Evaluation von Open Source Projekten: Ein GQM-basierter Ansatz

Diplomarbeit
im Fach Softwaretechnik
Studiengang Informatik
der
Freien Universität Berlin

Timo Fleischfresser

Erstprüfer:	Prof. Dr. Lutz Prechelt
Betreuer:	Christopher Oezbek

Bearbeitungszeitraum: 22. Aug. 2006 bis 22. Apr. 2007

Berlin, April 2007

Kurzfassung

Gegenstand der hier vorgestellten Arbeit ist die Entwicklung und Durchführung einer Methode zur Evaluation von Open Source Projekten. Die Zielgruppe der Evaluation besteht dabei aus Unternehmen, die Open Source Software einsetzen oder planen, dies zu tun. Basierend auf einer Umfrage an Unternehmen werden relevante Anforderungen an Open Source Projekte extrahiert, die als mögliche Zielfaktoren in die weitere Zerlegung des Evaluationsgegenstandes eingehen. Diese erfolgt im Rahmen der Arbeit nach dem Goal-Question-Metric-Verfahren (GQM-Verfahren), einem iterativen Top-Down-Verfahren, das insgesamt drei Analyseebenen unterscheidet. Diese bestehen in speziellen Zielen, die die Richtung der Untersuchungen vorgeben, speziellen Fragestellungen, die das jeweilige Ziel operationalisieren, und Metriken zur Messung der entwickelten Fragestellungen. Ausgehend von einer bestehenden allgemeinen Zerlegung von Open Source Projekten nach Cruz *et al.* wird der vorgezeichnete Ansatz aufgrund bestehender Schwächen um einen speziellen Kontext erweitert. Dieser besteht in der Fokussierung auf den Zielfaktor „Zukunftssicherheit“ und den Anwendungsbereich „Web Content Management Systeme“. Für diesen Kontext wird dann eine GQM-Zerlegung entwickelt, die den Zielfaktor „Zukunftssicherheit“ von Open Source Projekten in die Teilaspekte kommerzielle Ausrichtung, integrierter Schulungsprozess für neue Entwickler, Projektgröße und Projektaktivität sowie die Existenz einer Kernideologie aufspaltet. Schließlich erfolgt die Validierung der entwickelten Zerlegung im Rahmen einer Fallstudie von sechs ausgewählten Open Source Projekten aus dem Anwendungsbereich der Content Management Systeme. Die Fallstudien bestätigen dabei bestehende Zusammenhänge zwischen der kommerziellen Ausrichtung, dem etablierten Schulungsprozess und der Projektaktivität von Open Source Projekten und ihrer Zukunftssicherheit.

Abstract

Object of the presented diploma theses is the development of a method to evaluate Open Source Projects. The target group of the evaluation consists of companies, which already use Open Source Software or plan to use it in the future. Based on survey, important requirements of companies concerning Open Source Projects are extracted, which are potential objects of the decomposition of Open Source Projects in the next phase. The decomposition of Open Source Projects within the theses is realised with the Goal-Question-Metric-Approach, an iterative Top-Down-Method, which seperates three contentual levels. These are specific goals, questions to characterize these goals and

metrics to measure the questions. Based on a general decomposition of Open Source Projects by Cruz *et al.* an enhancement of the outlined approach is proposed. This enhancement consists of a specific context which includes the concentration on the specific exploration object „future prospects of Open Source Projects" and the software category „Web Content Management Systems". Within this context a GQM-based decomposition of Open Source Projects is developed, which includes the commercial orientation of Open Source Projects, the process to train new developers, the project size, the project activity and the existance of a core ideology. Finally this decomposition is validated through a case study with six Open Source Web Content Management Systems. The study confirms existing relations between the commercial orientation, the process to train new developers, the project activity and the future prospects of Open Source Projects.

Inhaltsverzeichnis

Kurzfassung **2**

Abstract **2**

Inhaltsverzeichnis **4**

Abbildungsverzeichnis **7**

Tabellenverzeichnis **8**

Abkürzungsverzeichnis **9**

Vorwort **10**

1 Ziele **11**

2 Überblick **12**

3 Grundlagen einer Evaluation **13**

3.1 Evaluationsarten 14

3.2 Eigenschaften von Evaluationen 14

3.2.1 Nützlichkeit 14

3.2.2 Durchführbarkeit 15

3.2.3 Fairness 15

3.2.4 Genauigkeit 15

3.3 Phasen einer Evaluation 16

4 Definition des Evaluationsproblems **17**

4.1 Beteiligte Gruppen 17

4.1.1 Anwendungsunternehmen 17

4.1.2 Softwareunternehmen 18

4.2 Evaluationszwecke 18

4.3 Open Source Projekte 19

4.3.1 Begriffe 19

4.3.2 Rollen 23

4.3.3 Infrastrukturkomponenten 26

4.3.4 Lebenszyklus 27

4.3.5 Lizenzen 36

4.3.6 Geschichte 45

4.4 Allgemeine Kontextanalyse 49

4.4.1 Soziale Rahmenbedingungen in Open Source Projekten 49

4.4.2 Wirtschaftliche Rahmenbedingungen in Open Source Projekten 58
4.4.3 Resultierende Stärken von Open Source Projekten 62
4.4.4 Resultierende Schwächen von Open Source Projekten 64
4.4.5 Ziele eines Einsatzes von Open Source Software in Unternehmen 66
4.4.6 Anforderungen der Unternehmen an Open Source Projekte 76

5 Planung der Evaluation 98

5.1 Der „Goal-Question-Metric"-Ansatz (GQM) 98

5.2 Übertragung des Ansatzes auf Open Source Projekte 100
5.2.1 Zerlegung nach Cruz *et al.* 100
5.2.2 Probleme 103

5.3 Erweiterung des Ansatzes von Cruz *et al.* 104
5.3.1 Der Zielfaktor „Zukunftssicherheit" 104
5.3.2 Das Einsatzgebiet „Web Content Management Systeme" 122

6 Informationsgewinnung 125

6.1 CVS/SVN-Daten 126
6.1.1 Alter der Open Source Projekte (1) 128
6.1.2 Größe der Open Source Projekte 129
6.1.3 Aktivität der Entwickler 130

6.2 Umfrage an die Projektadministratoren 130
6.2.1 Allgemeine Vorgehensweise 131
6.2.2 Kommerzielle Ausrichtung (1) 131
6.2.3 Ausformung des Schulungsprozesses für Entwickler und Benutzer 132
6.2.4 Projektideologie und Projektziele (1) 133
6.2.5 Zukunftssicherheit 134

6.3 Die Webseite 134
6.3.1 Alter der Open Source Projekte (2) 135
6.3.2 Kommerzielle Ausrichtung (2) 135
6.3.3 Projektideologie und Projektziele (2) 136

7 Informationsauswertung 137

7.1 Ergebnisse der Datenerhebung 137
7.1.1 Drupal 137
7.1.2 Magnolia 141
7.1.3 Mambo 145
7.1.4 OpenCms 148
7.1.5 TGS CMS 151
7.1.6 Typo3 153

7.2 Einflüsse auf den Zielfaktor „Zukunftssicherheit" 156
7.2.1 Kommerzielle Ausrichtung 156
7.2.2 Schulungsprozess 161

7.2.3 Projektgröße/Projektaktivität 165
7.2.4 Projektideologie/Projektziele 168

8 Zusammenfassung und Ausblick 171

Anhang A: Eigenschaften der unterschiedlichen Lizenzkategorien 174
A.1 „Copyleft“-Lizenzen 174
A.2 „BSD“-artige Lizenzen 175
A.3 „Mozilla“-artige Lizenzen 176
A.4 „Artistic“-Lizenzen 177

Anhang B: Umfrage zum Thema „Risiken und Vorteile eines Einsatzes von Open Source Software“ 178
B.1 Fragenkatalog der Unternehmensumfrage 178
B.2 Einsatzgebiete von OSS in Anwendungsunternehmen 180
B.3 Einsatzgebiete von OSS in Softwareunternehmen 181
B.4 Risiken eines Einsatzes von OSS aus Unternehmenssicht 182
B.5 Vorteile eines Einsatzes von OSS aus Unternehmenssicht 184

Anhang C: GQM-Modell nach Cruz *et al.* 186
C.1 Anforderungen der Unternehmen 186
C.2 Projektkriterien und abgeleitete Fragestellungen 187

Anhang D: Datenerhebung zum Thema „Zukunftssicherheit von Open Source Projekten“ 190
D.1 Fragenkatalog der Projektumfrage 190
D.2 Ergebnisse der CVS/SVN-Analysen 193

Glossar 194

Literaturverzeichnis 196

Erklärung 200

Abbildungsverzeichnis

Abbildung 1: Ziele und Vorgehensweise der Arbeit 11
Abbildung 2: Aufbau der Arbeit 12
Abbildung 3: Kategorien Freier und Nicht-Freier Software nach der FSF 21
Abbildung 4: Struktur einer Open Source Community nach Crowsten und Howison ... 23
Abbildung 5: Hierarchie und Rollenmigration im Apache-Projekt nach Jensen und Scacchi 52
Abbildung 6: Hierarchie und Rollenmigration im Mozilla-Projekt nach Jensen und Scacchi 52
Abbildung 7: Maximaler und minimaler Zentralisierungsgrad von Interaktionen in einem sozialen Netzwerk 57
Abbildung 8: Abstrakter Gesamtbetriebskostenverlauf bei einer Migration auf Open Source Software 71
Abbildung 9: Dimensionen eines Zielfaktors innerhalb des GQM-Verfahrens 99
Abbildung 10: Allgemeiner Aufbau eines GQM-Modells 100
Abbildung 11: Allgemeine Vorgehensweise von Cruz *et al.* 101
Abbildung 12: Allgemeine Modellierung der Einflüsse auf den Zielfaktor „Zukunftssicherheit“ 109
Abbildung 13: Schlüsselkonzept visionärer Unternehmen nach Collins und Porras 118
Abbildung 14: Entstehung einer Versionsverzweigung 126
Abbildung 15: Verlauf der Commitzahlen im Quellcodeverzeichnis von April 2006 bis März 2007 166

Tabellenverzeichnis

Tabelle 1: Phasen einer Evaluation 16
Tabelle 2: Typen der Rollenmigration in Open Source Projekten nach Jensen und Scacchi 53
Tabelle 3: Anzahl der Mitarbeiter der Unternehmen 80
Tabelle 4: Branchen der Unternehmen 81
Tabelle 5: Kontakt der Softwareunternehmen zu Open Source Projekten 86
Tabelle 6: Anforderungen der Anwendungsunternehmen an einen Einsatz von OSS 93
Tabelle 7: Anforderungen der Softwareunternehmen an einen Einsatz von OSS 95
Tabelle 8: GQM-Zerlegung des Faktors „Reduzierung der Kosten" nach Cruz *et al.* 102
Tabelle 9: GQM-Zerlegung des Zielfaktors „Zukunftssicherheit von Open Source Projekten" 121
Tabelle 10: Erfassung der Zukunftssicherheit des Magnolia- und des OpenCms-Projektes 157
Tabelle 11: Erfassung der Zukunftssicherheit von Drupal, Mambo, Typo3 und TGS CMS 158
Tabelle 12: Kriterien für den zukünftigen Erfolg der untersuchten Open Source Projekte 160
Tabelle 13: Etablierte Schulungsmechanismen im OpenCms- und im Magnolia-Projekt 162
Tabelle 14: Etablierte Schulungsmechanismen in den nicht-kommerziellen OSPs 163
Tabelle 15: Verhältnis der Entwicklerzahl zum resultierenden Output 165
Tabelle 16: Zusammenhang zwischen dem Quotienten LOC changed/LOC added in den letzten drei Monaten und der zukünftigen Softwarequalität 167
Tabelle 17: Zusammenhang zwischen dem Quotienten LOC added/#Commits in den letzten drei Monaten und der zukünftigen Softwarequalität 167
Tabelle 18: Eigenschaften von „Copyleft“-Lizenzen 174
Tabelle 19: Eigenschaften „BSD“-artiger Lizenzen 175
Tabelle 20: Eigenschaften „Mozilla“-artiger Lizenzen 176
Tabelle 21: Eigenschaften von „Artistic“-Lizenzen 177
Tabelle 22: Einsatzgebiete von OSS in Anwendungsunternehmen 180
Tabelle 23: Art des Einsatzes von OSS in Softwareunternehmen 181
Tabelle 24: Gegenüberstellung der Risiken eines Einsatzes von OSS aus Unternehmenssicht 182
Tabelle 25: Gegenüberstellung der Vorteile eines Einsatzes von OSS aus Unternehmenssicht 184
Tabelle 26: Anforderungen von Unternehmen an Open Source Software nach Cruz *et al.* 186
Tabelle 27: Projektkriterien und Fragestellungen nach Cruz *et al.* 187
Tabelle 28: Ergebnisse der CVS/SVN-Analysen von Drupal, Magnolia, Mambo, OpenCms, TGS CMS und Typo3 193

Abkürzungsverzeichnis

ASF	Apache Software Foundation
BSI	Bundesamt für Sicherheit in der Informationstechnik
CVS	Concurrent Version System
DeGEval	Deutsche Gesellschaft für Evaluation
ECMS	Enterprise Content Management System
FLOSS	Free/Libre/Open Source Software
FOSS	Free and Open Source Software
FSF	Free Software Foundation
GPL	General Public License
GQM	Goal-Question-Metric
IRC	Internet Relay Chat
JEMS	JBoss Enterprise Middleware Suite
LGPL	Lesser General Public License
OSI	Open Source Initiative
OSP	Open Source Projekt
OSS	Open Source Software
SVN	Subversion
TCO	Total Cost of Ownership
WCMS	Web Content Management System

Vorwort

Seit der Gründung des GNU-Projektes, des ersten bewusst als „frei“ bezeichneten Softwareprojektes vor mehr als 20 Jahren, hat sich die Softwarelandschaft im Bereich Open Source Software stark entwickelt. Die weit verbreitete Gleichsetzung von quelloffener Software mit dem von Linus Torvalds entwickelten Betriebssystem Linux trifft längst nicht mehr zu. Anwendungen wie der Apache-Webserver, die MySQL-Datenbank oder die von IBM und anderen Unternehmen entwickelte Entwicklungsumgebung Eclipse sind mittlerweile in zahlreichen Unternehmen etabliert und stehen vergleichbaren proprietären Systemen qualitativ in nichts nach. Die Entscheidung der Stadt München aus dem Jahr 2003 zu einer kompletten Migration ihrer 14.000 PC-Systeme von einem MS Windows-Betriebssystem mit MS Office-Anwendungssoftware hin zu einer auf Open Source Software basierenden Infrastruktur bestehend aus dem Linux-Betriebssystem und der Office-Anwendung OpenOffice zeigt, dass eine solche Lösung längst nicht mehr nur eine Alternative für den interessierten Einzelanwender ist, sondern auch im Kontext der gewerblichen Nutzung Anwendung findet. Die Vorteile für Unternehmen liegen dabei im Wesentlichen in der größeren Herstellerunabhängigkeit und den häufig geringeren Kosten.

Demgegenüber stehen allerdings auch spezifische Risiken, die der Einsatz von Open Source Software mit sich bringt. So kann für Softwareunternehmen die Anwendung des Copyleft dazu führen, dass das eigene Programm rechtlich gesehen ebenfalls zu Open Source Software wird, sofern Teile des Quellcodes offener Software unter bestimmten Lizenzen in die eigenen Entwicklungen eingebunden werden. Hinzu kommt, dass Open Source Projekte im Allgemeinen weder Garantien auf die Softwarequalität noch auf die Weiterentwicklung der Software liefern, so dass Unternehmen gegebenenfalls vor dem Risiko stehen, auf eine andere Software migrieren zu müssen.

Vor diesem Hintergrund soll die folgende Arbeit Unternehmen dabei unterstützen, eigenständig wichtige Einflussfaktoren in Open Source Projekten zu identifizieren und im Zuge einer Evaluation bewerten zu können.

1 Ziele

Die folgende Arbeit verfolgt im Wesentlichen zwei Ziele. Erstens soll sie ein Rahmenwerk zur Durchführung eigener Evaluationen zu Open Source Projekten durch Unternehmen als erklärte Zielgruppe darstellen. Dazu erfolgen zu Beginn eine genaue Analyse der Strukturen und Prozesse in den Projekten sowie eine Beleuchtung des betrieblichen Umfeldes. Dies soll gewährleisten, dass wichtige Einflussfaktoren einerseits erkannt und andererseits auch erklärt werden können. Im nächsten Schritt wird dann eine Methode zur Zerlegung des Evaluationsgegenstandes „Open Source Projekte“ vorgeschlagen, das *Goal-Question-Metric-Verfahren (GQM-Verfahren).* Dieses soll Unternehmen dabei unterstützen, für sie relevante *Zielfaktoren* methodisch auf Metriken herunterzubrechen, die eine Erfassung der entsprechenden Faktoren ermöglichen.

Zweitens soll die Arbeit parallel dazu die konkrete Durchführung einer Evaluation an Hand eines ausgewählten Teilaspekts von Open Source Projekten aufzeigen. Dazu wird auf der Basis einer durchgeführten Unternehmensumfrage sowie den in der Problemdefinitionsphase angeführten Untersuchungen der abstrakte Zielfaktor „Zukunftssicherheit“ extrahiert. Dieser wird dann im zweiten Schritt im Zuge der Einordnung in ein GQM-Modell operationalisiert. Zur Auswertung der ermittelten Zerlegung dienen schließlich drei Informationsquellen. Diese bestehen aus der Webseite des Open Source Projektes, den CVS- bzw. SVN-Daten und einer durchgeführten Befragung der Kernentwickler der Projekte.

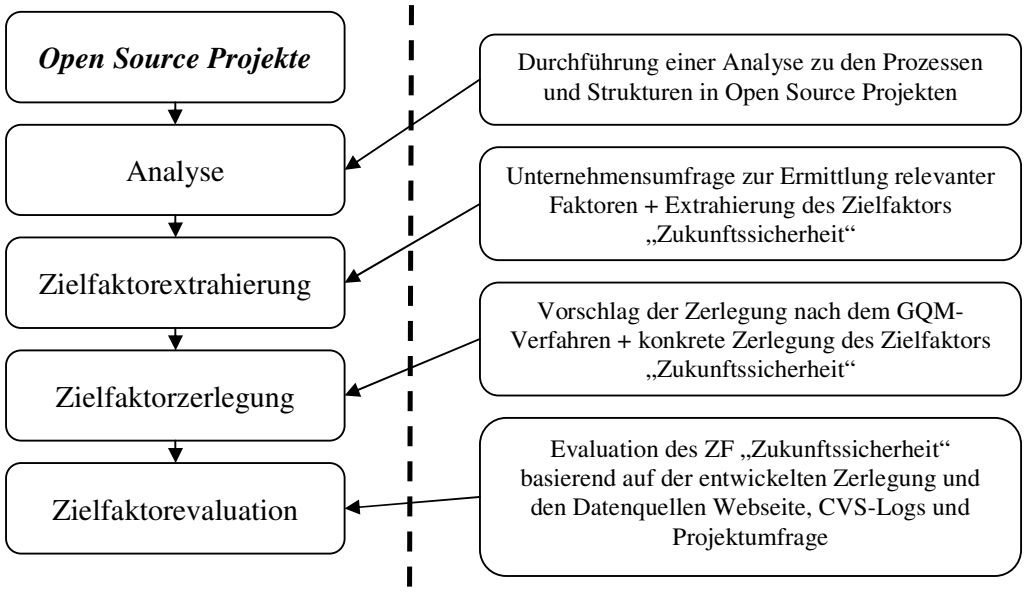

Abbildung 1: Ziele und Vorgehensweise der Arbeit

2 Überblick

Der Aufbau der folgenden Arbeit ist angelehnt an eine Anleitung zur Selbstevaluation von Open Source Projekten. Dazu wird in Kapitel 3 zunächst das formale Rahmenwerk der *Deutschen Gesellschaft für Evaluation (DeGEval)* vorgestellt, auf dem die weitere Vorgehensweise zur Evaluation des Untersuchungsgegenstandes beruht. Dies beinhaltet eine Unterscheidung verschiedener Evaluationsarten, die Definition von Evaluationsstandards zur Gewährleistung einer professionellen Betrachtung sowie die Identifikation mehrerer Phasen, die im Zuge einer Evaluation in der Regel sequentiell durchlaufen werden. Im ersten inhaltlichen Schritt erfolgt dann in Kapitel 4 die Definition des Evaluationsproblems. Dies umfasst eine genaue Analyse des Evaluationsgegenstandes, seiner Umgebung sowie die Klärung der Evaluationszwecke. Übertragen auf die definierten Ziele (siehe Abb. 1) bedeutet dies die Durchführung der Analyse zu den Prozessen und Strukturen in Open Source Projekten und die Ermittlung der allgemein relevanten Zielfaktoren auf der Basis der Unternehmensumfrage. In Kapitel 5 wird anschließend zunächst die methodische Grundlage des aufgezeigten Evaluationsweges, das GQM-Verfahren, erläutert. Im zweiten Schritt wird das Verfahren dann auf eine bereits bestehende allgemeine Zerlegung von Open Source Projekten durch Cruz *et al.* angewandt. Da begründete Kritik an der Allgemeingültigkeit der getroffenen Aussagen besteht, wird der Ansatz von Cruz *et al.* um einen speziellen Kontext erweitert. Dies beinhaltet die Fokussierung auf einen wesentlichen Zielfaktor sowie die Festlegung der Anwendungsart der Software. Schließlich erfolgt in den Kapiteln 6 und 7 die Durchführung der vorgezeichneten Evaluation für den Zielfaktor „Zukunftssicherheit“ im Bereich der *Web Content Management Systeme*.

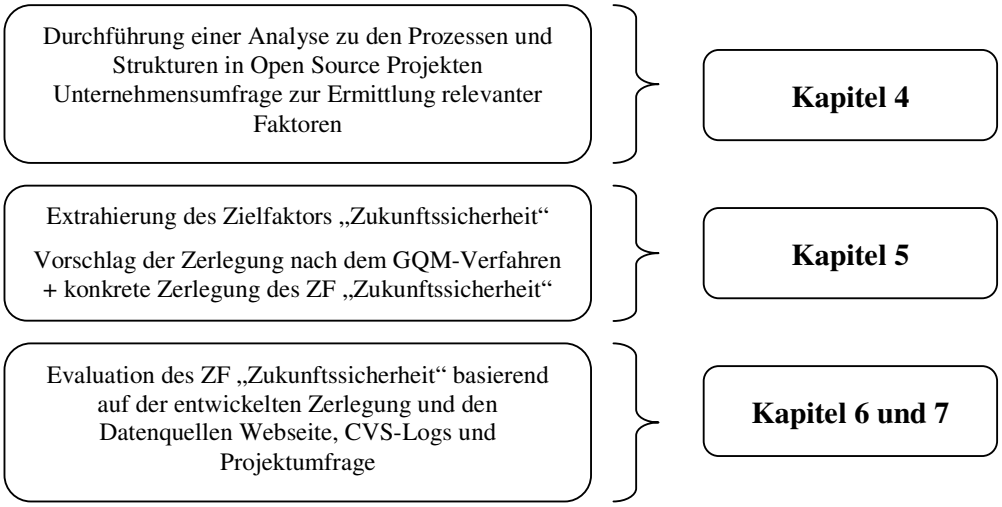

Abbildung 2: Aufbau der Arbeit

3 Grundlagen einer Evaluation

„Evaluation ist die systematische Untersuchung des Nutzens oder des Wertes eines Gegenstandes. Solche Evaluationsgegenstände können z.B. Programme, Projekte, Produkte, Maßnahmen, Leistungen, Organisationen, Politik, Technologien oder Forschung sein. Die erzielten Ergebnisse, Schlussfolgerungen oder Empfehlungen müssen nachvollziehbar auf empirisch gewonnenen quantitativen und/oder qualitativen Daten beruhen.“

- Deutsche Gesellschaft für Evaluation [DeGEval2001, S.13]

Der erste konzeptionelle Schritt zur Durchführung einer Evaluation besteht in der Modellierung der Vorgehensweise zu einer umfassenden Untersuchung und Bewertung des Evaluationsgegenstandes. Die angeführte Definition des Begriffes „Evaluation“ betont dazu vor allem zwei Aspekte, die für die Qualität einer Evaluation von entscheidender Bedeutung sind. Zum einen muss die Untersuchung des Objektes systematisch erfolgen. Dies impliziert eine strukturierte und nachvollziehbare Vorgehensweise. Zum anderen müssen die erzielten Resultate auf empirisch gewonnenen Daten basieren. Um beide Forderungen in die Abläufe einer Evaluation integrieren zu können, definiert die DeGEval[1] ein Rahmenwerk für den Kontext professioneller Evaluationen [DeGEval2001]. Dieses besteht erstens aus prototypischen Phasen, die im Zuge der Untersuchung und Bewertung des entsprechenden Objektes durchlaufen werden. Zweitens beschreibt es charakteristisch geforderte Eigenschaften von Evaluationen in Form von Standards, die den jeweiligen Phasen zugeordnet sind. Drittens schließlich werden zwei unterschiedliche Evaluationsarten unterschieden, die mit der Erfüllung verschiedener Zwecke verbunden sind. Das resultierende Gesamtrahmenwerk bildet die Grundlage für den in der Arbeit aufgezeigten Weg zur Evaluation von Open Source Projekten.

