has accomplished already. The more there are, the more transparent the labour market will get. For companies it's important to hire the right people so in this way they can see what the developers can do. It's also an advantage for a developer's CV. Everyone writes he knows Java. But what does this actually mean? [...] I know of several people who got a job because they are Apache committers. This might be an extreme example but in the PHP area, too, companies will look at what projects you were active in. It's quite usual to google about persons to see what they've done so far.

Which kind of communication is helpful, which not?

The mailing list is the working horse. It's quite old, widely used and practical if the traffic is not too high. You have an archive including a search option and everyone can read it even if he wasn't online. On the other hand it's quite annoying to see that although you reached a consensus and maybe even a decision, after a while people forget about it and the same discussion starts all over again. Therefore tools like Wikis and bug trackers help to do write down something permanently. Additionally real time communication like IRC is also important. Not to allow some kind of conspiracy or to do design decisions but to discuss with others and just hang around with them. I believe this is important for community building.

What are the opportunities and threats of Sprints?

The communication bandwidth is extremely high and it's excellent for those who can participate. Often things can be achieved during a Sprint which normally would need months to accomplish. We already had a couple of Sprints. There are actually two different opinions. Some say they just want to meet and look what to do. The other direction is to prepare it

in more detail and to have somebody leading the Sprint. Usually if there is just a meeting, the people end up reading their email in a room. For a Sprint to be successful, specific goals are needed and not just some abstract ideas like a new architecture. The goal should be realistically achievable during the Sprint.

What are the most efficient ways to do marketing for an open source project?

Many projects don't understand that they have to care about the last mile. Other projects who understand this like Plone are very successful. There we can learn how important it is to invest time to provide for easy installation and some out-of-the-box appeal. Even if a project has a great architecture but it's difficult to install, only a small number of people will be interested in it.

Could you comment on how to spread the word?

Press releases are the usual way. To feel the pulse of the community it's good to have an aggregation of all the related Blogs like Planet OSCOM or Planet PHP. It gives an abstract overview about a community. [...] I subscribed to several to be informed when something big starts to happen. I don't want to know about every commit in these projects but want to hear about new releases or when they found out something interesting.

What are the opportunities and threats of such Blogs?

They might be misused by people talking about the project behind the official communication. In larger projects it's necessary that centralized communication takes place. [...] If you start giving support on your own developer weblog something must be wrong.

What about communication skills of developers?

Good question. Traditionally developers were rather autistic and didn't use weblogs. It's changing now, I believe. With weblogs you can see better what the developers are working on and thus the reputation might improve. Also it's a possibility to enlarge the fan club. Another effect is the developers get used to express themselves and thus the mailing list also benefits. It helps when people learn to write better.

So every developer should have a Blog?

I believe so. Maybe some people are better in coding but at least they should think about using a Blog.

When does a personal Blog start helping an open source project?

Weblogs usually have a broader spectrum of readers than the project mailing list so a wider range of people is accessed this way. [...]

What are your "do's" and "don'ts"?

Most of the projects don't take marketing seriously enough. Plone for example grew from zero to one hundred in a very short time. They understood what was needed to attract attention, like a very good CSS and other things that are not really related to the code. [...] Like having a nice website and simply making your project look sexy. Of course you shouldn't spend too much time on this but on the other hand maybe there are just those people who really know how to do it. Neither should you promise anything on the website and then it doesn't come true - no standard pictures of people shaking hands. You should be able to download the product, it works and it's open source. For example Firefox is the most successful open source project of all time. What did they do? They removed features. Or there are many small CMS which are downloaded frequently just because they're better polished than others that actually can do more. So I'd suggest to do sensible and truthful marketing.

List of Figures

Figure 1: A Very Abstract Picture of an Open Source Project (OSP).....	14
--	----

List of Tables

Table 1: Process of Building Theory from Case Study Research.....	10
Table 2: Information about the Investigated Open Source Projects, Interviewees and Interviews.....	13
Table 3: Overview of the Subject-Level Promotion Matrix of Community Building.....	56
Table 4: Human and Project Based Properties for Successful OSP Initialisation.....	105
Table 5: The Subject-Level Promotion Matrix of Community Building.....	108
Table 6: Seven Do's and Don'ts for OSP Leaders.....	112

Glossary

API	Application Programming Interface
ASF	Apache Software Foundation
BSD	Berkeley Software Distribution
CMS	Content Management System
CPS	Collaboration Portal Server
CSS	Cascading Style Sheet
CVS	Current Version System
FOP	Formatting Object Project
FSF	Free Software Foundation
GNU	GNU's not Unix
GPL	General Public License
GUI	Graphical User Interface
HTML	Hypertext Mark-up Language
HTTP	Hyper Text Transfer Protocol
IRC	Internet Relay Chat
IT	Information Technology
JSR	Java Specification Request
JVM	Java Virtual Machine
KDE	K Desktop Environment
LGPL	Lesser General Public License
LOTS	Let's Open the Source
NDA	Non Disclosure Agreement
OSCOM	Open Source Content Management
OSI	Open Source Initiative
OSP	Open Source Project
PDF	Portable Document Format
PHP	PHP Hypertext Preprocessor
PLEP	Plone Enhancement Process
PMC	Project Management Committee
RTF	Rich Text Format
SVN	Subversion
UTF	Unicode Transformation Format
WAP	Wireless Application Protocol
WebDAV	Web Distributed Authoring and Versioning
WLAN	Wireless Local Area Network
WYSIWYG	What you see is what you get
XML	Extensible Mark-up Language
XP	Extreme Programming
ZPL	Zope Public License

Bibliography

[Bauer/Pizka 2003]

Bauer, A., Pizka, M., The Contribution of Free Software to Software Evolution, in: IEEE Computer Society (Ed.), Proceedings of the 6th International Workshop on Principles of Software Evolution (2003), p. 170.