[1] Deutsche Gesellschaft für Evaluation

3.1 Evaluationsarten

Entsprechend ihren unterschiedlichen Zwecksetzungen differenziert die DeGEval zwischen zwei Formen der Evaluation. Dies sind zum einen *Formative Evaluationen*, die den Entwicklungsprozess des Evaluationsgegenstandes begleiten und die bereits während der Produktentwicklung mögliche Prozessverbesserungen erkennbar machen sollen. Zum anderen nennt die DeGEval *Summative Evaluationen*, deren Leistung demgegenüber darin bestehen, eine zusammenfassende Bewertung des Evaluationsgegenstandes zu liefern und damit Entscheidungen über ihn zu ermöglichen. Die in der Arbeit aufgezeigte Methode zur Evaluation von Open Source Projekten entspricht einer Summativen Evaluation.

3.2 Eigenschaften von Evaluationen

Unabhängig von der Evaluationsart bedarf es zur Gewährleistung der Qualität einer Evaluation objektiver Normen, die sicherstellen, dass das Objekt systematisch, vollständig und unter Berücksichtigung der Interessen aller beteiligten Parteien untersucht wird. Dazu beinhaltet das Rahmenwerk der DeGEval 25 Standards, die als Pfeiler für die Modellierung einer Evaluation dienen. Die Standards sind in die vier Kategorien *Nützlichkeit, Durchführbarkeit, Fairness* und *Genauigkeit* unterteilt. Im Folgenden werden die Kategorien kurz erläutert sowie die entsprechenden Standards genannt.

3.2.1 Nützlichkeit

Die Nützlichkeitsstandards sollen die Ausrichtung der Evaluation an den geklärten Evaluationszwecken sowie am Informationsbedarf der vorgesehenen Nutzer und Nutzerinnen sicherstellen.

- N1 Identifizierung der Beteiligten und Betroffenen
- N2 Klärung der Evaluationszwecke
- N3 Glaubwürdigkeit und Kompetenz des Evaluators / der Evaluatorin
- N4 Auswahl und Umfang der Informationen
- N5 Transparenz von Werten
- N6 Vollständigkeit und Klarheit der Berichterstattung
- N7 Rechtzeitigkeit der Evaluation
- N8 Nutzung und Nutzen der Evaluation

3.2.2 Durchführbarkeit

Die Durchführbarkeitsstandards sollen eine realistische, gut durchdachte, diplomatische und kostenbewusste Planung und Ausführung der Evaluation sicherstellen.

- D1 Angemessene Verfahren
- D2 Diplomatisches Vorgehen
- D3 Effizienz von Evaluationen

3.2.3 Fairness

Die Fairnessstandards sollen den respektvollen und fairen Umgang mit den betroffenen Personen und Gruppen sicherstellen.

- F1 Formale Vereinbarungen
- F2 Schutz individueller Rechte
- F3 Vollständige und faire Überprüfung
- F4 Unparteiische Durchführung und Berichterstattung
- F5 Offenlegung der Ergebnisse

3.2.4 Genauigkeit

Die Genauigkeitsstandards schließlich sollen die Gültigkeit von Informationen und Ergebnissen zu dem jeweiligen Evaluationsgegenstand und den Evaluationsfragestellungen sicherstellen.

- G1 Beschreibung des Evaluationsgegenstandes
- G2 Kontextanalyse
- G3 Beschreibung von Zwecken und Vorgehen
- G4 Angabe von Informationsquellen
- G5 Valide und reliable Informationen
- G6 Systematische Fehlerprüfung
- G7 Analyse qualitativer und quantitativer Daten
- G8 Begründete Schlussfolgerungen
- G9 Meta-Evaluation

3.3 Phasen einer Evaluation

Schließlich unterteilt die DeGEval die Abläufe innerhalb einer Evaluation in insgesamt 10 aufeinander folgende Phasen, die typischerweise im Rahmen der Untersuchungen des Evaluationsgegenstandes durchlaufen werden. Jeder Phase sind dabei für sie charakteristische Standards zugeordnet, sodass als Resultat ein Rezept zur Durchführung von Evaluationen – das Rahmenwerk der DeGEval – entsteht. Da die aufgeführten Phasen für den Kontext kommerziell beauftragter Evaluationen entwickelt wurden, entsprechen nicht alle den Umständen der Arbeit, sodass lediglich die Phasen 2 bis 5 behandelt werden.

Tabelle 1: Phasen einer Evaluation

Nr.	Name der Phase	Standards
1	Entscheidung über die Durchführung einer Evaluation	N1, N2, N7, N8, D2, D3, F1, G1, G2
2	Definition des Evaluationsproblems	N1, N2, G1, G2, G3
3	Planung der Evaluation	N1, N2, N3, N4, N5, N7, N8, D1, D2, D3, F1, F3, F4, G1, G2, G3, G4, G5, G7, G8, G9
4	Informationsgewinnung	N3, N4, N5, D1, D2, F1, F2, F3, G1, G2, G3, G4, G5, G6, G9
5	Informationsauswertung	N5, F4, G1, G2, G6, G7, G8, G9
6	Berichterstattung zur Evaluation	N1, N4, N5, N6, N7, N8, F2, F3, F4, F5, G1, G2, G3, G4, G8, G9
7	Budgetierung der Evaluation	N2, N4, D3, F1, G1, G3
8	Evaluationsvertrag	N1, N2, N3, N4, N7, D2, D3, F1, F2, F4, F5, G1, G3, G9
9	Steuerung der Evaluation	N1, N2, N3, N7, N8, D2, D3, F1, F2, F4, G3, G6, G9
10	Personelle Ausstattung der Evaluation	N3, N4, D2, F4, G1

4 Definition des Evaluationsproblems

Der erste inhaltliche Schritt zur Evaluation von Open Source Projekten besteht in einer detaillierten Analyse des Evaluationsgegenstandes und seiner Umgebung. Übertragen auf das in Kapitel 3 vorgezeichnete Rahmenwerk (siehe Tab. 1) bedeutet dies zunächst die Identifikation der beteiligten Gruppen (N1) sowie die Klärung der Evaluationszwecke (N2). Danach erfolgen eine differenzierte Betrachtung des Evaluationsgegenstandes (G1) sowie eine allgemeine Kontextanalyse (G2). Die Betrachtung des Evaluationsgegenstandes beinhaltet dabei eine begriffliche Abgrenzung von Open Source Projekten und ihrer Software, eine Analyse der Infrastruktur, eine Darstellung des Lebenszyklus der Projekte sowie eine juristische und eine historische Einordnung. Im Zuge der allgemeinen Kontextanalyse werden dann die Rahmenbedingungen sowohl in den Projekten als auch in den Unternehmen als mögliche Zielgruppe der Projektsoftware genauer untersucht. Die während der Definition des Evaluationsproblems gesammelten Informationen bilden die Basis für die spätere Zerlegung des Evaluationsgegenstandes in der Planungsphase.

4.1 Beteiligte Gruppen

Neben Open Source Projekten selbst gehören Unternehmen und Behörden, die einen internen Einsatz von Open Source Software planen oder bereits realisiert haben, als Zielgruppe der Software zu den Beteiligten. Entsprechend ihren unterschiedlichen Anforderungen an die Projekte und ihr Produkt muss diese Gruppe in zwei Kategorien unterteilt werden. Dies sind zum einen *Anwendungsunternehmen*, die Open Source Software lediglich als Anwendung nutzen, ohne dabei Modifikationen der Software zu verbreiten. Zum anderen sind dies *Softwareunternehmen*, die offene Bibliotheken[2], Frameworks[3] oder Teile ganzer Anwendungen dazu verwenden, eigene kommerziell vertriebene Produkte zu entwickeln oder zu verbessern.

4.1.1 Anwendungsunternehmen

Unter dem Begriff „Anwendungsunternehmen“ werden im Rahmen der Arbeit alle Unternehmen zusammengefasst, die allgemein Software und speziell Open Source

[2] Softwarebibliotheken sind Implementierungen von Programmierschnittstellen, die zur Entwicklung von Anwendungen benutzt werden können

[3] Frameworks sind implementierte Softwarerahmenwerke, die als Basis für die Entwicklung erweiterter Anwendungen genutzt werden können

Software lediglich als Anwendung nutzen. Dies bedeutet, dass es nicht zum Portfolio der Unternehmen gehört, selbst erstellte oder modifizierte Software zu verkaufen. Das Anbieten von Diensten, die auf der Verwendung bestimmter Software beruhen, wie die Bereitstellung von Adressdatenbanken, ist dagegen erlaubt. Darüber hinaus besitzen auch Anwendungsunternehmen teilweise Entwicklungsteams, die Software an die Bedürfnisse des jeweiligen Unternehmens anpassen, wobei dies bereits juristische Abgrenzungsschwierigkeiten mit sich bringen kann.[4] Eine besondere Gruppe der Anwendungsunternehmen bilden öffentliche Behörden. Diese sind zwar keine Unternehmen im eigentlichen Sinne, zählen aber als potentielle Abnehmer von Open Source Software im Rahmen der Arbeit in diese Kategorie.

4.1.2 Softwareunternehmen

Im Gegensatz zu Anwendungsunternehmen verdienen Softwareunternehmen an der Erstellung oder Modifikation eigener bzw. fremder Software. Dies bedeutet, dass der Vertrieb einer Kombination aus Software des Unternehmens und Open Source Software bzw. der Vertrieb einer Modifikation von Open Source Software zu den Einnahmequellen des Unternehmens gehören kann. Dadurch entstehen in Verbindung mit einigen Open Source Lizenzen juristische Schwierigkeiten, die einem solchen Geschäftsmodell im Wege stehen. Darüber hinaus spielt für Softwareunternehmen im Gegensatz zu Anwendungsunternehmen auch die zunehmend aufkommende Konkurrenzsituation zu Open Source Projekten eine wesentliche Rolle. Die genauen Ziele, die sowohl Software- wie auch Anwendungsunternehmen mit dem Einsatz von Open Source Software verfolgen, werden später erläutert.[5]

4.2 Evaluationszwecke

Um eine aufwändige und in einem Unternehmen eventuell kostspielige Evaluation zu rechtfertigen, bedarf es der Klärung der Intentionen, die mit der Durchführung einer solchen verbunden sind. Die DeGEval unterscheidet dazu zwischen den zwei Begriffen „Ziel“ und „Zweck“. Während Ziele demnach mit den konkreten Vorstellungen der beteiligten Gruppen zum Ausgang einer Evaluation in Verbindung stehen und damit die Frage nach dem „was will ich“ beantworten, beschreiben die Evaluationszwecke die Gründe für eine Evaluation an sich und beantworten damit die Frage nach dem „warum will ich“.

---

[4] siehe Kapitel 4.3.5 „Lizenzen“

[5] siehe Kapitel 4.4 „Allgemeine Kontextanalyse“

Die Zwecke, die mit der im Rahmen der Arbeit aufgezeigten Evaluation des Evaluationsgegenstandes „Open Source Projekte“ verbunden sind, lassen sich gut durch ein Zitat von Christel Marquardt, Angestellte des *Bundesamtes für Sicherheit in der Informationstechnik (BSI)*, zum Einsatz von Open Source Software in deutschen Behörden verdeutlichen:

> *„[…] Ein großes Hindernis, das viele Behörden davon abhält, Open Source Software einzusetzen, sind fehlende Informationen.“*
>
> *- Christel Marquardt 2004 [Marquardt2004, S.58]*

Diesem Zitat folgend verfolgt die Arbeit vor allem den Zweck der Klärung des Informationsbedarfes. Diese erfolgt allerdings auf zwei Ebenen. Zum einen werden Open Source Projekte und ihr Umfeld beleuchtet und damit allgemeine Informationen zur besseren Einschätzung des Evaluationsgegenstandes bereitgestellt. Zum anderen wird die Evaluation eines ausgewählten Zielfaktors[6] direkt aufgezeigt und somit werden wichtige Einflussfaktoren identifiziert.

4.3 Open Source Projekte

Nachdem zuvor die beteiligten Gruppen im Umfeld von Open Source Projekten beschrieben wurden, ist vor allem der Aufbau des Evaluationsgegenstandes selbst ein Schwerpunkt der Analysen der Phase der Definition des Evaluationsproblems. Dies erfordert im ersten Schritt eine genaue Abgrenzung der unterschiedlichen Bezeichnungen, die sich im Zusammenhang mit dem Produkt der Projekte – der Open Source Software – entwickelt haben, sowie eine Definition des Begriffes *„Open Source Projekt“* selbst.

4.3.1 Begriffe

Um Open Source Projekte analysieren bzw. evaluieren zu können, ist es zunächst einmal erforderlich, den genauen Umfang abzustecken, für den der Begriff Gültigkeit besitzen soll. Im Rahmen dieser Arbeit steht die Bezeichnung „Open Source Projekt“ dabei nicht bloß für die Gruppe der beteiligten Individuen, sondern für die Gesamtheit der Entitäten und Aktivitäten, die mit der Entwicklung von Open Source Software verbunden sind.

---

[6] Teilaspekt von Open Source Projekten innerhalb des GQM-Verfahrens, siehe Kapitel 5 „Planung der Evaluation“

Definition: *Der Begriff „Open Source Projekt“ (OSP) bezeichnet die Gesamtheit der Strukturen, Prozesse und beteiligten Personen im Zusammenhang mit der Entwicklung von Open Source Software (OSS) sowie die Software selbst.*

Definition: *Die Gruppe der an der Entwicklung aktiv oder passiv beteiligten Personen wird als „Community“ des Open Source Projektes bezeichnet.*

Entscheidend für die beiden Definitionen ist der Bezug zur Entwicklung von Open Source Software. Dies wirft jedoch in der Realität einige Abgrenzungsschwierigkeiten auf, da sich im Laufe der Geschichte der offenen Softwarebewegung zahlreiche Begriffe zur Benennung von Klassen innerhalb oder außerhalb des Phänomens entwickelt haben. Diese bezeichnen zum einen nichtäquivalente Kategorien, die sich im Grad der Freiheiten unterscheiden, die sie dem Nutzer der Software gewähren. Zum anderen existieren auch unterschiedliche Formulierungen zur Bezeichnung derselben oder zumindest ähnlicher Klassen. Im Rahmen dieser Arbeit wird der Begriff *„Open Source Software“* verwendet, um Software zu kennzeichnen, deren Lizenz entweder von der *Free Software Foundation (FSF)*[7] oder der *Open Source Initiative (OSI)*[8] als freie Softwarelizenz eingestuft wird. Alternativ werden die Bezeichnungen *„offene Software“*, *„quelloffene Software“* oder *„freie Software“* verwendet.

Im Folgenden werden die wichtigsten Begriffe im Zusammenhang mit Open Source Software genannt und kurz erläutert. Abbildung 3 zeigt dazu eine Übersicht über die wichtigsten Kategorien Freier und Nicht-Freier Software entsprechend der FSF.[9]

---

[7] von Richard Stallman gegründete, gemeinnützige Organisation zur Unterstützung der Freien Softwarebewegung

[8] von Eric Raymonds und Bruce Perens gegründete Organisation zur Vermarktung und Unterstützung der Open Source Softwarebewegung

[9] siehe http://www.gnu.org/philosophy/categories.html

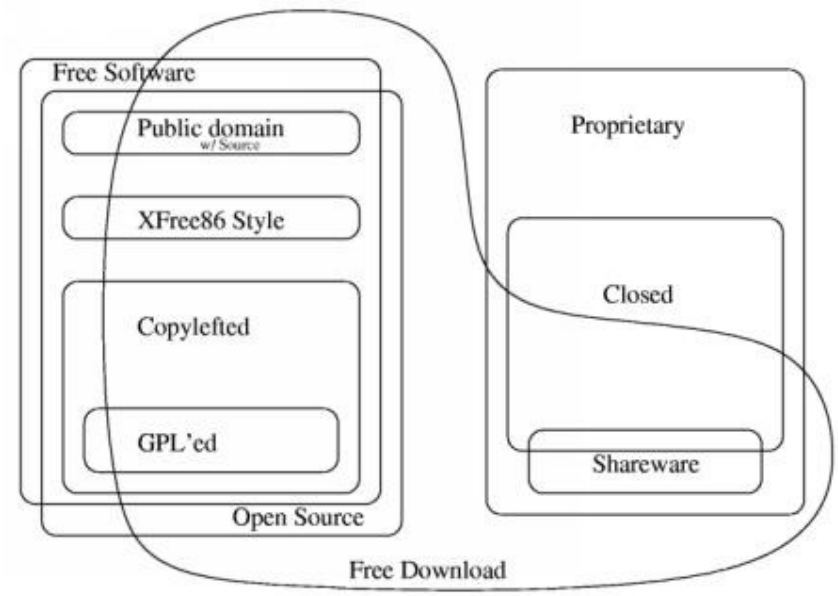

Abbildung 3: Kategorien Freier und Nicht-Freier Software nach der FSF

FLOSS

Die Abkürzung *FLOSS* steht für eine Zusammenfassung der Begriffe „Free Software“ und „Open Source Software“. Dabei bedeutet FLOSS *„Free/Libre/Open Source Software“*.

FOSS

Die Abkürzung *FOSS* steht für *„Free and Open Source Software“*. Der Begriff wurde vom U.S. Department of Defense als eine Zusammenfassung der Bezeichnungen „Free Software“ der FSF und „Open Source Software“ der OSI eingeführt.

Freeware

Freeware bezeichnet im Allgemeinen Software, die zwar weiterverbreitet, aber nicht modifiziert werden darf. Darüber hinaus steht der Quellcode des Programms in der Regel nicht zur Verfügung. Freeware ist dementsprechend keine Open Source Software im Sinne der FSF bzw. der OSI.[10]

Freie Software

Der Begriff *„Freie Software“* ist das Pendant der FSF zur Bezeichnung „Open Source Software“ der OSI. Genau wie Open Source Software garantiert auch Freie Software dem Benutzer die uneingeschränkte Nutzung, Vervielfältigung sowie die Modifikation und die Verbreitung modifizierter Versionen der Software. Im Gegensatz zur OSI stellt die FSF in ihrer Definition dabei allerdings stärker den Gedanken der Sicherung der

[10] siehe http://www.gnu.org/philosophy/categories.html

Freiheiten in den Vordergrund. Dies führt dazu, dass einige Lizenzen zwar von der OSI als Open Source Lizenzen anerkannt werden, die FSF diese allerdings nicht als Freie Softwarelizenzen anerkennt und umgekehrt.[12]

Open Source Software

Open Source Software ist Software, deren Lizenz dem Nutzer drei grundlegende Freiheiten garantiert – die uneingeschränkte Nutzung, das Kopieren sowie die Weiterverbreitung. Letzteres kann in veränderter oder unveränderter Form erfolgen. Für eine Verbreitung der Software dürfen dabei keine Lizenzgebühren erhoben werden. Dies schließt jedoch andere Formen von Gebühren nicht aus. Die Bezeichnung „Open Source Software“ stammt dabei aus dem Jahr 1998 und ist das Ergebnis einer Sitzung führender Köpfe der freien Softwarebewegung, die zur besseren Vermarktung den ideologisch vorbelasteten Begriff „Free Software“ („Freie Software“) ersetzen wollten.[11]

Proprietäre Software

Proprietäre Software bezeichnet das Gegenstück zu Open Source Software. Ihre Nutzung erfolgt nur gegen die Entrichtung einer Lizenzgebühr, eine Weitergabe oder Veränderung der Software ist verboten. Darüber hinaus steht der Quellcode des Programms in der Regel nicht zur Verfügung.[12]

Public Domain Software

Der Begriff *„Public Domain Software“* bezeichnet Software, die nicht unter einem Copyright steht. Diese kann „frei“ im Sinne der Definition der FSF sein, falls der Quellcode der Software offen verfügbar ist. Es ist jedoch auch möglich, dass Public Domain Software lediglich aus einem *Executable*[13] besteht, ohne dass der dazugehörige Quellcode zur Verfügung steht. In Deutschland ist die Verbreitung einer Software als Public Domain Software aus urheberrechtlichen Gründen nicht möglich.[12]

Shareware

Shareware gestattet die kostenlose Nutzung der Software für einen bestimmten Zeitraum. Dies beinhaltet zwar das Recht der Verbreitung der Software, allerdings ist jede Person, die die Software über den festgelegten Zeitraum hinaus nutzen möchte, zur Zahlung einer Lizenzgebühr verpflichtet. Darüber hinaus steht der Quellcode des

---

[11] siehe http://www.gnu.org/philosophy/categories.html und http://www.opensource.org/docs/osd

[12] siehe http://www.gnu.org/philosophy/categories.html

[13] Ein „Executable“ ist eine Datei, die in der Regel in Binärform vorliegt und direkt ausgeführt werden kann

Programms im Allgemeinen nicht zur Verfügung. Shareware ist keine Open Source Software.[12]

4.3.2 Rollen

Um die Prozesse in Open Source Projekten verstehen zu können, ist es wichtig, ein grundsätzliches Verständnis vom Aufbau der Communities der Projekte zu besitzen. In den meisten Projekten existiert dabei eine kleine Gruppe von Personen, die hauptverantwortlich für den Entwicklungsprozess der Software sind. Sie werden als *Kernentwickler (engl. „Core Developers“)* bezeichnet und nehmen in der Regel auch eine zentrale Rolle in der Administration der Projekte ein. Um diese Gruppe herum existieren weiterhin die so genannten *Co-Entwickler (engl. „Co-Developers“)*, die vor allem in den Bereichen Fehlerkorrektur und Programmerweiterungen tätig sind, und die eigentlichen Endnutzer der Software.

Crowsten und Howison [Crowsten2005] sowie Jensen und Scacchi [Jensen2007] vergleichen den Aufbau einer Open Source Community dabei mit dem einer Zwiebel. Ausgehend von der kleinen Gruppe der Kernentwickler, die am stärksten in den Entwicklungsprozess der Software involviert ist und das meiste Wissen über sie vereint, folgen die beiden weiteren Gruppen, die sich jeweils dadurch auszeichnen, dass ihr Wissen über den Kern der Anwendung sowie der Grad ihrer Involvierung im Projekt abnimmt, die Größe der Gruppe dagegen um jeweils eine Größenordnung zunimmt. So entsteht eine schichtweise aufgebaute Struktur der Community (sieh Abb. 4) [Crowsten2005].

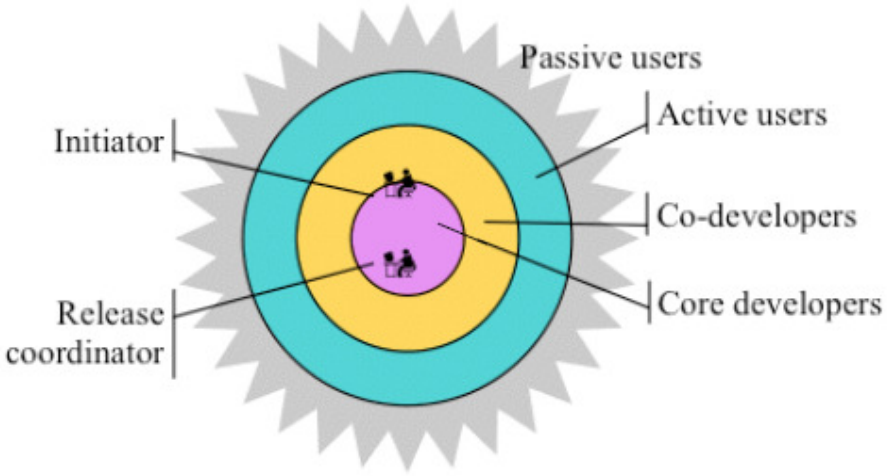

Abbildung 4: Struktur einer Open Source Community nach Crowsten und Howison

Neben den drei genannten Rollen bestehen innerhalb der jeweiligen Gruppen darüber hinaus spezifische Rollen, die in der Regel mit speziellen Aufgaben verbunden sind. Im

Folgenden werden sowohl die beschriebenen allgemeinen wie auch einige speziellere Rollen kurz erläutert, die in Open Source Projekten existieren.

Wichtig zum Verständnis der verschiedenen Kategorien ist, dass in unterschiedlichen Open Source Projekten teilweise auseinander gehende Bezeichnungen zur Benennung derselben Rollen existieren. Darüber hinaus gibt es vor allem in größeren Projekten spezifischere Rollen, die über den Rahmen dieser allgemeinen Kategorisierung hinausgehen.[14]

Co-Developer (Co-Entwickler)

Die Gruppe der „*Co-Developers*“ ist im Allgemeinen nicht mit der Weiterentwicklung des Kerns einer Open Source Anwendung beschäftigt. Ihre Aufgaben bestehen eher im Testen der Software, in der Entwicklung von *Patches*[15] für bestehende Programmfehler sowie in der Implementierung von Erweiterungen der Software und möglichen *Plug-Ins*[16]. Darüber hinaus unterstützen sie die Kernentwickler durch die Beantwortung von Fragen der Benutzer auf den Mailinglisten.

Contributor (Teilnehmer)

Mit dem Begriff *„Contributor“* werden alle aktiv in einem Open Source Projekt beteiligten Personen bezeichnet. Dies schließt neben den Kernentwicklern der Software auch die Nicht-Kernentwickler (oder Co-Entwickler) mit ein sowie aktive Benutzer, die Aufgaben wie das Testen der Software oder das Melden von Fehlern und *Feature Requests*[17] erledigen [Stuermer2005].

Core Developer (Kernentwickler)

Die Gruppe der „*Core Developers*“ besteht aus den Entwicklern der Community, die den Großteil des Wissens zur Projektsoftware unter sich vereinigen. Sie sind in der Regel die aktivsten Entwickler in Open Source Projekten und somit hauptverantwortlich für die Weiterentwicklung der Anwendungen. In erfolgreichen Open Source Projekten wie dem *Apache Webbrowser* umfasst diese Gruppe etwa 10-15 Entwickler. Ist die Anzahl ihrer Mitglieder größer, so entwickeln sich voraussichtlich Unterprojekte, um den stark ansteigenden Kommunikationsaufwand zu reduzieren [Mockus2002].

---

[14] siehe z.B. http://argouml.tigris.org/members/project_members.html

[15] Mit dem Begriff „Patches“ werden Korrekturen zu entdeckten Softwarefehlern bezeichnet

[16] „Plug-Ins“ sind funktionale Erweiterungen einer Software. Voraussetzung dafür ist eine sehr modulare Architektur, die so genannte Plug-In-Architektur, die versucht, den Kern einer Anwendung möglichst stark von den Erweiterungen zu trennen

[17] Der Begriff „Feature Request“ beschreibt das Melden zusätzlicher funktionaler Anforderungen an eine Software

Allerdings besitzt nur eine kleine Zahl von Open Source Projekten eine vergleichbar hohe Anzahl von Kernentwicklern.[18]

Developer (Entwickler)

Im Rahmen der Arbeit wird die Bezeichnung *„Developer“* bzw. *„Entwickler“* allgemein für eine Zusammenfassung der beiden Gruppen der „Kernentwickler“ sowie der „Co-Entwickler“ verwendet.

Initiator (Initiator)

Der *Initiator* eines Open Source Projects ist die Person, die den initialen Quellcode der Software geschrieben und diesen zur Weiterentwicklung veröffentlicht hat [Stuermer2005].

Project Administrator (Projektadministrator)

Der *Project Administrator* ist die hauptverantwortliche Person für die Organisation des Entwicklungsprozesses in Open Source Projekten. Größere Projekte besitzen häufig auch mehrere Administratoren, die sich die Verantwortung teilen. Meist gehören diese zum Kreis der Kernentwickler dazu. In einigen Open Source Projekten kann es jedoch vorkommen, dass den Projektadministratoren lediglich administrative Aufgaben wie die Planung der Entwicklungsrichtung, die Aufgabenzuteilung, das Marketing etc. zukommen.[19]

Release Manager

Der *Release Manager* (oder *Release Coordinator*) ist verantwortlich für die Planung und Zusammenstellung der stabilen Versionen einer Software. Er benötigt sowohl ein solides Wissen über die verschiedenen Module wie auch eine gewisse Autorität unter den restlichen Entwicklern, um die Einhaltung von Fristen und die Begrenzung von Funktionalitäten in der Community durchsetzen zu können. In der Regel gehört der Release Manager zu den Kernentwicklern eines Open Source Projektes.

User (Benutzer)

Users sind die Endnutzer einer Software. Crowsten und Howison [Crowsten2005], Jensen und Scacchi [Jensen2007] sowie Stürmer [Stuermer2005] unterscheiden innerhalb dieser Gruppe noch einmal zwischen aktiven Benutzern *(engl. „Active Users“)*, die Beiträge für das Projekt in Form von Feature Requests, Fehlermeldungen

---

[18] siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

[19] siehe z.B. das Open Source Projekt „Freeguide TV Guide“
http://sourceforge.net/mailarchive/message.php?msg_name=4564A005.9030809%40artificialworlds.net

oder Fehlerkorrekturen leisten, und passiven Benutzern der Software *(engl. „Passive Users“).*

4.3.3 Infrastrukturkomponenten

Zusätzlich zur Anordnung der beteiligten Gruppen in Open Source Projekten bildet auch ihre Infrastruktur eine spezifische Besonderheit. Da die Entwickler einer Open Source Community in der Regel geographisch verteilt agieren und somit keinen direkten Kontakt zueinander haben, benötigen Open Source Projekte spezielle Infrastrukturkomponenten zur Umsetzung einer gemeinschaftlichen und kollaborativen Arbeitsweise. Dazu zählen zum einen Werkzeuge, die direkt mit der Entwicklung und Verteilung des Quellcodes in Verbindung stehen. Zum anderen sind aber auch spezielle Kommunikationsmechanismen erforderlich, um den Entwicklungsprozess zu koordinieren. In der Folge werden die wichtigsten Komponenten einer Infrastruktur von Open Source Projekten genannt und kurz erläutert.

Blog

Ein *Weblog* oder kurz *Blog* ist eine Webanwendung zur Verwaltung von *Postings*[20] zu verschiedenen Themen. Ursprünglich diente ein Blog zur Präsentation persönlicher Informationen, kann aber beispielsweise auch als ein Medium zur Diskussion von Themengebieten zu Open Source Projekten genutzt werden [Stuermer2005].

Bug-Tracker

Ein *Bug-Tracker* ist ein Programm zur Steuerung des Prozesses zur Behebung von Programmfehlern. Es ermöglicht die Speicherung von Fehlerberichten, Statusinformationen und Kommentaren und bildet somit die Grundlage für einen verteilten Qualitätssicherungsprozess. Darüber hinaus werden neben Programmfehlern häufig auch Feature Requests innerhalb des Bug-Trackers verwaltet.

Concurrent Version System (CVS)

CVS oder *Concurrent Version System* ist eine Software zur Verwaltung verschiedener Dateiversionen. Die Benutzung von CVS ist in Open Source Projekten weit verbreitet, um die verschiedenen Versionen einer Software zu organisieren und gleichzeitig den Zugriff auf den Quellcode zu ermöglichen.

---

[20] Mit dem Begriff „Postings“ werden Eintragungen in Internetforen oder ähnlichen elektronischen Kommunikationsmedien bezeichnet

Internet Relay Chat (IRC)

Der *Internet Relay Chat* ist ein textbasiertes Chat-System, das sowohl 1-zu-1-Kommunikation wie auch Gruppenkommunikation unterstützt. Der Vorteil des Systems gegenüber anderen Chat-Anwendungen besteht in der Verteilung der Belastung auf mehrere Server. Dadurch wird der Internet Relay Chat sehr stabil und leistungsfähig. Zur Verbindung mit den entsprechenden Servern wird ein so genannter *IRC-Client* benötigt.

Mailinglisten

Eine *Mailingliste* ist eine Sammlung von Namen und Adressen, die vergleichbar zu einem Adressbuch zum Versenden bestimmter Nachrichten benutzt wird. Eine Nachricht wird dabei immer automatisch an alle Mitglieder der jeweiligen Liste geschickt. Darüber hinaus werden die gesendeten Mails häufig in Mail-Archiven zentral von den Projekten gespeichert, um die Diskussionen auch für Nicht-Mitglieder sichtbar zu machen. In der Regel bilden Mailinglisten das Herzstück der Kommunikation in Open Source Projekten [Stuermer2005].

Subversion (SVN)

Genau wie CVS ist auch *SVN* oder *Subversion* ein System zur Versionsverwaltung von Dateien. Im Vergleich zu CVS besitzt SVN ein überarbeitetes Versionsschema und bietet noch einige andere Vorteile, weshalb viele Projekte ihre Infrastruktur auf SVN umstellen.

Wiki

Ein *Wiki* ist eine Kollektion von Webseiten, die von ihren Benutzern nicht nur gelesen, sondern auch online geändert werden können. Dies ermöglicht eine gemeinschaftliche Bearbeitung der Seiten innerhalb der Community. Ein Einsatzgebiet für ein Wiki ist beispielsweise die Softwaredokumentation, da diese regelmäßig aktualisiert werden muss. Um ein Wiki benutzen zu können, bedarf es einer bestimmten Software, einer so genannten *Wiki-Engine*.

4.3.4 Lebenszyklus

Betrachtet man die Software erfolgreicher Open Source Projekte, so ist diese häufig durch eine hohe Qualität gekennzeichnet [Wheeler2005][Raymond2001][Mockus2002]. Hinter der Software stehen dabei meist zu traditionellen Entwicklerteams vergleichsweise große Communities, bestehend aus weltweit verteilten freiwilligen Entwicklern und aktiven Benutzern, die trotz der Freiwilligkeit der Beteiligung in einem produktiven Entwicklungsprozess organisiert sind. Allerdings trifft diese These bei weitem nicht für alle Open Source Projekte zu, sondern sogar nur für einen kleinen

Teil[21], sodass sich die Frage stellt, welche Entwicklungen haben die durch Erfolg gekennzeichneten Projekte hinter sich bzw. mit welchen Herausforderungen ist der von ihnen beschrittene Weg behaftet?

Um diese Frage zu beleuchten, soll nun der prototypische Lebenszyklus von Open Source Projekten erläutert werden. Dabei bezieht sich die Beschreibung auf den gesamten Zeitraum von der Identifikation des Bedarfes an einer Softwarelösung bis hin zur Etablierung eines reifen Projektes, das über gefestigte Strukturen verfügt. Obwohl eine solche Entwicklung sicherlich teilweise auch projektspezifisch ist, lässt sich doch ein allgemeines Entwicklungsschema der meisten community-begründeten[22] Open Source Projekte ausmachen.

Michlmayr und Senyard [Michlmayr2004] identifizieren dazu drei Phasen im Lebenszyklus von Open Source Projekten. Diese bezeichnen sie mit den Begriffen *„Cathedral Phase“*, *„Transition Phase“* und *„Bazaar Phase“*. Die Bezeichnungen der ersten und dritten Phase entsprechen dabei denen, die Eric Raymond für die Gegenüberstellung des traditionellen, geschlossenen Entwicklungsprozesses der meisten Softwareunternehmen und des offenen Entwicklungsprozesses der Open Source Projekte wählt [Raymond2001]. Dies liegt daran, dass Michlmayr und Senyard, ähnlich wie Raymond selbst, auch im Lebenszyklus von Open Source Projekten eine initiale Phase identifizieren, die dem traditionell geschlossenen Entwicklungsprozess folgt.

4.3.4.1 Cathedral Phase

Bevor weltweite Kollaborationen zur Entwicklung von Softwareanwendungen möglich werden, ist es zunächst einmal erforderlich, die Grundlagen für die Ausbildung einer Open Source Community zu schaffen.

Entwicklung des Prototyps

Dazu gehört in besonderem Maße die Entwicklung eines ersten Prototyps der Software. Dieser kann jedoch nicht im Zuge eines offenen Prozesses entwickelt werden, da zu diesem Zeitpunkt noch keine Entwicklergemeinschaft um das Projekt herum etabliert ist. Folglich ist es die Aufgabe der Projektinitiatoren, die erste Implementierung der Software zu realisieren. Die Gruppe der Projektinitiatoren besteht dabei zunächst häufig aus Einzelpersonen oder kleineren Entwicklergruppen, zunehmend initiieren aber auch Unternehmen eigene Open Source Projekte.[22] Die Wichtigkeit eines gelungenen Prototyps der Software für die weitere Entwicklung des Projektes wird deutlich, führt

---

[21] siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

[22] im Gegensatz zu unternehmensbegründeten OSPs, siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

man sich vor Augen, dass er die Basis für die Rekrutierung neuer Entwickler in der späteren offenen Entwicklungsphase bildet. Dies bestätigt Raymond insofern, als dass er betont, dass die initiale Implementierung der Software potentiellen Entwicklern die Aussicht auf einen zukünftigen Erfolg des Open Source Projektes liefern muss [Raymond2001]. Michlmayr und Senyard gehen sogar noch einen Schritt weiter und beschreiben die Anforderungen an den Prototyp und die Herausforderungen, die bereits in der Cathedral Phase bewältigt werden müssen, genauer. Dazu unterteilen sie die Phase in fünf Subphasen [Michlmayr2004].

Die Erste davon beschreibt zunächst schlicht den Bedarf an einer Softwarelösung für ein bestimmtes Problem. Ist ein solcher Bedarf vorhanden, erfolgt in der zweiten Phase die Suche nach bestehenden Open Source Projekten, die bereits eine Lösung für die spezifische Problemstellung anbieten. An dieser Stelle existieren zwei Möglichkeiten.

Wird ein Projekt gefunden, dessen Software bereits den gewünschten Funktionsumfang besitzt oder das als adäquat erachtet wird, eigene Anstrengungen in seine Weiterentwicklung zu investieren, werden die Bemühungen zur Gründung eines neuen Open Source Projektes eingestellt. Ist die Suche dagegen erfolglos, so kann in der dritten Phase die Initiierung eines eigenen Projektes beginnen.

Diese ist gekennzeichnet durch die Durchführung einer Risikoanalyse zum Potential des zu initiierenden Projektes. Eine solche Analyse sollte im Wesentlichen drei Aspekte berücksichtigen

- die Motivation der Projektinitiatoren zu einer langfristigen Beteiligung,
- die Konkurrenzfähigkeit im Vergleich zu bestehenden Projekten,
- die Herausforderung, die es potentiellen Neuentwicklern bietet.

Zusätzlich ist es von Vorteil, wenn bereits ein grober Zeitplan für die Entwicklung des Prototyps besteht, der unter den Projektinitiatoren kommuniziert werden kann [Michlmayr2004].

Im Fall einer positiven Bewertung der Risikoanalyse folgt der Übergang zur eigentlichen Entwicklungsphase des Prototyps. Diese lässt sich traditionell in die Unterphasen *Anforderungsbestimmung, Designentwurf, Implementierungsphase* und *Testphase* unterteilen. Wichtig beim Entwurf des Prototyps ist für seine spätere Weiterentwicklung innerhalb eines offenen Prozesses vor allem ein modulares Design mit klar definierten Schnittstellen, um es potentiellen Neuentwicklern zu ermöglichen, Änderungen an der Software vorzunehmen, ohne den Kern der Anwendung zu kennen.

Darüber hinaus bilden die Etablierung von Codierrichtlinien, um den Programmcode übersichtlicher zu gestalten, sowie die Wahl der Programmiersprache wichtige

Instrumente zur Steuerung der Programmentwicklung. Stürmer schreibt zum letzten Aspekt:

> *"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."*
>
> *- Stürmer 2005 [Stuermer2005, S.44]*

Nach der Fertigstellung des Prototyps am Ende der Cathedral Phase erfolgt eine erste Verbreitung der Software. Diese kann per Email oder auf einer Webseite erfolgen und dient zunächst nur der Rückkoppelung mit anderen Entwicklern.

4.3.4.2 Transition Phase

Die Ziele der Transition Phase sind im Wesentlichen bestimmt durch den Aufbau einer Infrastruktur, die eine verteilte Weiterentwicklung der Software innerhalb einer offenen Gemeinschaft von Entwicklern ermöglicht, sowie durch die Positionierung zu grundlegenden Entscheidungen, die die zukünftigen Prozesse innerhalb des Projektes beeinflussen.

Aufbau der Infrastruktur

Zum Aufbau der Infrastruktur eines Open Source Projektes existieren zwei grundlegende Vorgehensweisen. Die erste besteht darin, die notwendigen Komponenten manuell auf einem projekteigenen Serverrechner einzurichten. Dabei gehören vor allem ein leicht zugänglicher Mechanismus zur Veröffentlichung des Softwarequellcodes sowie ein oder mehrere Kommunikationskanäle als Plattform für Interaktionen der Benutzer und Entwickler zu den erforderlichen Bestandteilen der Infrastruktur eines Open Source Projektes.[23]

Zur Verteilung des Softwarequellcodes haben sich dabei in Open Source Projekten im Wesentlichen Versionsverwaltungssysteme wie CVS oder SVN etabliert. Diese bieten den Vorteil, dass sie zum einen den Zugriff auf den Quellcode, zum anderen aber auch die Integration von Quellcodeänderungen ermöglichen. Dies vereinfacht die verteilte Weiterentwicklung des Prototyps aus der Cathedral Phase in der anschließenden offenen Entwicklungsphase.

---

[23] siehe Kapitel 4.3.3 „Infrastrukturkomponenten"

Die Kommunikation in Open Source Projekten dagegen findet üblicherweise über Mailinglisten statt. In ihnen werden alle aktuellen Fragen, Probleme und Themen diskutiert, sodass getroffene Entscheidungen sowie geplante Vorgehensweisen für alle Mitglieder des Projektes transparent sind.

Dies erfordert allerdings eine Neuorientierung vor allem innerhalb der Gruppe der Projektinitiatoren. Während diese im Rahmen der Cathedral Phase auftretende Probleme und Fragen lediglich untereinander diskutieren mussten, so macht die Öffnung des Projektes in der nachfolgenden Bazaar Phase auch die Öffnung des Kommunikationsprozesses erforderlich. Wechner, Initiator und Kernentwickler des *Lenya-Projektes*, betont dieses Problem im Zusammenhang mit der Öffnung des Lenya-Projektes durch die Firma Wyona. Viele Mitarbeiter des Unternehmens waren die offene Kommunikation über Mailinglisten nicht gewohnt, so dass es einige Zeit dauerte, sie von der Notwendigkeit eines transparenten Kommunikationsprozesses zu überzeugen [Stuermer2005].

Zusätzlich zu Mailinglisten zählen der Internet Relay Chat (IRC), Blogs, Foren oder Wikis als weitere Kommunikationsmedien sowie Bug-Tracker zur Verwaltung von Fehlern und Defekten einer Software zu den etablierten Bestandteilen der Infrastruktur vieler offener Softwareprojekte.

Neben der manuellen Zusammenstellung der projektinternen Infrastruktur besteht die zweite Möglichkeit der Zusammenstellung in der Nutzung öffentlicher Hosting-Sites[24] wie *Sourceforge.net* [25] . Diese bieten Open Source Projekten bereits fertige Infrastrukturen an und bilden somit vor allem für junge Projekte eine gute Kollaborationsplattform. Allerdings ist der Rückgriff auf derartige Portale auch mit Nachteilen verbunden. So leidet Sourceforge beispielsweise an einem Mangel an Geschwindigkeit sowie fehlender Individualisierbarkeit [Stuermer2005], sodass große Open Source Projekte ihre Infrastrukturen häufig selbst organisieren.

Etablierung eines Managementstils

Ist der Aufbau der Infrastruktur abgeschlossen, gibt es in der Transition Phase zwei weitere Entscheidungen, die von den Projektinitiatoren getroffen werden müssen und die die Zukunft des Projektes in der anschließenden Bazaar Phase entscheidend beeinflussen. Dies ist zum einen die Art des Führungsstils und zum anderen die Wahl der entsprechenden Open Source Lizenz [Michlmayr2004].

---

[24] Hosting-Sites sind Internetportale, die die Entwickler von Open Source Projekten unterstützen, indem sie eine Infrastruktur zur verteilten Softwareentwicklung zur Verfügung stellen

[25] siehe www.sourceforge.net

Die Etablierung eines Führungs- oder Managementstils ist dabei eine besondere Herausforderung. Während in der Cathedral Phase die Kompetenzen und Befugnisse der Projektinitiatoren klar abgesteckt waren, muss die zentrale Kontrolle über das Projekt für einen Übergang in die Bazaar Phase gelockert werden, um potentiellen Neumitgliedern die Chance zu geben, die Geschehnisse selbst zu beeinflussen. Wie weit reichend die Entscheidungsprozesse im Projekt dabei demokratisiert werden, hängt von mehreren Faktoren ab.

Der erste Faktor besteht dabei in der Art der Anwendung bzw. damit verbunden im Einsatzgebiet der Software. Zur Verdeutlichung des Aspekts stellen Michlmayr und Senyard das durch Linus Torvalds stark zentral regulierte *Linux-Projekt* und das sehr offene *KDE-Projekt* gegenüber [Michlmayr2004]. Die Gründe für die unterschiedlichen Führungsstile der beiden Projekte liegen in den abweichenden Anforderungen an die Projektsoftware begründet, die wiederum mit den unterschiedlichen Einsatzgebieten verbunden sind. Während das Linux-Projekt an der Erstellung eines Systemkerns arbeitet, dessen Kernanforderungen Stabilität, Kompaktheit und Performanz sind, und somit eine Limitierung sowie eine strenge Kontrolle der eingehenden Quellcodeänderungen zwingend erforderlich ist, stellt eine Vielzahl von neu eingehenden Features für eine graphische Benutzeroberfläche wie KDE sogar eine Stärke dar.