[Crowston et.al. 2004]

Crowston, K., Annabi, H., Howison, J., Masango, C., Towards A Portfolio of FLOSS Project Success Measures, Syracuse University School of Information Studies

URL: <http://opensource.mit.edu/papers/crowston04towards.pdf>

[requested: 2005-02-24].

[Eisenhardt 1989]

Eisenhardt, K., Building Theories from Case Study Research, in: Academy of Management Review, Vol. 14, No. 4 (1989), p. 532-550.

[Enzer 2005]

Enzer, M., Glossary of Internet Terms

URL: <http://www.matisse.net/files/glossary.html> [requested: 2005-02-27]

[FOLDDOC 2005]

FOLDDOC – Free On-Line Dictionary Of Computing

URL: <http://foldoc.doc.ic.ac.uk> [requested: February 2005].

[Gale IT 2005]

Glossary of Java and Related Terms

URL: http://www.galeit.com/java_glossary.htm [requested: 2005-02-15].

[Hars/Ou 2001]

Hars, A., Ou, S., Working for free? - Motivations of Participating in Open Source Projects, in: Proceedings of the 34th Hawaii International Conference on System Sciences, Hawaii (2001).

[Java Programming 2005]

Sample Quiz Answers For Chapter 4

URL: <http://math.hws.edu/javanotes/c4/quiz-answers.htm>

[requested: 2005-02-15].

[Joppe 2005]

Joppe, M., The Research Process – Convenience Sampling

URL: <http://www.ryerson.ca/~mjoppe/rp.htm>

[requested: 2005-02-24].

[Koch 2004]

Koch, S., Profiling an Open Source Project Ecology and Its Programmers, in: Electronic Markets Volume 14, Number 2 (2004), p. 77-88.

[Lakhani/Hippel 2003]

Lakhani, K., Hippel, E., How Open Source Software Works: “free” User-to-User Assistance, in: *Research Policy* 32 (2003), p. 923–943.

[Lakhani/Wolf 2005]

Lakhani, K., Wolf, R., Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects, in: *Perspectives on Free and Open Source Software* (2005).

[Lehman/Ramil 2001]

Lehman, M., Ramil, J. F., Rules and tools for software evolution planning and management, in: Springer Science+Business Media B.V. (Ed.), *Annals of Software Engineering* Volume 11, Number 1 (2001), p. 15-44.

[miniGuide 2005]

miniGuide to Zope Sprinting - What Zope Sprints are and how to prepare for them

URL: http://www.zopemag.com/Guides/miniGuide_ZopeSprinting.html

[requested: 2005-02-15].

[Parnas 1972]

Parnas, D., On the Criteria To Be Used in Decomposing Systems into Modules, in: *Communications of the ACM*, Vol. 15, No. 12 (1972), p. 1053-1058.

[Raymond 1999]

Raymond, E., *The Cathedral and the Bazaar: Musings on Linux and Open Source from an Accidental Revolutionary*, Sebastapol: O'Reilly and Associates 1999.

[Rothfuss 2002]

Rothfuss, G., A Framework for Open Source Projects, Master Thesis in Computer Science, University of Zurich (2002)

URL: <http://opensource.mit.edu/papers/rothfuss.pdf> [requested 2005-02-14].

[Silverman 1993]

Silverman, D., *Interpreting Qualitative Data: Methods for Analysing Talk, Text and Interaction*, London: SAGE Publications Ltd. 1993.

[Välimäki 2003]

Välimäki, M., Dual Licensing in Open Source Software Industry, *Systemes d'Information et Management* 1/2003

URL: <http://opensource.mit.edu/papers/valimaki.pdf> [requested 2005-02-10].

[Van Wendel de Joode 2004]

Van Wendel de Joode, R., Managing Conflicts in Open Source Communities, in: *Electronic Markets* Volume 14, Number 2 (2004), p. 104-113.

[von Krogh/Spaet/Lakhani 2003]

von Krogh, G., Spaeth, S., Lakhani, R., Community, joining, and specialization in open source software innovation: a case study, in: *Research Policy* 32 (2003) 4, p. 1217 – 1241.

[Webopedia 2005]

Webopedia: Online Computer Dictionary for Computer and Internet Terms and Definitions

URL: <http://www.webopedia.com/TERM/A/API.html>

[requested: 2005-02-15].

[Wikipedia 2005]

Wikipedia, The Free Encyclopedia

URL: <http://en.wikipedia.org> [requested: February 2005].

[Zhao/Deek 2004]

Zhao, L., Deek, F., User Collaboration in Open Source Software Development, in: Electronic Markets Volume 14, Number 2 (2004), p. 89-103.

Declaration of Discreteness

„Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche kenntlich gemacht. Mir ist bekannt, dass andernfalls der Senat gemäss dem Gesetz über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.“

Bern, 2005-03-02

Matthias Stürmer