Der zweite Grund für unterschiedliche Führungsstile in Open Source Projekten besteht in der Historie der Projekte. Jensen und Scacchi beschreiben in ihrer Arbeit über Rollenmigration und Aufstiegsprozesse in Open Source Projekten den Weg eines Entwicklers vom einfachen Nutzer bis hin zum Kernentwickler eines Open Source Projektes [Jensen2007]. Dies geschieht am Beispiel des *Mozilla-, des Apache-* und des *Netbeans-Projektes*. Dabei identifizieren sie im Netbeans-Projekt bestimmte Rollen, die nicht im Zuge eines normalen Aufstieges innerhalb des Projektes erreichbar sind. Dies sind spezielle Management-Positionen, die nur Mitarbeiter der Firma Sun einnehmen können. Der Grund dafür besteht darin, dass das Netbeans-Projekt im Gegensatz zu den meisten anderen Open Source Projekten nicht durch eine Gemeinschaft freiwilliger Entwickler gegründet, sondern von der Firma Sun ins Leben gerufen wurde.

Dieses Modell der Initiierung von Open Source Projekten hat sich für Unternehmen etabliert, um externe Entwickler zu gewinnen, am Support für die Software Geld zu verdienen, sich Marketingvorteile zu verschaffen oder Bereiche auszulagern, die nicht den Kern des Unternehmens betreffen [O'Mahony2005].[26] Charakteristisch dabei ist jedoch, dass die entsprechenden Unternehmen trotzdem weiterhin Einfluss auf die

[26] siehe http://www.opensource.org/advocacy/case_for_business.php oder Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen"

Entwicklung der Software im Open Source Projekt haben wollen, sodass eine Projektinitiierung durch Unternehmen häufig mit einer größeren Kontrolle durch die verantwortlichen Personen einhergeht. Weitere Beispiele für solche Projektinitiierungen sind das *Eclipse-Projekt* durch die Firma IBM und das *OpenOffice-Projekt* ebenfalls durch die Firma Sun [O'Mahony2005].

Der dritte Grund für die Wahl eines mehr oder weniger stark zentralisierten Führungsstils sind schlicht Vorlieben der Projektinitiatoren, inwiefern sie auch nach der Freigabe des initialen Quellcodes die Geschehnisse im Projekt noch entscheidend prägen wollen. Zu achten ist dabei allerdings darauf, potentielle Neumitglieder mit einem zu stark zentralisierten Führungsstil nicht abzuschrecken.

Wahl der Open Source Lizenz

Die Wahl der Open Source Lizenz ist die letzte wichtige Entscheidung, die in der Transition Phase vor dem Einstieg in die Bazaar Phase getroffen werden muss. Diese kann insofern einen großen Einfluss auf die zukünftige Entwicklung des Projektes haben, als dass sie erstens die Möglichkeiten einer kommerziellen Vermarktung der Software tangiert und zweitens auch direkt den Kreis der potentiellen Entwickler beeinflusst.

Eine grobe Unterscheidung teilt dabei die Lizenzarten in Lizenzen mit einem *Copyleft* [27] wie die *GPL* und Lizenzen ohne ein solches. Ein Copyleft bringt den Vorteil mit sich, dass Entwickler, die die Software weiterentwickeln, ihre Änderungen mit der Community teilen müssen, da eine Veröffentlichung modifizierter Softwareversionen zu denselben Bedingungen erfolgen muss. Der Nachteil besteht allerdings darin, dass kommerzielle Unternehmen häufig abgeschreckt werden, da sie das Risiko eingehen, bei einer Zusammenstellung von Software unter einem Copyleft und eigener Software auch den Quellcode der eigenen Anwendung offen legen zu müssen.

Darüber hinaus wird der Kreis der potentiellen Neuentwickler im Projekt durch die Wahl der Open Source Lizenz insofern beeinflusst, als dass verschiedene Entwickler verschiedene Vorlieben besitzen. So werden vor allem die Anhänger der Freien Softwarebewegung nach Richard Stallman auf die Verwendung der GPL drängen, da sie die gewährten Freiheiten auch für Modifikationen der Software gesichert wissen möchten.[28] Zudem spielt auch die Popularität der Lizenz eine wichtige Rolle für die Gewinnung neuer Projektmitglieder, wie Guido Wesdorp, einer der Kernentwickler des *Kupu-Projektes*, in einem Interview bestätigt [Stuermer2005].

---

[27] Von Richard Stallman eingeführtes Konzept zum Schutz der Freiheitsrechte an freier Software, siehe Kapitel 4.3.5 „Lizenzen"

[28] siehe Kapitel 4.3.6 „Geschichte"

4.3.4.3 Bazaar Phase

Sind die Voraussetzungen für einen offenen Entwicklungsprozess durch die Cathedral Phase und die Transition Phase erfüllt, so erfolgt der Übergang in die Bazaar Phase. Dies ist die Phase, in der sich das Projekt erstmalig als Open Source Projekt präsentiert und in der die Entwicklung der Software schließlich verteilt innerhalb einer offenen Gemeinschaft von Entwicklern erfolgt. Allerdings gelingt es nur einem kleinen Teil von Open Source Projekten, eine größere Anzahl von Entwicklern um sich zu vereinen [Krishnam2002][Healy2003]. Dies bestätigt zum einen die Wichtigkeit der Aktivitäten in den beiden vorhergehenden Phasen, zum anderen widerlegt es auch den weit verbreiteten Mythos, dass allein die Initiierung eines Open Source Projektes mit einer großen Anzahl von Entwicklern einhergeht [Gorling2003].

Zeitpunkt der Offenlegung des Quellcodes

Ob ein Open Source Projekt die kritische Startphase übersteht, hängt von zahlreichen Faktoren ab. Einer dieser Faktoren ist der Zeitpunkt des Einstiegs in die Bazaar Phase. Dies hängt mit der Reife des Prototyps der Software zu diesem Zeitpunkt zusammen. Um potentielle Neuentwickler in ein Projekt zu locken, sollte die initiale Implementierung der Anwendung einen zukünftigen Erfolg in Aussicht stellen [Raymond2001].

Dabei gilt es abzuwägen. Ist die Qualität des Prototyps zu niedrig, so ist das Potential der Software für neue Entwickler nicht erkennbar und der Aufwand zur Erstellung einer ersten stabilen Version zu groß [Michlmayr2004]. Ist die Komplexität allerdings bereits zu hoch, so ist es erstens sehr schwer, sich in das Programm einzuarbeiten [O'Mahony2005]. Zweitens bietet eine reifere Software weniger Möglichkeiten für neue Entwickler, die Entwicklungsrichtung entscheidend zu prägen und sich somit einen Ruf innerhalb der Projektcommunity zu verschaffen.[29]

Darüber hinaus spielt der Moment des Einstiegs in die Bazaar Phase eine zweite wichtige Rolle. Damit die Software von einer breiten Gruppe von Benutzern angenommen wird, muss zu diesem Zeitpunkt ein grundsätzliches Interesse an der Problemstellung vorhanden sein. Guido Wesdorp, einer der Kernentwickler im Kupu-Projekt, berichtet dazu in einem Interview:

> *„It's so hard to say what people find interesting. […] It's also something dependent on trends. It's very hard to say if a project is going to be successful or not."*
>
> *- Guido Wesdorp [Stuermer2005, S.50]*

---

[29] Für die Motive der Entwickler, siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten"

Die Art der Problemstellung spielt zudem eine weitere wichtige Rolle, da sie auch direkt die Zusammensetzung der Benutzergruppe der Software beeinflusst. Dies ist insofern ein wesentliches Kriterium, da es einfacher ist, eine aktive Entwicklergemeinschaft um eine Software zu etablieren, deren Zielgruppe aus Entwicklern besteht, als um eine reine Endnutzeranwendung [Stuermer2005].

Aktivitäten der Projektinitiatoren

Abgesehen von den genannten schwer beeinflussbaren Faktoren, die die Entwicklung eines Open Source Projektes zu Beginn der Bazaar Phase bestimmen, hängt die erfolgreiche Etablierung einer Entwicklergemeinschaft zu einem großen Teil von den Aktivitäten, dem Einsatz und den Fähigkeiten der Projektinitiatoren ab.

Nachdem diese in den ersten beiden Phasen den Prototyp geschaffen sowie die Infrastruktur des Projektes aufgebaut haben, besteht ihre Hauptaufgabe am Anfang der Bazaar Phase in der Rekrutierung neuer Entwickler. Dies fordert insbesondere die kommunikativen und sozialen Fähigkeiten der Projektinitiatoren [Raymond2001][Rothfuss2002] und beinhaltet die Ankündigung des neuen Projektes in entsprechenden *Newsgroups*[30], die Kommunikation mit Benutzern und potentiellen Entwicklern der Software und die Erstellung von Dokumenten zur Unterstützung der Benutzer.

Eine besondere Stellung nimmt dabei die intensive Teilnahme an den Diskussionen in den Mailinglisten oder im Internet Relay Chat ein. Bevor auch neu integrierte Projektmitglieder Hilfestellungen und Anregungen in den Mailinglisten geben, sind zunächst die Projektinitiatoren die alleinigen Ansprechpartner [Stuermer2005]. Dabei beinhaltet die Partizipation an den Diskussionen nicht nur die Beantwortung von Fragen zur Software, sondern auch die Formulierung von Projektzielen sowie die Steuerung der Prozesse zur Ausbildung der Führungsstrukturen. Da diese Aktivitäten gerade in der Startphase eines Open Source Projektes sehr viel Zeit in Anspruch nehmen und aufgrund der nicht formalisierten Hierarchien stark an die sozialen Kompetenzen der Projektinitiatoren gekoppelt sind [Rothfuss2002], ist es nicht verwunderlich, dass nur wenige Projekte jemals eine aktive Community entwickeln [O'Mahony2005].

Hat das Projekt schließlich eine bestimmte Größe erreicht, so müssen Strukturen zur Organisation und Steuerung des Entwicklungsprozesses der Software ausgebildet werden. Dies ist notwendig, um sowohl die Konformität mit den formulierten Projektzielen als auch mit den qualitativen Anforderungen an eingehende Programmierarbeiten zu gewährleisten. Der Begriff *„Struktur“* steht in diesem Zusammenhang sowohl für indirekte Kontrollmechanismen wie Codierrichtlinien, *API-*

[30] „Newsgroups“ sind Foren im Internet zu unterschiedlichen Themen

Spezifikationen [31] oder Listen offener Features als auch für die Etablierung von Hierarchien zur direkten Kontrolle der Entwicklungen im Projekt. Auch wenn im Zusammenhang mit dem letzten Aspekt Open Source Projekten häufig lediglich flache Strukturen [32] zugesprochen werden [Raymond2001], so identifizieren Healy und Schussmann [Healy2003] sowie Jensen und Scacchi [Jensen2007] stark ausgeprägte Hierarchien in erfolgreichen Open Source Projekten wie dem Mozilla-, dem Apache- und dem wohl bekanntesten, dem Linux-Projekt. O'Mahony und West folgend haben reife Projekte eine Reihe von *Major Releases* [33] veröffentlicht, besitzen Aufnahmekriterien sowie Führungsmechanismen, die die Etablierung eines kommerziellen Zweiges ermöglichen. Außerdem kooperieren sie häufig mit gemeinnützigen oder kommerziellen Organisationen, die die Entwicklung der Software in den Projekten unterstützen [O'Mahony2005].

Schließlich helfen in die Zukunft gerichtete Maßnahmen wie ein funktionierender Schulungsprozess für neue Entwickler, ein System zur Akkreditierung aktiver Entwickler oder die Formulierung dauerhaft gültiger Projektnormen die Fortentwicklung des Projektes abzusichern.

4.3.5 Lizenzen

Ein weiterer wichtiger Aspekt zur Evaluierung von Open Source Projekten aus Unternehmenssicht ist die Betrachtung der juristischen Folgen, die sich aus einer Nutzung oder Verbreitung der Projektsoftware ergeben. Zur Wahrung der Interessen der Softwareurheber sowie zur Regelung der Rechte und Pflichten ihrer Nutzer haben sich in diesem Zusammenhang eine Vielzahl von Open Source Lizenzen etabliert, die die rechtliche Grundlage für die Verbreitung der unterschiedlichen Anwendungen bilden.

Die Lizenzen unterscheiden sich von traditionellen, kommerziellen Softwarelizenzen dabei in den grundlegenden Freiheiten, die sie den Nutzern der Software gewähren. Allerdings existieren auch innerhalb des Feldes der Open Source Lizenzen verschiedene Konzepte. So gibt es etwa 58 bzw. 66 unterschiedliche Lizenzen, die die zwei wesentlichen Organisationen im Bereich Open Source Software, die Free Software Foundation (FSF) und die Open Source Initiative (OSI), als „frei" deklarieren.[34] Jaeger und Metzger [Jaeger2002] unterteilen diese in vier wesentliche Lizenzkategorien sowie

---

[31] Eine „Application Programming Interface"-Spezifikation (API) ist eine Beschreibung von Programmierschnittstellen, die der Strukturierung von Programmierarbeiten dient

[32] Flache Hierarchien sind Unternehmensstrukturen mit nur wenigen Führungsebenen zwischen der untersten und der obersten Hierarchieebene

[33] spezielle, ausgezeichnete Versionen einer Software

[34] siehe http://www.fsf.org/licensing/licenses bzw. http://www.opensource.org/licenses/alphabetical

Lizenzen mit Sonderbestimmungen. Die vier Hauptkategorien sind dabei die *„Copyleft"-Lizenzen*, die *„BSD"-artigen Lizenzen*, die *„Mozilla"-artigen Lizenzen* und die *„Artistic"-Lizenzen*.

Im Folgenden werden aufgrund ihrer hohen Verbreitung und der Tatsache, dass sich viele der angeführten Konzepte auch in anderen Lizenzen widerspiegeln, zunächst die genauen Merkmale der zwei wesentlichen Lizenzen der Kategorie der „Copyleft"-Lizenzen – der *„General Public License" (GPL)* und der *„Lesser General Public License" (LGPL)* – erläutert. Danach erfolgt zusätzlich eine allgemeine Beschreibung der anderen Lizenzkategorien.

Die Ausführungen basieren dabei auf den Untersuchungen von Jaeger und Metzger [Jaeger2002]. Ein Überblick über die genauen Bestimmungen aller Lizenzkategorien befindet sich im Anhang.[35]

4.3.5.1 „Copyleft"-Lizenzen

Die so genannten „Copyleft"-Lizenzen zeichnen sich gegenüber anderen Lizenzarten vor allem durch die Verpflichtung der Nutzer aus, Veränderungen der Originalsoftware bei einer Veröffentlichung wieder zu den Bedingungen der Originallizenz zu verbreiten. Der Begriff der „Copyleft"-Lizenz stammt dabei von der Free Software Foundation (FSF), die die von ihr selbst erstellten Lizenzen unter diesem Begriff zusammenfasst. Die beiden populärsten Lizenzen dieser Kategorie sind zum einen die GNU General Public License (GPL) sowie zum anderen die modifizierte GNU Lesser General Public License (LGPL), die im Folgenden diskutiert werden.[36]

Die GNU General Public License (GPL)

Die GPL ist mit einer Verbreitung von etwa 66%[37] die weltweit am häufigsten eingesetzte Open Source Lizenz. Entwickelt wurde sie von Richard Stallman und Eben Moglan mit dem Ziel, das von Stallman entwickelte Betriebssystem GNU einer breiten Öffentlichkeit zugänglich zu machen und die gewährten Freiheiten gleichzeitig auch für Modifikationen der Software zu sichern.

Rechte der Nutzer

a) Einräumung eines einfachen Nutzungsrechts an jedermann

   Die GPL räumt jedem potentiellen Nutzer ein so genanntes einfaches Nutzungsrecht ein. Dieses erwirbt er urheberrechtlich betrachtet in jedem Fall

[35] siehe Anhang A

[36] siehe auch Anhang A.1

[37] siehe http://www.freshmeat.net/stats (Stand 4.4.2007)

direkt vom Urheber, auch wenn er die Software von Softwaredistributoren[38] erhält. Mit dem Erwerb des einfachen Nutzungsrechtes erhält der Nutzer die vom Urheber ausdrücklich genannten Verwertungsrechte sowie die Befugnis, die Software zu vervielfältigen, zu verbreiten, zu vermieten und sie verfügbar zu machen *(„Making-available"-Recht)*. Letzteres bezieht sich vor allem auf die Verbreitung der Software im Internet und ist rechtlich gesehen nicht mit der einfachen Verbreitung der Software gleichzusetzen, da diese in körperlicher Form erfolgen muss. Die Gewährung der genannten Rechte erfolgt dabei ohne die Entrichtung einer Lizenzgebühr. Dies bedeutet jedoch nicht, dass für den Vertrieb von Open Source Software keine Gebühren verlangt werden dürfen. Die GPL verbietet lediglich die Erhebung von Lizenzgebühren.

b) Embedded Nutzung

Da die GPL den Vertrieb von Software im Objektcode generell gestattet und die Lizenz auch keine Einschränkungen in Bezug auf das Medium der Verbreitung enthält, ist die Nutzung von GPL-Software auch im Bereich der *Embedded Systeme*[39] prinzipiell erlaubt. Allerdings ist sie in diesem Bereich gebunden an die Einhaltung der Bedingungen, die mit einer Verbreitung von GPL-Software im Objektcode verknüpft sind (siehe „Pflichten der Nutzer").

c) Veränderungsfreiheit

Grundsätzlich geht mit der Lizenzierung von Software unter einer Open Source Lizenz auch immer die Freiheit für den Nutzer einher, diese nach eigenen Vorstellungen zu verändern und die modifizierte Version auch zu veröffentlichen. Nach deutschem Recht ist diese Freiheit allerdings einer Einschränkung unterworfen. Mit der Lizenzierung der Software unter einer Open Source Lizenz tritt der Softwareurheber nicht seine Urheberpersönlichkeitsrechte ab. Dies bedeutet insbesondere, dass die Veröffentlichung einer Modifikation der Software nicht die Persönlichkeitsrechte des Urhebers verletzen darf. Dies kann beispielsweise der Fall sein, wenn die veränderte Software von geringer Qualität ist und somit dem Ruf des Urhebers schadet. In diesem Fall kann dieser auf die Einhaltung seiner Persönlichkeitsrechte klagen. Ob eine Verletzung der Persönlichkeitsrechte vorliegt, muss im Einzelfall entschieden werden, ist aber vor

---

[38] „Softwaredistributoren" sind Unternehmen, die sich auf die Zusammenstellung und den Verkauf von Open Source Software spezialisiert haben (siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen")

[39] „Embedded (dt. „Eingebettete") Systeme" bezeichnen Rechnersysteme, die zur Steuerung in andere Komponenten wie z.B. Flugzeuge oder Autos integriert, und somit nicht sichtbar sind.

dem Hintergrund, dass Änderungen der Software auffällig gekennzeichnet werden müssen, zumindest fraglich.

d) Nicht bekannte Nutzungsarten

Die GPL beschränkt die Einräumung der Nutzungsrechte nur auf die zum Zeitpunkt der Freigabe bekannten Nutzungsarten. Ergeben sich durch technischen Fortschritt oder andere Umstände neue Nutzungsarten der Software, so besitzt der Nutzer zunächst keine Rechte zur Verwendung der Software in dieser Form. Betrachtet werden muss dabei immer der Zeitpunkt der ersten Lizenzierung der Software unter der GPL.

Pflichten der Nutzer

a) Verpflichtung zur Freigabe von Bearbeitungen

Grundsätzlich verpflichtet die GPL die Nutzer der Software dazu, Veränderungen derselben wieder unter den Bedingungen der GPL zu veröffentlichen. Dies muss weiterhin geschehen, ohne dass dafür Lizenzgebühren verlangt werden dürfen. Das Ziel dieser Bestimmung ist die Wahrung der freiheitlichen Nutzungsrechte an der Software bzw. an aus ihr hervorgegangenen Modifikationen. Dies wirft in der Praxis jedoch Probleme bezüglich der Kombination von GPL-Software und proprietärer Software auf. Die Frage, ob eine Kombination aus offenen und proprietär vertriebenen Komponenten eine Bearbeitung der freien Software darstellt oder ob es sich um zwei unabhängige Werke handelt, entscheidet im Einzelfall, in wieweit das entstehende Gesamtwerk proprietär vermarktet werden darf. Im Folgenden werden dazu die Bedingungen diskutiert, unter denen eine Kombination von offener und nicht-offener Software möglich ist.

Zunächst einmal ist eine Freigabe nur dann erforderlich, wenn das resultierende Softwarepaket auch veröffentlicht wird. Modifikationen für den privaten Einsatz sind von der Verpflichtung ausgenommen. Allerdings liegt nach dem deutschen Urheberrecht bereits eine Veröffentlichung vor, sofern der Kreis der Nutzer nicht durch „gegenseitige Beziehungen oder durch Beziehungen zum Veranstalter persönlich untereinander verbunden sind“ (§15 Abs. 3 UrhG.). Dementsprechend kann bereits die Veröffentlichung innerhalb eines Unternehmens dazu führen, dass die Verpflichtungen der GPL in Kraft treten.

Des Weiteren ist eine getrennte Verbreitung von GPL-Software und Unternehmens- oder Privatsoftware problemlos möglich. Auch in diesem Fall greifen die Bestimmungen der GPL nicht, sodass die eigenen Entwicklungen proprietär vertrieben werden können.

Erst mit der gemeinsamen Verbreitung von GPL-Software und anderen Programmteilen stellt sich die Frage, ob es sich bei der Kombination um ein

abgeleitetes Gesamtwerk, eine so genannte *„derived work“* handelt, oder ob zwei unabhängige Programmteile vorliegen. Hierbei müssen sowohl technische wie auch funktionale Kriterien herangezogen werden. Technische Kriterien, die für die Annahme einer „derived work“ sprechen, sind das gemeinsame Laden von vorbestehendem und hinzugefügtem Code, die Tatsache, dass sie Teil eines Executables sind sowie die Ausführung im gleichen Adressraum. Kriterien, die gegen diese Annahme sprechen, sind Kommunikationsmechanismen zwischen den einzelnen Modulen wie *Pipes* [40], *Queues* [41], *Sockets* [42] oder *Kommandozeilenargumente*[43], die typischerweise zur Kommunikation zwischen verschiedenen Programmen herangezogen werden. In einigen Fällen reichen diese technischen Kriterien jedoch nicht aus, um zu bestimmen, ob es sich um ein abgeleitetes Werk handelt oder ob zwei separate Programme vorliegen. In diesen Fällen müssen funktionale Kriterien herangezogen werden. Hier gilt, erfüllen GPL-Software und die Benutzermodifikation nach der Anschauung eines Durchschnittsfachmanns dieselbe Funktion, so wird von einer abgeleiteten Arbeit ausgegangen. Ist dies nicht der Fall, so handelt es sich um zwei unabhängige Programme. Als Leitlinie für diese Entscheidung gilt, dass komplexe Programme wie Betriebssysteme oder Textverarbeitungsprogramme immer als eine Zusammenstellung mehrerer unabhängiger Einzelprogramme angesehen werden.

Sonderfälle sind das Einbinden von *GPL-Bibliotheken* und die Verwendung von *GPL-Compilern*. Im Fall der Bibliotheken gilt, dass die Anwendung der technischen Kriterien zumindest bei einer dynamischen Verlinkung widersprüchliche Ergebnisse liefert, sodass in diesem Fall zumeist funktionale Kriterien herangezogen werden müssen. Eine statische Verlinkung zwischen Programm und GPL-Bibliothek spricht dagegen für die Annahme eines abgeleiteten Gesamtwerks. Die Nutzung von Compilern, die unter der GPL lizenziert sind, ist in der Regel unproblematisch. Der entstehende Objektcode wird im Normalfall keine urheberrechtlich geschützten Teile des Compilerquellcodes enthalten. Allerdings existieren auch Compiler, die im Zuge des Kompiliervorgangs auch Teile des Programmquellcodes in den Objektcode

---

[40] Der Begriff „Pipe“ bezeichnet einen gepufferten Datenstrom zwischen zwei Prozessen

[41] Eine „Queue“ (oder dt. „Warteschlange“) ist eine Datenstruktur zur Pufferung der Datenübergabe zwischen Prozessen. Sie arbeitet nach dem „First-In-First-Out-Prinzip“.

[42] Ein „Socket“ ist eine Softwareschnittstelle, die zur bidirektionalen Kommunikation in Netzwerken oder zwischen Prozessen eingesetzt wird

[43] „Kommandozeilenargumente“ sind Optionen, die Programmen beim Start manuell mitgegeben werden können

übernehmen. In diesem Fall muss dann auch der resultierende Objektcode der GPL unterstellt werden und darf nicht proprietär vertrieben werden.

b) Verpflichtung zur Mitlieferung der GPL

Die Verpflichtung zur Mitlieferung der GPL besagt, dass eine Verbreitung der GPL-Software nur gestattet ist, wenn dem Programm eine Kopie der Lizenz beigelegt wird.

c) Das Lizenzgebührenverbot

Für die Verbreitung von GPL-Software darf keine Lizenzgebühr erhoben werden. Dies schließt allerdings nicht aus, dass Gebühren für gewährleistete Garantien, die Erstellung von CD-ROMs oder anderem Support verlangt werden dürfen. Die GPL verbietet lediglich die Erhebung von Lizenzgebühren.

d) Verbot der Veränderung von Urhebervermerken

Urhebervermerke, die eine GPL-Software enthält, dürfen nicht verändert werden und müssen bei einer Verbreitung der Software mitverbreitet werden. Änderungen müssen deutlich sichtbar gekennzeichnet werden. Dies kann jedoch auch anonym geschehen. Es besteht keine Verpflichtung, Urhebervermerke zu eigenen Änderungen zu ergänzen.

e) Verpflichtungen bei einem Vertrieb im Objektcode

Da das Hauptziel der GPL die Offenlegung des Quellcodes einer Software betrifft, ist die Verbreitung von GPL-Software im Objektcode an die Einhaltung einer von drei Bedingungen gebunden.

Die erste Möglichkeit, den Verpflichtungen der GPL nachzukommen, besteht dabei in der Mitlieferung des Quelltextes. Dies muss auf einem für den Datenaustausch üblichen Medium erfolgen.

Die zweite Möglichkeit ist die Lieferung eines schriftlichen Angebots, den Quellcode der Software bei Bedarf auf einem Datenträger nachzuliefern. Dieses Angebot muss mindestens drei Jahre Gültigkeit besitzen.

Die letzte Möglichkeit bezieht sich auf Nutzer der Software, die, wie im zweiten Punkt erläutert, ein schriftliches Angebot zur Lieferung des Quellcodes erhalten haben. Diese dürfen bei einer Weiterverbreitung der Software anstelle des Quellcodes auch das erhaltene schriftliche Angebot weiterleiten und müssen so nur bei Bedarf den dazugehörigen Code ausliefern.

Die GNU Lesser General Public License (LGPL)

In ihren Grundzügen gleicht die LGPL der GPL. Die Unterschiede zwischen den beiden Lizenzen resultieren aus den Problemen, die sich hinsichtlich der Nutzung von Open

Source Softwarebibliotheken in kommerziellen, proprietär vertriebenen Anwendungen ergeben. Da die GPL hinsichtlich der Einhaltung des Copylefts sehr strikte Forderungen beinhaltet, führt dies dazu, dass GPL-Bibliotheken nur schwer in proprietäre Anwendungen integrierbar sind, ohne dabei den eigenen Quellcode ebenfalls offen legen zu müssen. Da es aber für die Verbreitung des offenen Betriebssystems GNU/Linux von entscheidender Bedeutung war und ist, dass auch proprietäre Anwendungen auf die Systembibliotheken zugreifen können, wurde die LGPL entwickelt. Sie ist im Hinblick auf die Einhaltung des Copylefts weniger strikt, sodass eine Einbindung offener Bibliotheken in proprietäre Anwendungen leichter möglich wird.

Rechte der Nutzer

Die Rechte, die der Nutzer mit dem Erwerb einer unter der LGPL lizenzierten Software erhält, gleichen grundsätzlich denen, die ihm die GPL gewährt. Dies beinhaltet im Wesentlichen die Gewährung eines einfachen Nutzungsrechtes für alle zum Zeitpunkt der Lizenzierung bekannten Nutzungsarten sowie das Recht, die Software zu modifizieren und veränderte Versionen wieder zu verbreiten. Zusätzlich ist es den Benutzern allerdings gestattet, LGPL-Bibliotheken gemeinsam mit proprietärer Software zu verbreiten, ohne dass das resultierende Gesamtwerk automatisch unter der LGPL lizenziert werden muss. Dieses Recht ist allerdings an die Einhaltung bestimmter Bedingungen (siehe „Pflichten der Nutzer“) geknüpft.

Darüber hinaus können LGPL-Bibliotheken und proprietäre Bibliotheken beliebig miteinander kombiniert werden, sofern sich ein auffälliger Verweis auf die freien Bibliotheken in der Gesamtbibliothek befindet und die freien Bibliotheken zusätzlich noch einmal unter den Bedingungen der LGPL mitgeliefert werden.

Schließlich erhält jeder Nutzer noch das Recht, die Software auch unter den strengeren Bedingungen der GPL zu verbreiten.

Pflichten der Nutzer

a) Verpflichtungen bei Verbreitung und Veränderung der Bibliothek

   Diese entsprechen den Verpflichtungen, die sich aus den Bedingungen der GPL ergeben. Veränderte Bibliotheken müssen bei einer Veröffentlichung wieder unter den Bedingungen der LGPL verbreitet und die Veränderungen dabei deutlich sichtbar gekennzeichnet werden. Urhebervermerke dürfen nicht geändert sowie Lizenzgebühren nicht erhoben werden. Schließlich muss der Software stets eine Kopie der Lizenz beigelegt werden.

b) Verpflichtungen bei der Kombination mit „proprietären“ Programmen

Den gemeinsamen Vertrieb einer Kombination von freien Softwarebibliotheken und proprietären Programmen, die auf diese Bibliotheken zugreifen, knüpft die LGPL an spezielle Bedingungen. So muss es dem Endnutzer der resultierenden Gesamtanwendung gestattet sein, Veränderungen für die eigenen Bedürfnisse vorzunehmen. Dies impliziert unter anderem auch die Erlaubnis, die proprietären Teile des Systems zu dekompilieren, um den Quellcode modifizieren zu können. Darüber hinaus muss entweder sogar der gesamte Quellcode des Programms offen gelegt werden, oder der *Linking-Mechanismus*[44] zwischen der Open Source Bibliothek und dem proprietären Programm muss *„suitable“*, d.h. derart gestaltet sein, dass eine Änderung der Implementierung der freien Bibliotheken keine Auswirkungen auf die Funktionsweise der gesamten Anwendung hat, sofern die Schnittstellen des Systems eingehalten werden.

Über die genannten Freiheiten zur Anpassung des Quellcodes hinaus erwirbt der Endnutzer allerdings nicht automatisch die weitergehenden Rechte zum Kopieren und Verbreiten der proprietären Software, die typischerweise mit der Nutzung von GPL- oder LGPL-Software einhergehen. Dies bedeutet, dass der kommerzielle Vertrieb einer Kombination aus freier Softwarebibliothek und proprietären Programmteilen in Verbindung mit der Erhebung einer Lizenzgebühr möglich ist. Allerdings muss der Hersteller der Software große Zugeständnisse im Hinblick auf die Offenlegung des eigenen Quellcodes machen.

Ein isolierter Vertrieb von proprietärer Software und LGPL-Bibliotheken ist dagegen analog zu den Bestimmungen der GPL problemlos möglich.

4.3.5.2 „BSD“-artige Lizenzen

Die Abkürzung „BSD“ stammt von einer Unix-Variante, die in den 70er Jahren an der Universität von Berkeley entwickelt wurde und bedeutet „Berkeley Software Distribution“. „BSD“-artige Lizenzen zeichnen sich vor allem dadurch aus, dass sie keine mit der GPL vergleichbaren Bestimmungen zur Freigabe von abgeleiteten Werken enthalten. Modifikationen der Software oder Kombinationen mit anderen Softwarekomponenten können demnach auch proprietär vertrieben werden. Allerdings enthalten einige der Lizenzen dieser Kategorie die Verpflichtung, bei einer Verbreitung unveränderter Versionen der Software, entweder allein stehend oder in Kombination mit anderen Anwendungsteilen, einen Werbehinweis auf den Urheber der Software mitzuverbreiten. Wird die Software demgegenüber modifiziert, so ist die Nennung der Urheber untersagt, insbesondere innerhalb von Werbematerialien. Darüber hinaus

[44] Verbindungsmechanismus zwischen Softwarekomponenten wie beispielsweise ein Funktionsaufruf

müssen die Lizenzbestimmungen sowie der Haftungs- und Gewährleistungsausschluss mitverteilt werden.[45]

4.3.5.3 „Mozilla“-artige Lizenzen

„Mozilla“-artige Lizenzen sind grundsätzlich eng verwandt mit den „Copyleft“-Lizenzen, da sie ebenfalls die Verpflichtung beinhalten, Modifikationen einer Anwendung wieder derselben Lizenz zu unterstellen. Allerdings sind sie dabei vor allem aufgrund der Entstehungsgeschichte der *Mozilla Public License*, des Hauptvertreters dieser Kategorie, weniger streng im Hinblick auf die gemeinsame Verbreitung mit proprietären Programmteilen. Da die Firma Netscape zum Zeitpunkt der Veröffentlichung des Quellcodes ihres Browsers *„Navigator“*, um den sich später das Mozilla-Projekt bildete, nicht die Urheberrechte am gesamten Programmcode besaß, musste eine Lizenz zur Verbreitung der Software eine Kombination mit proprietären Anwendungsteilen gestatten.

Gleichzeitig wollte man sich allerdings die Möglichkeit offen halten, externe Verbesserungen der Software selbst nutzen zu können. Das Ergebnis war eine Kompromisslösung zwischen den echten „Copyleft“-Lizenzen und den „BSD“-artigen Lizenzen. „Mozilla“-artige Lizenzen erlauben dabei den Vertrieb einer Kombination aus proprietären und offenen Softwarekomponenten als ein Gesamtwerk. Modifikationen einer Open Source Software dagegen müssen wieder zu den Bedingungen der jeweiligen Open Source Lizenz veröffentlicht werden. Eine Modifikation liegt dann vor, wenn bestehende Dateien verändert werden. Werden stattdessen neue Dateien hinzugefügt, so zählt dies nicht als Modifikation im Sinne der Lizenz, sodass die Verpflichtung zur Offenlegung der Erweiterungen nicht greift.

Zusätzlich enthalten „Mozilla“-artige Lizenzen ebenfalls die Verpflichtung zur Mitlieferung des Lizenztextes sowie Urhebervermerken.[46]

4.3.5.4 „Artistic“-Lizenzen

Die „Artistic“-Lizenzen weichen von den zuvor genannten Lizenzkategorien insofern ab, als dass sie abhängig von der Art der Veränderungen der Originalsoftware unterschiedliche Rechtsfolgen beinhalten. So bringen weniger weit reichende Modifikationen einer Software, die unter einer „Artistic“-Lizenz lizenziert ist, keine zusätzlichen Verpflichtungen mit sich. Die weitergehenden Bestimmungen greifen erst bei größeren Veränderungen einer Anwendung. Demnach muss das resultierende Softwarepaket entweder ebenfalls offen gelegt werden, die Verbreitung findet lediglich

---

[45] siehe Anhang A.2

[46] siehe Anhang A.3

innerhalb eines Unternehmens bzw. einer Organisation statt, oder die veränderte Version erhält einen neuen Namen und das ursprüngliche Programm wird gemeinsam mit einer genauen Beschreibung der Änderungen mitgeliefert. Wann genau eine wesentliche oder unwesentliche Modifikation eines Softwarepaketes vorliegt, ist unklar und muss im Einzelfall entschieden werden. Eine kommerzielle Nutzung von Open Source Software unter einer „Artistic"-Lizenz ist vor dem Hintergrund der potentiellen Umbenennung einer Anwendung allerdings zumindest möglich.[47]

4.3.6 Geschichte

Anfänge

Die Geschichte der Freien oder Open Source Projekte hat ihren Ursprung bereits in den 60er Jahren. Zu diesem Zeitpunkt war es an amerikanischen Universitäten und Forschungseinrichtungen üblich, entwickelte Software untereinander auszutauschen. Dieser Austausch beinhaltete auch die Weitergabe des Quellcodes der Anwendungen. Es entwickelte sich eine Philosophie der gegenseitigen Unterstützung, die so genannte *Hacker-Ethik* [48].

Im Jahr 1969 entschied sich dann die Firma IBM, ihre Software künftig unter urheberrechtlichen Schutz zu stellen und nur in Verbindung mit entsprechenden Lizenzverträgen zu verbreiten. Die bislang freie Dienstleistung zur Wartung und Weiterentwicklung von Software wurde kostenpflichtig. Andere Unternehmen folgten dem Beispiel von IBM, so dass sich Software mehr und mehr zu einer kommerziell genutzten Ware entwickelte, die zur Wahrung der wirtschaftlichen Interessen des jeweiligen Unternehmens geschützt und geheim gehalten werden musste. Verstärkt wurde dieser Effekt noch durch das Aufkommen bezahlbarer Mikrocomputer, die das Produkt „Software" für eine größere Zielgruppe interessant machten.

Im Zuge dieser Entwicklungen wurden zunehmend auch die Softwareexperten der Universitäten und Forschungseinrichtungen, die so genannten *Hacker*, von Softwarefirmen abgeworben. Dies führte dazu, dass dort bestehende Hacker-Gemeinschaften allmählich immer kleiner wurden. Einer dieser Experten war Richard Stallman. Er arbeitete am *AI Lab*[49] des Massachusetts Institute of Technology und war den freien Umgang mit Software und dem dazugehörigen Quellcode gewohnt. Als die

---

[47] siehe Anhang A.4

[48] Mit dem Begriff „Hacker-Ethik" wird die Philosophie früherer Softwareexperten der Universitäten und Forschungseinrichtungen, den so genannten Hackern, bezeichnet. Demnach ist die Weitergabe und Verteilung von Programmen und dem damit verbundenen Wissen ein ehrenhaftes Ziel.

[49] Abteilung für Künstliche Intelligenz

Firma Symbolics nach und nach große Teile der Hacker-Gemeinschaft seiner Abteilung abwarb, stand auch er vor der Entscheidung, der proprietären Softwarewelt beizutreten. Da er aber den Grundsatz der Geheimhaltung des Quellcodes einer Software ablehnte und stattdessen von einem freien und kooperativen Austausch überzeugt war, fasste er den Entschluss, dies nicht zu tun. Stattdessen gründete er 1983 das GNU-Projekt, nachdem zu Beginn des Jahres die Firma AT&T eine proprietäre Version ihres Betriebssystems *UNIX*[50] auf den Markt gebracht hatte, sodass keine frei verfügbare Version des Betriebssystems mehr existierte.[51]

Das GNU-Projekt

Die Ziele des 1983 von Richard Stallman ausgerufenen GNU-Projektes waren zum einen die Bildung einer neuen Hacker-Gemeinschaft, wie er sie von seiner Arbeit aus dem AI Labor des Massachusetts Institute of Technology kannte. Zum anderen plante er die Entwicklung eines freien Betriebssystems als eine Alternative zu den bestehenden, proprietär vertriebenen Systemen. GNU-Software sollte „frei“ sein. Die Bezeichnung „frei“ bedeutete für Stallman allerdings nicht die Gleichsetzung mit dem Begriff „kostenlos“. Vielmehr meinte er die Freiheiten, die mit der Nutzung einer Software einhergehen sollten. Freie Software sollte vier Maximen gehorchen:

1) „You have the freedom to run the program, for any purpose.”

2) „You have the freedom to modify the program to suit your needs. (To make this freedom effective in practice, you must have access to the source code, since making changes in a program without having the source code is exceedingly difficult.)”

3) „You have the freedom to redistribute copies, either gratis or for a fee.”

4) „You have the freedom to distribute modified versions of the program, so that the community can benefit from your improvements.”

- Richard Stallman[52]

Um seine Vorstellungen von einem freien Betriebssystem realisieren zu können, kündigte Stallman zunächst seine Stelle am Massachusetts Institute of Technology, da er sonst nach amerikanischem Recht fürchten musste, die Rechte an der von ihm entwickelten Software zu verlieren. Um seine Entwicklungen dennoch finanzieren zu können, gründete er zusammen mit der allmählich anwachsenden

---

[50] UNIX ist ein Anfang der 70er Jahre von den „Bell Laboratories“ entwickeltes Betriebssystem

[51] Zur Geschichte „Freier Software“ und des GNU-Projektes, siehe http://de.wkipedia.org/Freie_Software und http://www.gnu.org/gnu/thegnuproject.html

[52] siehe http://www.gnu.org/gnu/thegnuproject.html

Entwicklergemeinschaft 1985 die Free Software Foundation (FSF), eine steuerbefreite gemeinnützige Gesellschaft zur Förderung der Freien Softwarebewegung, die noch heute besteht. Die FSF nahm Spenden entgegen und vertrieb gleichzeitig den Verkauf des von Stallman bereits entwickelten Texteditors GNU Emacs und finanzierte so die Weiterentwicklung des GNU-Projektes. 1988 unternahm Stallman dann einen weiteren Schritt zur Etablierung der Freien Software. Nachdem er Teile des GNU *Emacs Texteditors* ändern musste, da ein ehemals freier Programmquellcode, den er für seine Entwicklungen benutzt hatte, nachträglich unter ein Copyright gestellt wurde, überlegte er, wie er die von ihm gewährten Freiheitsrechte an der GNU-Software dauerhaft absichern konnte. Das Ergebnis war die Schaffung der GNU General Public License (GPL)[53], der heute am weitesten verbreiteten Open Source Lizenz. Das besondere an der Lizenz war das Prinzip des Copyleft, das ein Gegenstück zum bestehenden Copyright bildete. Während das Copyright die Rechte an einem Gegenstand privatisierte, sollte das Copyleft die Freiheiten, die Stallman den Nutzern der GNU-Software an ihr gewährte, für die Allgemeinheit absichern. Dazu musste sich jeder Nutzer mit der Annahme der Lizenzbestimmungen dazu verpflichten, die Software oder Modifikationen davon nur wieder unter den Bestimmungen der GPL zu veröffentlichen. So wollte Stallman sicherstellen, dass Modifikationen der GNU-Software nicht unter einer proprietären Lizenz weiterverbreitet werden konnten.

Bis zum Ende der 80er Jahre hatten Stallman und die GNU-Community ihr Ziel, ein freies Betriebssystem zu entwickeln, fast verwirklicht. Allein der Betriebssystemkern fehlte noch.

GNU/Linux

1991 kam der letzte noch fehlende Baustein hinzu. Der finnische Student Linus Torvalds hatte im April des Jahres damit begonnen, ein eigenes Betriebssystem zu entwickeln, da ihm die Fähigkeiten des damals an seiner Universität verbreiteten Systems *Minix*[54] nicht ausreichten. Dazu bediente er sich Teilen des von Stallman entwickelten GNU-Systems und fügte seinen Betriebssystemkern hinzu. Nachdem er im August des Jahres in einer Minix-Newsgroup sein Vorhaben, ein Betriebssystem zu entwickeln, angekündigt und um Anregungen gebeten hatte, erschien im September 1991 die Version 0.01 und bereits nur 3 Monate später die Version 0.10. Der Name des neuen Betriebssystems war Linux entsprechend dem ersten öffentlichen Verzeichnis, in dem der Quellcode der Software verfügbar war.

---

[53] siehe Kapitel 4.3.5 Lizenzen

[54] Ein von Andrew Tanenbaum entwickeltes Betriebssystem, das als Lehrsystem an Universitäten eingesetzt wurde

Torvalds entschied sich, die Software unter der von Stallman entwickelten GNU General Public License zu lizenzieren, um sie für andere Studenten und interessierte Entwickler zugänglich zu machen. Die Zahl der Interessenten und Entwickler stieg begünstigt durch den Aufschwung des neuen Kommunikationsmediums *„Internet“* sehr schnell an, sodass sich das „GNU/Linux“-Betriebssystem bald zu einer echten Alternative zu proprietären Betriebssystemen entwickelte. Dies führte dazu, dass auch kommerzielle Unternehmen auf den Linux-Zug aufsprangen und fertige Distributionen und andere Dienstleistungen wie Support oder Qualitätsgarantien gegen Gebühren anboten.[55] Der Weg für eine Nutzung von GNU/Linux in Unternehmen war damit geebnet.

Netscape Navigator/Mozilla Firefox

Der nächste historische Schritt in der Geschichte der Freien Software war die Freigabe des Quellcodes des damals führenden Browsers „Navigator“ der Firma Netscape. Nachdem 1997 Eric S. Raymond seinen Aufsatz mit dem Titel *„The cathedral and the bazaar“* veröffentlicht hatte, in dem er die Vorteile eines offenen Softwareentwicklungsprozesses beschrieb, und der Druck der Firma Microsoft mit ihrem aggressiv vermarkteten Browser *„Internet Explorer“* immer größer wurde, kündigte Netscape im Januar 1998 an, den Quellcode des Navigators offen zu legen. Ziel des Unternehmens war dabei zum einen der Rückgriff auf die Programmierarbeiten externer Entwickler, insbesondere da Teile des Browsers urheberrechtlich nicht Netscape gehörten, sodass die offene Version allein nicht lauffähig war. Zum anderen erhoffte man sich durch diesen Schritt einen Werbeeffekt für den Kernbereich des Unternehmens.

In der Folge der Ankündigung von Netscape trafen sich die führenden Köpfe der Freien Softwarebewegung, um im Zuge der kurzfristig gestiegenen öffentlichen Wahrnehmung über eine Möglichkeit der besseren Vermarktung des Konzeptes „Freie Software“ zu beraten. Das Ergebnis war die Schaffung des neuen Begriffs „Open Source Software“ zu Marketingzwecken. Dieser sollte im Gegensatz zur bis dahin üblichen Bezeichnung „Freie Software“ weniger den ideologischen Charakter hervorheben, als vielmehr den technischen Aspekt des frei zugänglichen Quellcodes, um der kritischen Haltung vieler Unternehmen hinsichtlich des wirtschaftlichen Potentials des Phänomens entgegenzuwirken. Wenig später gründeten Eric S. Raymond und Bruce Perens die „Open Source Initiative“ (OSI), die vergleichbar zur Free Software Foundation den institutionellen Arm zur Unterstützung der Open Source Bewegung bilden sollte. Damit war der Grundstein für die zunehmende kommerzielle Nutzung von Open Source Software gelegt.

---

[55] Bekannte Distributoren sind z. B. die Firmen Red Hat, Suse, Mandrake und Debian

4.4 Allgemeine Kontextanalyse

Um die Prozesse in Open Source Projekten genauer verstehen zu können, gehört dem Rahmenwerk der DeGEval folgend neben der Klärung der Evaluationszwecke und der Skizzierung des Evaluationsgegenstandes sowie seines Umfeldes vor allem eine *allgemeine Kontextanalyse*[56] zur genauen Untersuchung der Vorgänge in Open Source Projekten zu den Aufgaben in der Phase der Definition des Evaluationsproblems. Da die Strukturen und Prozesse in den Projekten mittelbar auch durch die Verhältnisse im Umfeld beeinflusst werden, muss dies eine Analyse der Situation in den Unternehmen als mögliche Zielgruppe der Projektsoftware mit einschließen.

4.4.1 Soziale Rahmenbedingungen in Open Source Projekten

Einer der wesentlichen Unterschiede zwischen Open Source Projekten und den Entwicklerteams der Softwareunternehmen besteht in der Struktur ihrer sozialen Netzwerke. Diese zeichnen sich den Ausführungen von Crowsten und Howison [Crowsten2005] folgend in Softwareprojekten durch die Ausprägung dreier Aspekte aus – die beteiligten Individuen, deren Aktionen sowie deren Interaktionen untereinander.

Die beteiligten Individuen

Art der Beteiligten

Die beteiligten Personen in Open Source Projekten sind typischerweise freiwillige Mitarbeiter. Ihre Arbeit erfolgt in der Regel ohne eine direkte Bezahlung, auch wenn eine Studie von Hars und Ou aus dem Jahr 2001 [Hars2001] belegt, dass bereits 16% der befragten, in Open Source Projekten involvierten Personen einen finanziellen Ausgleich für ihre Arbeit erhalten. Hinzu kommt, dass die führenden Entwickler in bekannten Open Source Projekten im Gegensatz zum weit verbreiteten Mythos[57] professionelle Softwareentwickler sind, die auch hauptberuflich mit der Entwicklung von Software beschäftigt sind. Dies gilt zwar für kleinere Projekte nicht in gleichem Maße [Gorling2003], aber Projekte, die innerhalb der Open Source Bewegung eine Rolle spielen, werden in der Regel von professionellen Softwareentwicklern geführt [Healy2003][Gorling2003].

---

[56] Die Bezeichnung „allgemein“ wird verwendet, um den Unterschied zur Definition des Evaluationskontextes in Kapitel 5 zu verdeutlichen

[57] Ein Mythos im Zusammenhang mit Open Source Software besteht darin, dass die Open Source Bewegung lediglich aus Freizeitentwicklern besteht, die gemeinsam das Microsoft-Imperium gefährden

Anzahl der Beteiligten

Der offene Entwicklungsprozess in den Projekten ermöglicht es einer breiten Entwicklergemeinschaft, Beiträge zur Verbesserung der jeweiligen Software zu leisten. Hierin besteht eine wesentliche Stärke von Open Source Projekten. Während die Entwicklerteams der Softwareunternehmen in der Regel eine limitierte Anzahl von Entwicklern umfassen, können Open Source Projekte teilweise auf mehrere hundert involvierte Personen verweisen. So besitzt beispielsweise das *Debian-Projekt* inzwischen über 1200 Maintainer [Michlmayr2005c], d.h. Pfleger bestimmter Dateien und Module. Allerdings gilt dies nur für einen relativ kleinen Anteil von Open Source Projekten. Über 70% der bei Sourceforge registrierten Projekte bestehen lediglich aus zwei oder sogar nur einem Entwickler [Gorling2003].

Organisation der Beteiligten

Hat sich allerdings erst einmal eine größere Gemeinschaft von aktiven Teilnehmern in einem Open Source Projekt gebildet, so stellt sich die Frage nach der Organisation ihrer Mitglieder. Während Raymond [Raymond2001] dazu in Open Source Projekten lediglich flache Hierarchien unter den Projektmitgliedern annimmt, postulieren Healy und Schussmann in erfolgreichen Projekten sogar stark ausgeprägte hierarchische Strukturen [Healy2003]. Letztere Annahme wird von den Untersuchungen von Jensen und Scacchi [Jensen2007] zu den Strukturen und Prozessen im Apache-, im Mozilla- und im Netbeans-Projekt bestätigt. Größere Projekte scheinen vergleichbar zu Organisationen bestimmte hierarchische Strukturen auszubilden. Diese sind dabei allerdings zumindest in der Startphase weniger stark formalisiert. Sie äußern sich in einer Art *Hackordnung* [Healy2003], die aufgrund von Faktoren wie wahrgenommener Kompetenz bzw. Grad oder Dauer der Involvierung der beteiligten Personen in das Projekt entsteht. Raymond erwähnt in diesem Zusammenhang, dass nur etwa 5% der talentiertesten Entwickler überhaupt von der Community akzeptiert werden. Dies reduziert den Aufwand zur Organisation der Entwicklergemeinschaft erheblich und bildet eines der Abgrenzungsmerkmale zu kommerziellen Softwareprojekten, in denen die Projektleiter zu einem großen Teil damit beschäftigt sind, die Entwicklungsarbeit der anderen 95% zu organisieren [Raymond2001].

Trotzdem bildet gerade die Führung von Open Source Projekten eine besondere Herausforderung für die verantwortlichen Personen. Die Aufgabe der Projektadministratoren besteht dabei zum einen in der Delegation der anstehenden Arbeiten, zum anderen aber auch in der Kontrolle der eingehenden Beiträge. Dies erfordert sowohl ein breites technisches Wissen zur Software als auch soziale Kompetenz, da die Delegation von anstehenden Aufgaben die Überzeugung der übrigen Entwickler zu einem Engagement bedingt [Rothfuss2002].

Darüber hinaus sind zwei weitere Aspekte in Bezug auf die Organisation der beteiligten Personen in Open Source Projekten voneinander zu unterscheiden.

Erstens ist der Status einer Person unter Umständen gekoppelt an einen bestimmten Bereich. Vor allem große Projekte bilden oftmals Subprojekte auf Modulebene aus, um den Koordinationsaufwand zur Auflösung bestehender Abhängigkeiten in den Programmierarbeiten unterschiedlicher Entwickler möglichst gering zu halten [Mockus2002]. Dies führt dazu, dass sich auch die wahrgenommene Kompetenz der jeweiligen Entwickler und damit verbunden ihr sozialer Status lediglich auf das entsprechende Modul beziehen. So kann ein Entwickler im Mozilla-Projekt beispielsweise durch kontinuierliche Arbeit an einem bestimmten Modul zum so genannten *„Module Owner“* werden, d.h. zum Verantwortlichen für das entsprechende Modul [Jensen2007], sodass seine Stimme bei speziellen Fragen zu diesem Modul ein entsprechendes Gewicht besitzt.

Zweitens muss hinsichtlich des Hierarchisierungsgrades grundsätzlich zwischen Arbeiten an den Programmerweiterungen, den so genannten Plug-Ins, und der Arbeit am Kern einer Software unterschieden werden. Während die Projektadministratoren die Kräfte im Bereich der Programmerweiterungen meist freier walten lassen, ist die Arbeit am Kern der Anwendung meist geprägt durch eine starke Kontrolle und Zuteilung der Kompetenzen [Healy2003].

Einige Projekte formulieren zusätzlich sogar explizit die Prozesse zu einem projektinternen Aufstieg und etablieren so vergleichbar zu Organisationen bestimmte Laufbahnen. Jensen und Scacchi [Jensen2007] identifizieren dazu in ihrer vergleichenden Analyse der drei Open Source Projekte Mozilla, Apache und Netbeans sieben unterschiedliche Kategorien von Rollen für beteiligte Personen. Diese unterscheiden sich zum einen in der Art der Aufgaben, die mit der Ausübung einer Rolle der entsprechenden Kategorie verbunden sind, zum anderen in dem notwendigen Wissen über das Projekt und seine Software, das vorausgesetzt werden muss. Die sieben Kategorien sind *Qualitätssicherung, Quellcodeentwicklung, Quellcodeversionisierung, Benutzbarkeit, Marketing, Lizenzierung* und *Projektplanung*.

Die genaue Ausformung der möglichen Laufbahnen variiert dabei von Projekt zu Projekt. So besitzt das Apache-Projekt beispielsweise einen eindimensionalen Pfad der Rollenmigration, während das Mozilla-Projekt die unterschiedlichen Rollen inhaltlich stärker differenziert (siehe Abb.5 und 6).

Zudem identifizieren Jensen und Scacchi innerhalb der unterschiedlichen, projektspezifischen Laufbahnen insgesamt vier mögliche Arten des Rollenerwerbs, im Zuge derer Entwickler in Open Source Projekten weitergehende Rollen zugewiesen bekommen. Dies kann implizit durch die Ausführung einer speziellen Aufgabe, durch die Gewährung durch höher stehende Communitymitglieder, durch eine Wahl der

Community oder durch die Zuweisung einer Rolle durch autorisierte Gruppen erfolgen. Der zweite und der vierte Fall unterscheiden sich dabei darin, dass im zweiten Fall in der Regel ein Antrag des jeweiligen Mitglieds der Community vorausgeht (siehe Tab.2).

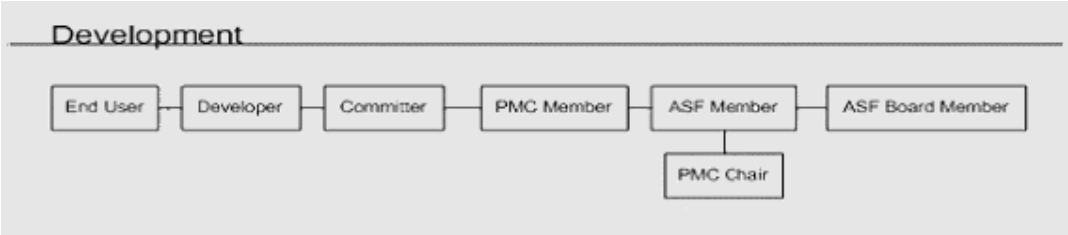

Abbildung 5: Hierarchie und Rollenmigration im Apache-Projekt nach Jensen und Scacchi

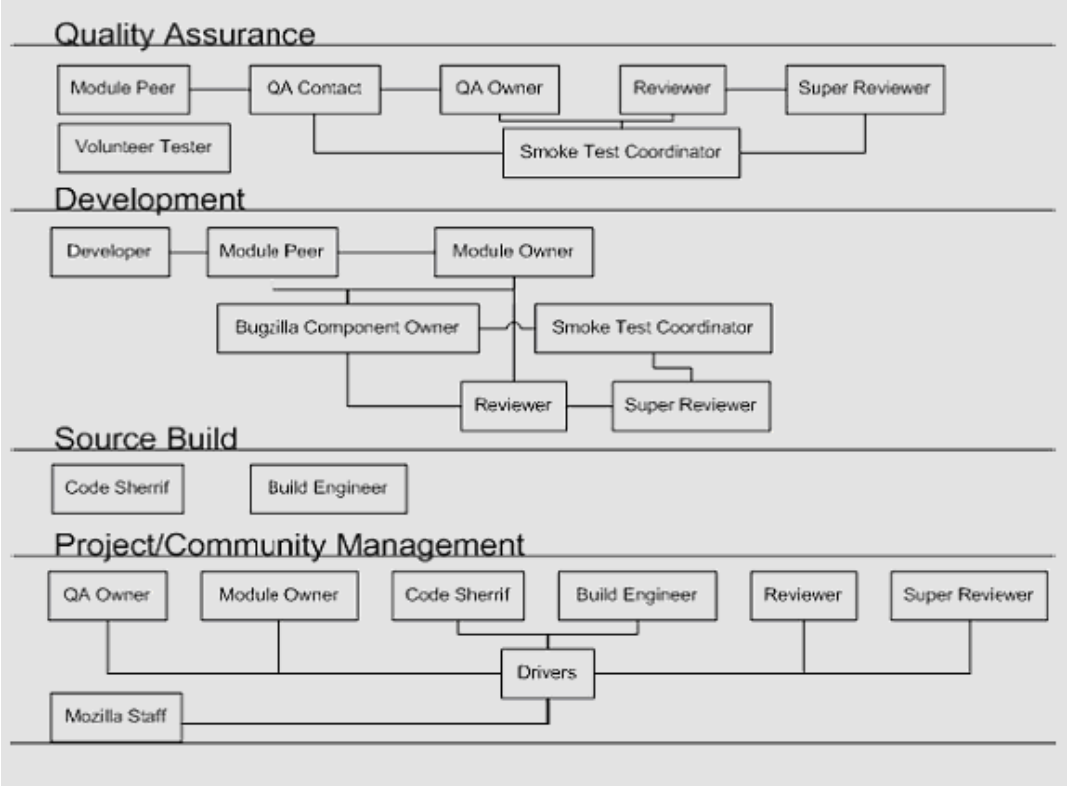

Abbildung 6: Hierarchie und Rollenmigration im Mozilla-Projekt nach Jensen und Scacchi

Tabelle 2: Typen der Rollenmigration in Open Source Projekten nach Jensen und Scacchi

Role Acquisition Method	*Description*
Implicit	Acquired by performing a task.
Earned/Granted	An individual or body of authority grants the rank to the community member. This may require that the community member apply for the position, or that he or she is nominated or sponsored by a higher ranking member, possibly involving a vote from the granting body.
Elected	An individual is voted into a position by the community at large or a subcommittee.
Appointed/Assigned	An individual or body of authority appoints the community member to a position.

Motive der Beteiligten

Da eine Beteiligung in Open Source Projekten wie erwähnt in der Regel ohne Bezahlung erfolgt, stellt sich die Frage, welches sind die Motive, die die Entwickler dazu bewegen, trotzdem große Teile ihrer Freizeit in die Weiterentwicklung oder Pflege von Open Source Software zu investieren?

Dazu unterscheiden Hars und Ou [Hars2001], Rossi und Bonaccorsi [Bonaccorsi2005] sowie Bitzer, Schrettl und Schröder [Bitzer2004] zwischen *intrinsischen* und *extrinsischen* Motiven. Während intrinsische Motive Umstände für menschliches Verhalten beschreiben, die mit Phänomenen wie „Spaß“, „Interesse“ oder „Herausforderung“ zusammenhängen, stehen extrinsische Motive mit der Aussicht auf eine externe Belohnung in Verbindung. Bitzer *et al.* nennen drei intrinsische Motive für die Teilnahme von Entwicklern an Open Source Projekten:

- Bedarf an einer bestimmten Softwarelösung
- „homo ludens payoff“
- Soziale Position, „gift culture“

Am Anfang des Engagements in einem Open Source Projekt steht demnach meist der Bedarf oder das Interesse an einer bestimmten Softwarelösung. Dies ist Bitzer *et al.* folgend zwar eine notwendige, aber noch keine hinreichende Voraussetzung für das Engagement in einem Open Source Projekt. Hinzu kommen zum einen Faktoren wie „Spaß am Programmieren“, „das Meistern einer Herausforderung“ oder „Intellektuelle Stimulation“, die sie unter dem Begriff *„homo ludens payoff“* zusammenfassen. Zum anderen ist das Arbeiten und Etablieren innerhalb einer Gemeinschaft ein wichtiges Motiv für Open Source Entwickler. Dies beinhaltet dabei sowohl die Freude am Arbeiten in einer „familiären“ Gemeinschaft an sich, wie auch das Erarbeiten eines sozialen Status innerhalb dieser Gruppe. Bitzer *et al.* bezeichnen dies mit dem Begriff *„gift culture“*. Jeder Entwickler überlässt der Gemeinschaft seine Entwicklungsarbeit und erhält als Ausgleich einen entsprechenden sozialen Status.

Vergleichbar dazu erwähnen Rossi und Bonaccorsi [Bonaccorsi2005] das Motiv des *„Reputation among peers“*. Entwickler, die sich aktiv an Open Source Projekten beteiligen, erlangen abhängig vom Umfang und der Qualität ihrer Beteiligung einen Ruf innerhalb der jeweiligen Projektgemeinschaft oder sogar innerhalb der gesamten Open Source Bewegung. Dies kann zum einen der Selbstbestätigung dienen, zum anderen ist es auch denkbar, dass ein guter Ruf innerhalb der Open Source Bewegung durchaus Vorteile bei der späteren Bewerbung für eine Arbeitsstelle haben kann. Dementsprechend ist der Ruf eines Entwicklers in Open Source Projekten sowohl intrinsisches wie auch extrinsisches Motiv. Andere extrinsische Motive sind für Rossi und Bonaccorsi finanzielle Entlohnung, Karrierevorteile, mit der Beteiligung verbundene Lernprozesse oder das Füllen einer Marktlücke [Bonaccorsi2005], das vergleichbar ist mit dem intrinsischen Motiv des Bedarfs an einer Software nach Bitzer *et al.*

Eine Umfrage von Lakhani, Wolf und Bates aus dem Jahr 2002, der so genannten *„Hacker survey“*, an Entwickler in Open Source Projekten fasst die genannten Motive zusammen. Demnach finden 43% der Befragten die Arbeit in Open Source Projekten intellektuell stimulierend bzw. versprechen sich eine Verbesserung ihrer Programmierfähigkeiten. 34% geben die Philosophie des freien Quellcodes als Motiv zur Partizipation in Open Source Projekten an, 20% der Befragten schätzen die Arbeit im Team und 17% sind professionell mit der Weiterentwicklung von Open Source Software beschäftigt. Andere Motive sind die Anpassung der Software für die eigene Benutzung (28,3%), der Ruf innerhalb der Open Source Bewegung (11%) oder der Wunsch, proprietäre Software zu übertreffen (11,1%) [BCG2002].

Aktionen der beteiligten Individuen

Arten der Aktionen

Der zweite Aspekt, der die sozialen Strukturen in Open Source Projekten prägt, sind die Aktionen der beteiligten Personen. Diese können, wie zuvor erwähnt, in verschiedene Bereiche unterteilt werden.

Der Einstieg in ein Open Source Projekt erfolgt meist über kleinere Aufgaben aus den Bereichen Qualitätssicherung und Benutzbarkeit. Beispiele dafür sind das Testen der Software, das Melden von Fehlern, das Einsenden von Fehlerkorrekturen, das Schreiben von Dokumentationen, das Übersetzen von Dokumenten in andere Sprachen und die aktive Teilnahme an Diskussionen auf der Entwicklermailingliste. Charakteristisch für diese Phase der Beteiligung ist, dass das jeweilige Mitglied der Community meist noch kein Schreibrecht für das Versionsverwaltungssystem besitzt. Fehlerkorrekturen oder Änderungsvorschläge werden entweder direkt an die verantwortlichen Entwickler geschickt oder offen auf der Mailingliste diskutiert. Überzeugt dieses Mitglied die

Verantwortlichen wiederholt von seiner Kompetenz, so erhält es schließlich den Schreibzugriff für das CVS oder SVN-System.

Ab diesem Punkt unterscheiden sich die weiteren Aufgaben für neu hinzugekommene Entwickler von Projekt zu Projekt. In der Regel werden sie zunächst nicht direkt mit Arbeiten beschäftigt sein, die den Kern der jeweiligen Anwendung betreffen. Typische Aufgaben sind dagegen das Verbessern bestehender Module außerhalb des Anwendungskerns sowie das Entwickeln neuer Plug-Ins. Entspricht die dabei gezeigte Qualität der Arbeiten erneut wiederholt den Vorstellungen der verantwortlichen Projektadministratoren, so erfolgt unter Umständen die Ernennung zu einem „Module Owner“. Dies verdeutlicht den Trend, dass Entwickler mit zunehmender Erfahrung und zunehmendem Wissen zentralere und vor allem vermehrt auch administrative Aufgaben übernehmen [Jensen2007]. Dieser Prozess ist dabei nicht nur auf die Weiterentwicklung des Quellcodes beschränkt. Jensen und Scacchi identifizieren im Netbeans-Projekt darüber hinaus Laufbahnen in den Bereichen Qualitätssicherung, Benutzerschnittstellen, Marketing und Webdesign.

Organisation der Aktionen

Neben der Art der Aktivitäten ist vor allem auch der Grad ihrer Zentralisierung ein wichtiges Kriterium für die Ausprägung der sozialen Netzwerke in Open Source Projekten. Entsprechend der angenommenen starken Hierarchisierung in erfolgreichen Open Source Projekten erfolgt dort auch die Entwicklung der Software zentralisiert. Dabei muss allerdings wie angesprochen zwischen Arbeiten am Kern einer Anwendung und Arbeiten im Bereich der Programmerweiterungen unterschieden werden. Je weiter sich die Aufgaben vom Kern einer Anwendung entfernen, desto niedriger ist der Grad der zentralen Kontrolle [Healy2003].

Darüber hinaus muss differenziert werden zwischen erzwungener Zentralisierung aufgrund einer geringen Anzahl von Entwicklern und Zentralisierung aufgrund organisatorischer Aspekte.

Der letzte Punkt bezieht sich dabei vor allem auf die Folgen von *„Brooks Law“* [Brooks2000]. Dieses Gesetz des Projektmanagement besagt, dass das Hinzufügen zusätzlicher Ressourcen in Form von Entwicklern zu Softwareprojekten, die außerhalb ihres Zeitplanes sind, die Entwicklung der Anwendungen zusätzlich verzögert. Der Grund dafür besteht in der stark anwachsenden Komplexität der Kommunikation innerhalb einer größeren Entwicklergruppe. Diese Zunahme im Koordinationsaufwand wirkt sich stärker negativ auf die Produktivität des Entwicklungsteams aus, als die zusätzlichen Entwicklungsressourcen ihn positiv beeinflussen können.

Infolge dessen ist es vor allem für große Open Source Projekte effektiv, die Entwicklungsarbeit wie von Mockus *et al.* [Mockus2002] postuliert in Form von

Subprojekten zu delegieren, die zentral von einer kleinen Gruppe von hauptverantwortlichen Kernentwicklern geführt werden. Die von Verfechtern der Open Source Bewegung angenommene Aussetzung von „Brooks Law“ [Raymond2001] beschränkt sich dabei lediglich auf die Prozesse zum Aufspüren von Programmfehlern [Michlmayr2004] sowie teilweise auf die Implementierung von Plug-Ins zur Software.[58]

Interaktionen der beteiligten Individuen

Der dritte Aspekt schließlich, der die Ausprägung der sozialen Netzwerke in Open Source Projekten bestimmt, sind die Interaktionen der beteiligten Personen. Die Kommunikation über Mailinglisten, Internet Relay Chat, Foren, Wiki oder die Website bildet den Leim zwischen den beteiligten Personen einerseits und ihren Aktivitäten andererseits. Neue Projektmitglieder müssen geworben, die Entwicklungsarbeiten koordiniert und Fragen beantwortet werden. Da die Softwareentwicklung in Open Source Projekten typischerweise auf freiwilligem Engagement der beteiligten Personen beruht, ist es schwierig, eine Planung und Steuerung der Entwicklungsarbeiten lediglich durch zentrale Vorgaben zu regulieren. Bestehende Abhängigkeiten zwischen den Aufgaben unterschiedlicher Entwickler werden entweder durch soziale Konventionen oder durch Interaktionen der Betreffenden koordiniert [Rothfuss2002].

Entscheidende Punkte in diesem Zusammenhang sind einerseits die Verfügbarkeit der genannten Kommunikationswege [59] sowie andererseits die Organisation der Kommunikation. Raymond entsprechend erfolgen die Planung und die damit verbundenen Kommunikationsprozesse in Open Source Projekten dezentral [Raymond2001], d.h. jedes Projektmitglied kommuniziert mit jedem anderen, ohne dass der entstehende Kommunikationsfluss zentral reguliert oder kontrolliert wird. Dies besitzt den Vorteil, dass ein direkter und spontaner Informationsaustausch zwischen allen Mitgliedern des Projektes möglich ist. Der Nachteil besteht jedoch darin, dass auch redundante bzw. nicht zielführende Interaktionen Teil des Kommunikationsprozesses sind.

Dazu berichtet Alan Cox, einer der Kernentwickler des Open Source Projektes *„Linux für 8086“*, in einem Aufsatz von 1998, dass nur wenige Entwickler innerhalb des Projektes sowohl das Interesse wie auch die Fähigkeiten hatten, die Linux-Software weiterzuentwickeln. Allerdings gab es viele *„[…] dangerously half-clued people […]“* [Cox1998], die zwar eine Meinung zu bestimmten Themen hatten, jedoch nicht die erforderlichen Kenntnisse. Das Ergebnis war eine Organisationsform, die Cox selbst als

---

[58] siehe Kapitel 4.4.3 „Resultierende Stärken von Open Source Projekten“

[59] siehe auch Kapitel 4.3.3 „Infrastrukturkomponenten“

„town council“ bezeichnet, in der jeder seine Meinung frei äußern konnte, die aber auch mit vielen uneffektiven Nebengeräuschen verbunden war.

Um diese zu vermeiden, sahen sich die Kernentwickler gezwungen, Teile der Gemeinschaft durch die Bildung von *kill-files*[60] von der aktiven Teilnahme an der Projektmailingliste auszuschließen, so dass ein *„core team“* entstand, das das Zentrum der Kommunikation bildete. Die Bezeichnung von Cox für die entstehende Organisationsform lautet *„clique“*. Damit will er andeuten, dass die dezentrale Kommunikation zugunsten einer stärkeren Kontrolle aufgegeben wurde. Dass dieses Vorgehen nicht optimal ist, um die Ressource „Entwickler“ effektiv zu nutzen, gesteht er selbst ein. Stattdessen sollten die Strukturen innerhalb eines Open Source Projektes derart gestaltet sein, dass die erfahrenen Entwickler der übrigen Gemeinschaft zeigen, wie sie produktiv zum Fortschritt des Projektes beitragen kann.

Im Gegensatz zu Raymond gehen Crowsten und Howison [Crowsten2005] nicht von einer prototypischen Ausformung der Organisation der Kommunikation in Open Source Projekten aus. In einer Untersuchung von 120 Projekten von Sourceforge zum Zentralisierungsgrad der Interaktionen in den Bug-Tracking-Systemen der Projekte, identifizieren sie stark auseinander gehende Ergebnisse. Gemessen wird der Zentralisierungsgrad dabei anhand der Konzentration der Verteilung ausgehender Nachrichten (siehe Abb. 7).

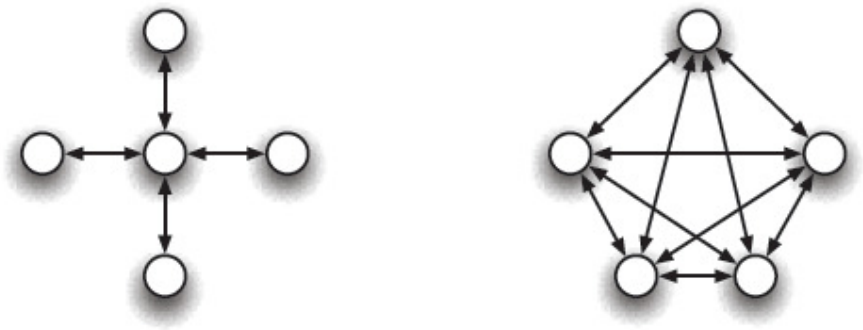

Abbildung 7: Maximaler und minimaler Zentralisierungsgrad von Interaktionen in einem sozialen Netzwerk

Demzufolge hängt die von Raymond angenommene dezentrale Planung in Open Source Projekten nicht direkt mit dem Ansatz der offenen Softwareentwicklung zusammen, sondern ist eine spezifische Ausprägung bestimmter Projekte. Crowsten und Howison entdecken darüber hinaus allerdings einen Zusammenhang zwischen der Größe eines

[60] Ein „kill-file“ ist eine Datei zur Herausfilterung uninteressanter Nachrichten oder Personen, die ausgeschlossen werden sollen

Projektes, gemessen an der Anzahl der aktiven Personen im Bug-Tracker, und dem Zentralisierungsgrad der Interaktionen.

Demzufolge besteht der Trend, dass die Kommunikationsprozesse in größeren Open Source Projekten dezentraler verlaufen als in entsprechend kleineren Projekten. Als Grund dafür vermuten Crowsten und Howison [Crowsten2005] die steigende Komplexität wachsender Open Source Anwendungen. Im Zuge dessen verteilen sich die Entwickler entsprechend ihrer Kompetenzen auf die unterschiedlichen Module der Software, so dass größere Open Source Projekte effektiv als die Summe mehrerer kleinerer Projekte auf Modulebene angesehen werden können. Dies deckt sich mit den bereits genannten Annahmen von Mockus *et al.* [Mockus2002] zu den Strukturen von Open Source Communities. Folglich verlaufen auch die Kommunikationsprozesse in Richtung der Personen mit den größten Kompetenzen, die sich in wachsenden Anwendungen allerdings auf unterschiedliche Module einer Software verteilen.

Zusammenfassung der sozialen Netzwerke in Open Source Projekten

Zusammenfassend sind die sozialen Netzwerke in Open Source Projekten geprägt durch das Zusammenspiel einer Gruppe von Kernentwicklern, die in erfolgreichen Projekten in der Regel aus professionellen Softwareentwicklern besteht, und einer um ein bzw. zwei Größenordnungen größeren Gruppe von Co-Entwicklern bzw. aktiven Benutzern der Software. Die Entwicklung der Anwendung ist gekennzeichnet durch eine starke Kontrolle im Kern sowie eine teilweise liberalere Vorgehensweise im Bereich der Programmerweiterungen. Die Hierarchien innerhalb von Open Source Projekten basieren, sofern nicht anders formuliert, auf Kriterien wie Wissen um die Software bzw. Grad und Dauer der Involvierung im Projekt, wodurch ein interner Aufstieg auch für außen stehende Entwickler möglich wird. Die wesentlichsten Motive für Entwickler zur Beteiligung in Open Source Projekten sind die Verbesserung der eigenen Fähigkeiten sowie die Bewältigung einer Herausforderung. Darüber hinaus sind allerdings auch die Verbesserung der Berufschancen und direkte finanzielle Anreize Gründe für ein Engagement vor allem in späteren Phasen der Beteiligung. Die Organisation der Kommunikationsprozesse ist nicht generalisierbar, sondern besitzt eine projektspezifische Ausprägung. Allerdings besteht der Trend, dass die Interaktionen in großen Projekten zunehmend dezentraler erfolgen. Dies lässt die Annahme zu, dass größere Open Source Projekte in kleinere Subprojekte untergliedert sind, in denen die Kommunikation dann in Richtung der Kernentwickler des jeweiligen Moduls verläuft.

4.4.2 Wirtschaftliche Rahmenbedingungen in Open Source Projekten

Da die Arbeit in größer und komplexer werdenden Open Source Projekten sehr viel Zeit in Anspruch nimmt, stellt sich vor allem vor dem Hintergrund nachlassender Entwicklermotivation die Frage nach einem finanziellen Ausgleich für deren

Entwickler. Dies gilt insbesondere, betrachtet man die Tatsache, dass in einigen Projekten die Weiterentwicklung der Software so aufwändig ist, dass die Freizeit der Kernentwickler dafür nicht mehr ausreicht.[61]

Der traditionelle Weg zur Generierung von Einnahmen, die Erhebung einer Lizenzgebühr, entfällt dabei in Open Source Projekten normalerweise. Zudem besteht in der Praxis das Problem, im Zuge der weiten Verteilung der Projektmitglieder eine faire Aufteilung von Geldern zu gewährleisten, sodass meist nur die aktivsten Teilnehmer eines Projektes von vorhandenen Einnahmen profitieren [Rothfuss2002]. Dennoch ist eine kommerzielle Nutzung der Software möglich und widerspricht auch nicht der Philosophie der Freien Softwarebewegung.[62]

Spenden

Die erste Möglichkeit für Open Source Projekte zur Generierung von Einnahmen besteht in der Annahme von Spenden. Spenden können von Privatpersonen, von interessierten Unternehmen oder von gemeinnützigen Organisationen zur Unterstützung der Open Source Bewegung stammen.

Spenden von Unternehmen

Die Motive von Unternehmen zur Finanzierung von Open Source Projekten können vielschichtig sein.[63] Generell sind allerdings zwei grundlegende Fälle zu unterscheiden, die mit unterschiedlichen Historien der Projekte in Verbindung stehen.

Im ersten Fall erfolgt die Initiierung von Open Source Projekten durch Unternehmen. Im Zuge dessen investieren Softwareunternehmen zunächst eigene Entwicklungsarbeit in die Erschaffung eines Softwareprototyps und erhoffen sich dann im nächsten Schritt durch eine Offenlegung des Quellcodes den Zugriff auf externe Ressourcen. Populäre Beispiele dafür sind die Freigabe des Navigator-Codes durch die Firma Netscape 1998, die Gründung des OpenOffice-Projektes durch die Firma Sun oder die Veröffentlichung des Quellcodes des Java-Compilers der Firma IBM, die zur Gründung des *Jikes-Projektes* führte [O'Mahony2005]. Die besondere Schwierigkeit für Unternehmen besteht dabei darin, externe Entwickler für ein Engagement im eigenen Projekt zu motivieren. Dies hat mehrere Gründe. Zum einen fehlt kommerziellen Unternehmen innerhalb der Open Source Bewegung oftmals die Glaubwürdigkeit. Zum anderen erschweren die meist bereits vorhandene hohe Komplexität der Anwendung

---

[61] Ein Beispiel dafür ist das Projekt „dillo Web Browser" auf www.dillo.org, das von lediglich zwei Entwicklern geführt wird, deren Kapazitäten für eine Weiterentwicklung der Software nicht ausreichen (siehe http://lists.auriga.wearlab.de/pipermail/dillo-dev/2006-September/003097.html)

[62] siehe http://www.gnu.org/philosophy/selling.html

[63] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen"

[O'Mahony2005] sowie die nur begrenzt vorhandene Möglichkeit, sich durch die Prägung des Projektes einen Ruf zu verschaffen, den Einstieg neuer Entwickler. Zudem sind die Unternehmen häufig nicht gewillt, die Kontrolle am Projekt vollständig abzugeben, sodass es für die entsprechenden Softwareunternehmen bereits aus Gründen der Entwicklermotivation unerlässlich ist, auch finanzielle Anreize zu geben.

Im zweiten Fall unterstützen Unternehmen bereits bestehende Open Source Projekte, um die Weiterentwicklung der Software in den Projekten abzusichern. Dies kann wie erwähnt unterschiedliche Hintergründe haben. Verbreitete Motive sind dabei vor allem für Anwendungsunternehmen die Verringerung von Herstellerabhängigkeiten sowie die Reduzierung von Softwarekosten in Verbindung mit einer Migration auf Open Source Software. Darüber hinaus bietet sich Softwareunternehmen die Chance, durch die Verwendung von offenen Softwarekomponenten eigene Entwicklungsarbeit einzusparen.[64]

Spenden von gemeinnützigen Organisationen

Neben interessierten Unternehmen unterstützen auch einige gemeinnützige Organisationen die Entwicklungen in Open Source Projekten. Diese bilden sich dabei häufig im Umfeld erfolgreicher Projekte, die im Zuge ihres Wachstums einen juristischen Arm zur Formalisierung ihrer internen Strukturen benötigen. Damit verbunden sind die Regelung von Fragen, die sich aus Urheberrechtsansprüchen unterschiedlicher Entwickler im Projekt ergeben, die Formierung einer rechtlichen Instanz nach außen sowie vor allem auch die Annahme und Weiterverteilung von externen Spendengeldern, die zur Deckung der Kosten für Infrastruktur und Entwickleraufwand verwendet werden. Mit zunehmendem Popularitätsgrad der Projekte und damit verbunden der entsprechenden Organisationen werden unter Umständen auch andere Open Source Projekte in die Förderprogramme der Organisationen mit aufgenommen. So unterstützt die *Apache Software Foundation* [65] beispielsweise inzwischen zahlreiche Projekte, die sie als qualifiziert erachtet, um in die Förderung integriert zu werden. Weitere Organisationen zur Unterstützung von Open Source Projekten sind die Free Software Foundation (FSF)[66], die Open Source Initiative (OSI)[67] oder die Mozilla Foundation.[68]

---

[64] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen"

[65] sieh http://www.apache.org/foundation/

[66] siehe http://www.fsf.org

[67] siehe http://www.opensource.org

[68] siehe http://www.mozilla.org/foundation

Weitere Einnahmequellen

Dienstleistungen

Eine weitere Einnahmequelle für Open Source Projekte besteht im Anbieten von Dienstleistungen zur Software. Dienstleistungen können dabei in verschiedenen Formen erfolgen. Etabliert haben sich in Verbindung mit Open Source Software im Wesentlichen das Zusammenstellen fertiger, leicht installierbarer Distributionen, das Anbieten von technischer Unterstützung für Benutzer, das Anpassen der Software an spezifische Umstände, das Anbieten von Schulungen und Weiterbildungen sowie die Zertifizierung bestimmter Softwareversionen. Häufig stehen solche Dienstleistungen auch im Zusammenhang mit Unternehmen, die sich im Umfeld der Open Source Projekte bilden.[69]

Dual Licensing

Eine besondere Form des Vertriebs von Open Source Software ist das so genannte *„Dual Licensing“*. Dabei veröffentlichen die Urheber einer Software diese einerseits unter einer Open Source Lizenz, andererseits erfolgt aber gleichzeitig die Verbreitung unter einer proprietären Lizenz in Verbindung mit der Erhebung von Lizenzgebühren. Dies hat zwei Hintergründe.

Zum einen ermöglicht es eine Unterscheidung zwischen privaten bzw. öffentlichen Endnutzern, die die Software lediglich unverändert einsetzen, und Softwareunternehmen, die sie in die eigene, proprietär vertriebene Anwendung integrieren wollen. In diesem Fall bezahlen lediglich die Softwareunternehmen für eine Integration des Programms in die eigene Software eine Lizenzgebühr, während bloße Anwender die Open Source Lizenz nutzen können.

Zum anderen bietet es die Möglichkeit zur Zertifizierung bestimmter Versionen eines Softwarepaketes. Da Open Source Lizenzen einen Haftungsausschluss enthalten, ist es sowohl für Anwender wie auch für Hersteller eigener Programme wichtig, darüber hinaus Qualitätsgarantien auf ausgezeichnete Versionen einer Software zu erhalten. In diesem Zusammenhang kann das „Dual Licensing“ dazu benutzt werden, ältere Programmversionen kostenfrei, die aktuelle, zertifizierte Version dagegen lizenzgebührenpflichtig anzubieten [Clement2002].

Merchandising

Neben den genannten Formen der Generierung von Einnahmen nennt die OSI darüber hinaus den Begriff des *„Accessorizing“* (dt. „Ausstattung“) im Zusammenhang mit möglichen, auf Open Source Software basierenden Geschäftsmodellen. Gemeint ist

---

[69] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

damit das Anbieten verschiedener Produkte zu Open Source Projekten und ihrer Software. Mögliche Produkte sind dabei zur Software korrespondierende Hardwarekomponenten, Bücher, T-Shirts oder andere Merchandising-Artikel. Etabliert haben sich dabei vor allem der Verkauf von Hardware sowie Büchern zu Open Source Software.[70]

Werbung

Schließlich bildet Werbung eine Möglichkeit zur Finanzierung der projektinternen Entwicklungsarbeit. Werbung erfolgt dabei durch die Verlinkung der Projekthomepage mit Webseiten von Unternehmen und anderen Organisationen gegen eine entsprechende Bezahlung.

4.4.3 Resultierende Stärken von Open Source Projekten

Entwicklerzahl

Erfolgreiche Open Source Projekte wie das Linux-Projekt, der Apache-Webserver oder das Mozilla-Projekt besitzen eine Vielzahl von Benutzern und Entwicklern, die aktiv in die Weiterentwicklung des Projektes und seiner Software involviert sind. Dies spiegelt sich vor allem in einem gut funktionierenden Prozess zur Pflege der Software wider. Während die Partizipation an der Weiterentwicklung des Kerns einer Software limitiert ist durch Faktoren wie den Grad seiner Modularisierung [Brooks2000], das genaue Wissen um die Architektur sowie organisatorische Aspekte, unterliegen die Prozesse zur Pflege einer Software kaum Einschränkungen in Bezug auf die Anzahl der beteiligten Personen. Aktivitäten zur Pflege einer Software sind das Aufspüren und Beheben von Fehlern, das Entwickeln von Dokumentationen, das Einsenden von Änderungswünschen oder auch die Entwicklung zusätzlicher Features. Letzteres setzt allerdings voraus, dass die Software eine stark modularisierte bzw. eine Plug-In-Struktur besitzt. Da der Aufwand für Aktivitäten zur Pflege und Wartung einer Software innerhalb ihres Lebenszyklus erfahrungsgemäß um Faktor 2 bis 4 höher liegt als für ihre eigentliche Entwicklung [Balzert2000], spielt die große Anzahl von Entwicklern, die in die Pflege der Software involviert sind, in erfolgreichen Open Source Projekten eine wesentliche Rolle für die Qualitätssicherung.

Parallelitätsgrad

Neben einer hohen Anzahl von Entwicklern ist auch der *Grad der Parallelität* der Aktivitäten in Open Source Projekten ein wichtiger Vorteil, der sich aus dem offenen, verteilten Entwicklungsprozess ergibt. Dabei gibt es zwei Formen der Parallelität. Zum

[70] siehe http://www.opensource.org/advocacy/case_for_business.php

einen ermöglicht die offene Entwicklung eine parallele Implementierung der Software, d.h. mehrere Entwickler arbeiten zugleich an der Weiterentwicklung des Quellcodes. Zum anderen erfolgt auch eine parallele Bearbeitung traditionell verschiedener Entwicklungsphasen. Während das erste wie zuvor erwähnt durch den Grad der Modularisierung der Software begrenzt wird, beschreibt letzteres die Aufhebung der sequentiellen Abarbeitung von Implementierungs-, Test- und Wartungsphase. Damit werden kürzere Releasezyklen möglich. Allerdings ist ein solches Prozessmodell nicht wie häufig dargestellt auf Open Source Projekte beschränkt [Fugetta2003]. Es existieren auch traditionelle Prozessmodelle, die im Gegensatz zum *Wasserfallmodell*[71] einen höheren Grad an Parallelität in der Abarbeitung der einzelnen Entwicklungsphasen beinhalten. So erfolgt im Prototypenmodell die Entwicklung verschiedener Prototypen der Software, um bereits während der Entwicklungsphase ein Feedback des Kunden zu erhalten. Vergleichbar dazu zielen auch das evolutionäre sowie das inkrementelle Modell auf einen früheren Kontakt mit dem Kunden ab. Der Unterschied besteht darin, dass die früh realisierten Kernfunktionen bereits funktionsfähig sein müssen. Auf die Spitze getrieben wird der Grad der Parallelität im so genannten nebenläufigen Modell. Hierbei wird versucht, innerhalb eines großen Teams die einzelnen Entwicklungsaktivitäten so weit wie möglich zu parallelisieren [Balzert1998]. Dies birgt natürlich auch das Risiko, dass bereits realisierte Arbeiten aufgrund von neuen Erkenntnissen hinfällig werden. Für den Vergleich mit Open Source Projekten bedeutet dies, dass nicht das Prozessmodell an sich revolutionär ist, sondern nur der Grad der Skalierung [Fugetta2003]. Die Vorteile liegen auch hier vor allem im großen Potential an Entwicklern.

> *„Linus's innovation was not so much in doing quickturnaround releases incorporating lots of user feedback (something like this had been Unix-world tradition for a long time), but in scaling it up to a level of intensity that matched the complexity of what he was developing."*
>
> *- Eric Raymond [Raymond2001, S.29]*

Innovationspotential

Eine weitere Stärke von Open Source Projekten ist das hohe *Innovationspotential*. Dieses Argument bezieht seine Berechtigung im Wesentlichen aus der Vielschichtigkeit der beteiligten Projektmitglieder. Während die Entwicklungen in Softwareunternehmen lediglich von einem Team von Entwicklern mit meist ähnlichen Interessen und Erfahrungen gesteuert werden, bietet die Offenheit von Open Source Projekten den verschiedensten Benutzern der Software die Möglichkeit, den Entwicklungsprozess in

[71] Das Wasserfallmodell ist ein Prozessmodell, bei dem die einzelnen Phasen sequentiell durchlaufen werden

den Projekten zu beeinflussen oder zumindest nachzuvollziehen. Dementsprechend erlangen weit verbreitete Open Source Projekte Feedback aus verschiedenen Perspektiven und somit auch innovative Einflüsse.

Konformität des Einsatzgebietes der Entwickler

Schließlich spielt auch die freie Wahl des Einsatzgebiets der Entwickler in Open Source Projekten eine wesentliche Rolle. Da die Projektmitglieder ihre Aufgaben entsprechend ihren Fähigkeiten selbst wählen können, werden ihre Kenntnisse auch produktiv genutzt. Zudem besitzt eine angemessene Wahl des Einsatzgebietes einen großen Einfluss auf die Motivation und damit verbunden auf die Konzentration der Entwickler. Lakhani und Wolf identifizieren dazu einen Zustand, den sie mit dem Begriff *„Flow“* bezeichnen. Dieser entsteht, wenn die Fähigkeiten einer Person genau den Herausforderungen einer bestimmten Aufgabe entsprechen und ist charakterisiert durch intensive Konzentration und Fokussierung [Lakhani2005][Raymond2001].

4.4.4 Resultierende Schwächen von Open Source Projekten

Entwicklerzahl

Neben den genannten Stärken besitzt der Entwicklungsprozess in Open Source Projekten auch einige Schwächen, die vor allem in weniger erfolgreichen Projekten deutlich werden. Ein wesentlicher Punkt besteht darin, dass nur wenige Open Source Projekte eine vergleichbare Anzahl von Entwicklern zu erfolgreichen Projekten wie dem Linux- oder dem Debian-Projekt besitzen. So ergab eine Untersuchung von 100 als „mature“ [72] eingestuften Projekten von Krishnamurthy aus dem Jahr 2002 [Krishnam2002] z.B. eine durchschnittliche Entwicklerzahl von nur einem Entwickler. Dies bedeutet, dass das angeführte Argument der hohen Skalierbarkeit nur in einigen Projekten überhaupt zutrifft. Darüber hinaus besteht häufig eine hohe Abhängigkeit von einer kleinen Gruppe von so genannten Kernentwicklern, die den größten Teil der Arbeit erledigen und das größte Wissen über die Software besitzen [Healy2003][Gorling2003]. Verlassen Kernentwickler das Projekt, so ist dies schwer zu kompensieren.

Entwicklerzentrierte Softwareentwicklung

Ein weiteres Problem von Open Source Projekten ist, dass die Entwicklung der Software nicht benutzer- sondern entwicklerzentriert erfolgt, da meist keine direkte finanzielle Bindung an Kunden und Endnutzer besteht. Zudem bewirkt die

[72] Die Bezeichnung „mature“ steht für den Reifegrad einer Software. Demnach ist eine Software, die mature ist, qualitativ sehr ausgereift.

Freiwilligkeit der Arbeit, dass bevorzugt Codierbeiträge geleistet werden. Weniger motivierende Aufgaben wie das Erstellen von Dokumentationen, die darüber hinaus mit einer geringeren Aussicht auf Anerkennung verbunden sind, werden oftmals vernachlässigt [Rothfuss2002]. Das Resultat ist teilweise unzureichende Dokumentation sowie unübersichtliche Benutzeroberflächen. Da solche Aspekte wichtig für potentielle Benutzer der Software sind, hat sich in diesem Zusammenhang ein Geschäftsmodell etabliert, das auf dem Support zu bestehender Open Source Software basiert.[73]

Hohe Skalierbarkeit vs. Chaotische Entwicklung

Zusätzlich zur Entwicklerzentrierung stellt sich die Frage nach dem von Raymond angeführten hohen Grad der Skalierbarkeit des offenen Entwicklungsprozesses [Raymond2001]. Da er in Open Source Projekten keine Abhängigkeiten von zentralen Projektmanagern vermutet, deren Engagement nur begrenzt skalierbar ist, sondern eine große, sich selbst organisierende Gemeinschaft von Gleichgesinnten, postuliert Raymond, dass die Entwicklungen und Prozesse in den Projekten in einem hohen Maße optimierbar sind. Dabei vernachlässigt er allerdings den Aspekt, dass gerade in erfolgreichen Open Source Projekten kompetente, von der Community akzeptierte Autoritäten hinter der Entwicklung der jeweiligen Software stehen. Dies bestätigt Connell, Softwareberater und Dozent der Softwaretechnik an der Universität von Boston, der dazu unter anderem Aussagen von Linus Torvalds aus einem Interview zitiert [Connell2000]. Demnach ist der Prozess der Softwareentwicklung in Open Source Projekten sehr stark an ihre führenden Köpfe gebunden, die als Projektmanager agieren, wodurch die hohe Skalierbarkeit des Prozesses in Frage gestellt wird. Darüber hinaus argumentiert Connell, dass sich das Fehlen entsprechender Autoritäten nicht in einem selbst organisierenden Entwicklungsprozess manifestiert, sondern in eher chaotischen Zuständen. Dies Problem nennt auch Rothfuss [Rothfuss2002], der in diesem Zusammenhang vor allem auch die erforderlichen sozialen Kompetenzen der Projektadministratoren zur Koordination des Entwicklungsprozesses betont.

Interessenkonflikte

Vor diesem Hintergrund können sich vor allem auch verschiedene Interessen, die innerhalb einer Open Source Community bestehen, zu einem Problem entwickeln. Dies kann sich in zwei verschiedenen Ausdrucksformen widerspiegeln. Da aufgrund der Offenheit von Open Source Projekten vielschichtige Anforderungen an die Software aufeinanderprallen, kann dies zu einem so genannten *„Feature Creep“* führen. Dabei entwickeln verschiedene Entwickler entsprechend ihren Anforderungen fortwährend neue Funktionalitäten, sodass das systematische Testen und Verbessern zur Erstellung

[73] siehe Kapitel 4.3.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

einer stabilen Version der Software nicht möglich ist. In diesem Fall ist es die Aufgabe des Release Managers, den Umfang der implementierten Funktionalitäten einzufrieren.

Kritischer ist dagegen die zweite Form des Interessenkonflikts. Bestehen zwischen großen Teilen der Projektgemeinschaft grundlegende Differenzen hinsichtlich der Prioritäten der Entwicklungsrichtung, so kann dies zu einer Gabelung oder *„Fork“* des Projektes führen. In diesem Fall spaltet sich ein Teil der Projektgemeinschaft ab, um die Weiterentwicklung der Software in die eigene Richtung voranzutreiben. Dies kann vor allem in kleineren Projekten zur Behinderung oder sogar zum Stillstand der Aktivitäten führen, wenn die Größe der Projektgemeinschaft danach nicht mehr der Komplexität der Aufgabe entspricht. Mittel, diesem Problem entgegenzukommen, sind die Wahl einer geeigneten Lizenz sowie eine möglichst frühe Formulierung von gemeinsamen Projektzielen.

Unvorhersehbarkeit der Weiterentwicklung

Schließlich besteht eines der größten Probleme von Open Source Projekten in der Unvorhersehbarkeit der Motivation der Entwickler. Da die Aktivitäten in Open Source Projekten zu einem großen Teil auf der Basis einer freiwilligen Teilnahme der Mitglieder erfolgen, besteht keine Garantie, dass die Entwickler auch zukünftig motiviert sind, an der Weiterentwicklung der Software zu arbeiten. Dies ist vor allem für Unternehmen ein kritischer Punkt, da die Migration auf eine Open Source Software zunächst mit Migrations- und Schulungskosten verbunden ist und sich langfristig rechnet oder rechnen kann.[74] Erheblich verringert wird dieses Risiko durch die Etablierung einer breiteren Gruppe von Kernentwicklern, da die Abhängigkeit von einzelnen Personen verringert wird, sowie durch die Existenz von kommerziellen Interessen eines Open Source Projektes [Stuermer2005].

4.4.5 Ziele eines Einsatzes von Open Source Software in Unternehmen

Entsprechend ihrem Geschäftsmodell verfolgen Unternehmen unterschiedliche Ziele mit einem Einsatz von Open Source Software. Diese können vielschichtig sein. Einerseits können direkte finanzielle Interessen ausschlaggebende Faktoren für eine Ausrichtung der Infrastruktur oder des eigenen Portfolios auf Open Source Software sein, andererseits können auch strategische Überlegungen eine wichtige Rolle spielen. In einer Analyse der Anforderungen von Unternehmen beschreiben Cruz *et al.* [Cruz2006] dazu verschiedene Szenarien für einen Einsatz von Open Source Software. Die Unternehmensziele, die mit den entsprechenden Szenarien verbunden sind, lassen sich in die Kategorien *politische*, *wirtschaftliche* und *technische Ziele* unterteilen.

---

[74] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

Politische Ziele

Herstellerunabhängigkeit

Ein wesentliches politisches Ziel vor allem für Anwendungsunternehmen ist die Verringerung der Abhängigkeit von bestimmten Softwareherstellern. Softwarehersteller wie *Microsoft* versuchen häufig, durch die Etablierung eigener, proprietär vertriebener Formate langfristige Abhängigkeiten zwischen Herstellern und Kunden zu schaffen. Dies geschieht dabei meist durch einen preiswerten Einstieg sowie spätere hohe Lebenszykluskosten [Ebert2004].

Die Nachteile einer solchen Konstellation für Unternehmen bestehen dabei in der engen Bindung an die Preise, Releasezyklen und weiteren Entwicklungspläne der Softwarehersteller. Hier kann Open Source Software eingesetzt werden, um vorhandene Abhängigkeiten zu verringern. Die Offenheit des Quellcodes ermöglicht es unterschiedlichen Unternehmen, Dienstleistungen zu einer Software anzubieten. Dies erhöht den Konkurrenzdruck für Serviceanbieter in diesem Bereich und bewirkt somit eine bessere Kundenorientierung [Clement2002]. Darüber hinaus bietet es Unternehmen die Möglichkeit, jederzeit selbstständig Änderungen an einer Software vorzunehmen bzw. diese gezielt zu beauftragen.

Marketingstrategien

Der zweite Bereich für politische Ziele ist vor allem für Softwareunternehmen relevant und besteht in speziellen Marketingstrategien.

Dazu nennt die OSI in einer Beschreibung möglicher, auf Open Source Software basierender Geschäftsmodelle den Begriff des *„Loss Leader“*.[75] Bei diesem Modell geben Unternehmen bestimmte Produkte, die nicht zu ihren Kernprodukten gehören, preisgünstig ab, um das Kerngeschäft dagegen zu fördern. Das wohl bekannteste Beispiel dieser Kategorie ist die Freigabe des Quellcodes des Internet-Browsers „Navigator“ durch die Firma Netscape. Der damals marktführende Browser drohte, seine Position an Microsofts „Internet Explorer“ zu verlieren. Da der Browsermarkt nicht zum Kerngeschäft des Unternehmens gehörte, sondern lediglich ein Feld zur besseren Vermarktung des Serverbereiches[74] [Gorling2003] darstellte, entschied sich Netscape für eine Freigabe des Quellcodes, um die Monopolstellung Microsofts anzufechten [Raymond2001]. Dies sollte zum einen die Aufmerksamkeit wieder auf den Serverbereich Netscapes lenken. Zum anderen ermöglichte die Kontrolle des Browsermarktes auch die Kontrolle über den Prozess der HTML-Standardisierung [Gorling2003][Raymond2001].

---

[75] siehe http://www.opensource.org/advocacy/case_for_business.php

Eine weitere strategische Anwendungsmöglichkeit der Öffnung des Quellcodes einer Software ist das von der OSI mit dem Begriff *„Widget Frosting“* bezeichnete Geschäftsmodell von Hardwareherstellern[74] [Raymond2001]. Im Zuge des Vertriebs der eigenen Hardware sind Hardwarefirmen meist gezwungen, auch Software zur Steuerung und Konfiguration der entwickelten Geräte zu entwickeln und zu pflegen. Solche Anwendungen umfassen dabei ein Spektrum von Gerätetreibern über Konfigurationstools bis hin zu kompletten Betriebssystemen. Da die Entwicklung und die Wartung solcher Softwareanwendungen nicht zu den Einnahmequellen der Unternehmen zählen, sehr wohl aber einen erforderlichen Service darstellen, bilden sie einen erheblichen finanziellen und zeitlichen Zusatzaufwand für die Firmen.

Vor diesem Hintergrund bringen die Offenlegung des eigenen Quellcodes bzw. die Unterstützung vergleichbarer Open Source Projekte für Hardwarehersteller zwei entscheidende Vorteile mit sich.

Erstens ermöglichen die Weiterentwicklung, Pflege und Wartung der Anwendungen innerhalb eines Open Source Projektes den Zugriff auf externe Entwicklungsressourcen. Dies ist vor allem im Zusammenhang mit der Tatsache, dass die Wartungs- und Pflegekosten einer Software im Vergleich zu den ursprünglichen Entwicklungskosten wesentlich höher sind [Balzert2000], ein sehr relevanter Aspekt.

Zweitens bietet sich Kunden darüber hinaus die Möglichkeit, auch nach dem Ablauf der Garantiezeiten noch Unterstützung für ihre Hardware zu bekommen, sodass eine höhere Kundenzufriedenheit gewährleistet werden kann [Raymond2001].

Zwei Beispiele für Unternehmen, die zur Unterstützung ihrer Hardware Open Source Projekte finanziell fördern, sind die Firmen IBM und Silicon Graphics. IBM hat nach eigenen Angaben allein im Jahr 2001 über 1 Milliarde U.S. Dollar in die Anpassung des Apache-Webservers sowie des Linux-Betriebssystems an die IBM-Serverhardware investiert. Die Ziele waren dabei vor allem die Etablierung einer uniformen Serverplattform, bestehend aus IBM-Hardware und dem angepassten Linux-Betriebssystem sowie der anschließende Verkauf der eigenen Hardware [Kalwey2003]. Silicon Graphics unterstützt dagegen das *SAMBA-Projekt* [76], das eine Software entwickelt, die es Unix-Servern ermöglicht, das von Windows 95/NT/2000/XP-Servern verwendete *SMB-Protokoll* [77] zu benutzen und somit einen Windows-Server zu emulieren.

---

[76] siehe http://us3.samba.org/samba/

[77] Das SMB- oder „Server Message Block“-Protokoll ist ein von Microsoft und Intel entwickeltes Protokoll zur Aufteilung von Dateien, Druckern oder anderen Kommunikationsschnittstellen zwischen Computern

Neben den beiden Modellen des „Loss Leader“ und des „Widget Frosting“ ist für Softwareunternehmen schließlich auch die Etablierung von Standards ein interessanter strategischer Aspekt von Open Source Software. Dies gilt vor allem für Unternehmen, die nicht den führenden Standard in einem bestimmten Marktsegment besitzen. Die Vorgehensweise besteht dabei darin, die Plattform für die eigenen Produkte kostenlos abzugeben, um darauf aufbauende Produkte später verkaufen zu können.

Vor diesem Hintergrund stellt die Firma SUN beispielsweise die von ihr entwickelte Programmiersprache *Java* frei zur Verfügung, um im Gegenzug auf Java basierende Entwicklungswerkzeuge absetzen zu können [Clement2002]. Auch die bereits genannte Investition der Firma IBM in eine Anpassung des Linux-Betriebssystem an die eigene Serverhardware zielt auf die Verdrängung der Vielzahl unterschiedlicher Unix-Systeme in diesem Bereich sowie die Etablierung einer Standardarchitektur [Kalwey2003].

Ein wichtiger Aspekt im Zusammenhang mit diesem Geschäftsmodell ist, dass es nicht nur für bereits bestehende Märkte Anwendung findet, sondern vor dem Hintergrund, dass die Begründer eines Marktbereiches häufig den Standard in diesem Feld etablieren, vor allem auch eine Möglichkeit bietet, neue Tätigkeitsfelder zu schaffen [Clement2002].

Wirtschaftliche Ziele

Das zweite wesentliche Feld für Unternehmensziele verbunden mit einem Einsatz von Open Source Software sind direkte wirtschaftliche Interessen der Betriebe. Dazu muss erneut zwischen Anwendungsunternehmen, die Open Source Software lediglich nutzen, und Softwareunternehmen, die ihr Geschäftsmodell darauf aufbauen, unterschieden werden.

Reduzierung der Kosten

Das Hauptmotiv der ersten Gruppe besteht dabei in der Reduzierung der Softwarekosten. Da Open Source Software lizenzgebührenfrei erhältlich ist, können vor allem große Unternehmen, die eine Vielzahl von Arbeitsplatzrechnern mit den entsprechenden Softwaresystemen ausstatten müssen, erhebliche initiale Summen einsparen.

So kostet beispielsweise die Ausstattung von 25 Rechnern mit dem Betriebssystem *Microsoft Windows 2000* ein Unternehmen $1510 an Lizenzgebühren, während für eine professionelle Linux-Distribution der Firma Red Hat dagegen nur $156 Zusammenstellungsgebühr anfallen [Wheeler2005]. Darüber hinaus ist die Nutzung der Linux-Distribution im Gegensatz zum Windows-System von Microsoft nicht an eine bestimmte Anzahl von Rechnern gebunden, sondern erfolgt unlimitiert.

Allerdings umfassen die realen Kosten eines Produkteinsatzes nicht bloß die Lizenzgebühren. Hinzu kommen nach dem Ansatz der *Total Cost of Ownership (TCO)* [78] weitere direkte sowie indirekte Kosten. Weitere direkte Kosten sind Hardwareanschaffungs-, Administrations-, Support-, Schulungs- und Migrationskosten. Indirekte Kosten sind Endnutzerkosten [79] sowie Verluste, die aufgrund von Systemausfällen entstehen.

Ob eine Entscheidung für Open Source Software unter Berücksichtigung der Gesamtausgaben dabei die günstigere Wahl ist, entscheidet sich im Einzelfall und wird kontrovers diskutiert. Die Firma Microsoft benennt auf ihrem Webportal verschiedene Studien, die belegen, dass Unternehmen durch den Einsatz von Microsoft-Produkten gegenüber einem Einsatz vergleichbarer Open Source Software über 20% der TCO einsparen können.[80] Demgegenüber führt Wheeler zahlreiche Studien an, die zu einem gegenteiligen Ergebnis kommen [Wheeler2005].

Festzuhalten bleibt die Tendenz, dass eine Migration auf Open Source Software nach einer kostengünstigeren Anschaffungsphase gekennzeichnet ist durch eine kostenintensivere Schulungs- und Anpassungsphase. Diese Mehrausgaben scheinen sich jedoch langfristig aufgrund von günstigerem Support sowie geringeren Nachrüstungskosten im Vergleich zu proprietärer Software zu relativieren, sodass die Gesamtbetriebskosten abnehmen [Reiter2004]. Eine genaue Betrachtung muss dabei jedoch im Einzelfall erfolgen.

---

[78] Der Ansatz der Total Cost of Ownership ist ein Abrechnungsverfahren , nach dem nicht nur direkte sondern auch indirekte Kosten berücksichtigt werden müssen

[79] Endnutzerkosten sind Kosten, die dem Nutzer einer Software durch die Nutzung entstehen. Dies beinhaltet vor allem auch die Frage nach der effizienten Nutzung einer Software

[80] siehe http://www.microsoft.com/windowsserver/facts/casestudies/default.mspx

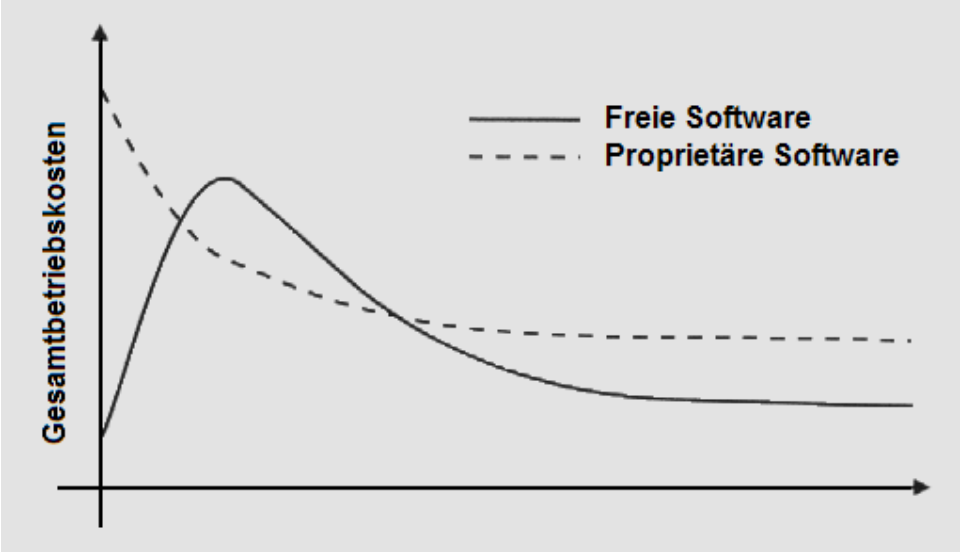

Abbildung 8: Abstrakter Gesamtbetriebskostenverlauf bei einer Migration auf Open Source Software

Verkauf von Open Source Software

Der zweite Fall in der Betrachtung wirtschaftlicher Ziele von Unternehmen sind Softwareunternehmen, die ihr Portfolio um Open Source Software erweitern oder ihr Geschäftsmodell komplett darauf ausrichten. Allgemein existieren dazu zwei Modelle zur Generierung von Einnahmen, die sich in Unternehmen allerdings durchaus überlappen können. Diese bestehen zum einen im Verkauf von Open Source Software, entweder allein stehend oder in Kombination mit Unternehmenssoftware, sowie zum anderen im Anbieten bestimmter Dienstleistungen zu Open Source Produkten.

Der lizenzgebührenpflichtige Verkauf von Open Source Software ist aufgrund entsprechender Bestimmungen der meisten Open Source Lizenzen in der Regel nicht möglich.[81] Zusätzlich erschweren solche Lizenzen darüber hinaus oftmals auch die gemeinsame Verbreitung einer Kombination aus proprietärer und offener Software. Dies hat den Grund, dass die Urheber der Software die von ihnen gewährten Freiheitsrechte auch im Zuge einer Weiterverbreitung der Software gesichert wissen wollen. Allerdings existieren auch weniger restriktive Lizenzen wie die BSD-Lizenz, die den gemeinsamen Vertrieb einer proprietären Anwendung und offenen Softwarekomponenten ermöglichen. Dies bietet Softwareunternehmen den Vorteil, Entwicklungs-, Pflege- und Wartungskosten einsparen zu können, da ein Teil der Entwicklungsarbeit an die Open Source Projekte abgegeben werden kann. Zum Ausgleich unterstützen diese Unternehmen dann die entsprechenden Projekte in Form

[81] siehe Kapitel 4.3.5 „Lizenzen“

von Spenden oder Entwicklungsarbeit, um die Weiterentwicklung der Software abzusichern und gleichzeitig steuern zu können [Clement2002].

Eine Alternative zum Verkauf von Lizenzen ist das Geschäftsmodell, das Firmen wie Red Hat, Mandrakesoft oder SuSE im Zusammenhang mit Linux etabliert haben. Dabei erfolgt nicht die Erhebung einer Lizenzgebühr, sondern einer Gebühr für die Erstellung fertiger Softwaredistributionen. Dies beinhaltet die Zusammenstellung der unterschiedlichen Softwarepakete in einer leicht installierbaren Form sowie die Ergänzung von Dokumentation und zusätzlichen nützlichen Funktionen. Darüber hinaus werden Qualitätsgarantien auf die Software mitgeliefert, da Open Source Lizenzen ansonsten einen Haftungsausschluss enthalten. Allerdings birgt dieses Geschäftsmodell für Unternehmen auch Risiken in sich. Zum einen ist die Konkurrenzsituation aufgrund des frei verfügbaren Quellcodes sehr hoch. Zum anderen investieren zunehmend auch große Unternehmen wie IBM oder SUN in diesen Bereich, wie die Zusammenarbeit von IBM und SuSE sowie Sun und Mandrakesoft zeigt [Clement2002]. Dies erschwert die Etablierung eines eigenen Markennamens für unabhängige Unternehmen in diesem Feld.

Die dritte Möglichkeit zum Verkauf von Open Source Software ist schließlich die bereits erwähnte Verbreitungsform des „Dual Licensing“[82]. Dabei erfolgt sowohl die Lizenzierung der Software unter einer Open Source Lizenz wie auch unter einem proprietären Lizenzformat. Ein bekanntes Beispiel für die Umsetzung dieses Geschäftsmodell ist die Firma MySQL AB, die das populäre Datenbanksystem *MySQL* vertreibt. Einerseits bietet sie die Software für Einzelpersonen und Unternehmen, die das System nicht in eine eigene, proprietäre Anwendung integrieren wollen, zu den Bedingungen der GPL an. Andererseits erfolgt gleichzeitig der Verkauf unter einer proprietären Lizenz an Unternehmen, die damit eine eigene proprietäre Anwendung zusammenstellen wollen.[83]

Dienstleistungen zu Open Source Software

Über den Verkauf hinaus hat sich vor allem auch das Anbieten von Dienstleistungen zu Open Source Software als Geschäftsmodell von Softwareunternehmen etabliert. Dies hat zur Folge, dass in diesen Unternehmen nicht mehr das frei verfügbare Produkt den eigentlichen Wert bildet, sondern das darüber gesammelte Wissen.[84] Dieses Wissen bieten Firmen wie Mandrakesoft für Linux, MySQL AB für das MySQL-Datenbanksystem oder Red Hat für die *JBoss Enterprise Middleware Suite (JEMS)* in

---

[82] siehe Kapitel 4.4.2 „Wirtschaftliche Rahmenbedingungen in Open Source Projekten“

[83] siehe http://www.mysql.com/company/ und http://www.mysql.com/company/legal/licensing/

[84] siehe http://www.opensource.org/advocacy/case_for_business.php

Verbindung mit verschiedenen Dienstleistungen an. Die Dienstleistungen stammen dabei im Wesentlichen aus den Bereichen *Training, Consulting* und *Support*.[85]

Training beinhaltet dabei die Schulung von Personen in den Belangen einer Software. Dazu besitzen größere Dienstleistungsunternehmen in diesem Bereich wie Red Hat oder MySQL AB in Verbindung mit anderen autorisierten Partnerunternehmen weltweit verteilte Schulungszentren, in denen Weiterbildungen angeboten werden.[86]

Consulting betrifft dagegen die Anpassung einer Software an die spezifischen Gegebenheiten der Kunden. Diese kann Aspekte wie Planung und Ausarbeitung einer Softwarestrategie, Realisierung einer Migration auf ein neues Softwaresystem, Anpassung der Software an die spezifische Infrastruktur der Kunden oder die Entwicklung zusätzlicher Funktionalitäten umfassen.

Support schließlich bezieht sich auf die langfristige Unterstützung der Kunden. Dies beinhaltet z.B. die Beantwortung technischer Fragen oder auch die Aktualisierung veralteter Versionen einer Software.

Neben der isolierten Form des Serviceanbieters werden die genannten Dienstleistungen oftmals auch in Verbindung mit dem Verkauf von Softwaredistributionen angeboten. Der Vorteil der Fokussierung auf den Bereich Open Source Software besteht für Serviceanbieter vor allem in zwei Punkten. Zum einen erleichtern die freie Verfügbarkeit des Quellcodes sowie eine aktive Open Source Community die Ausbildung des eigenen Know-hows zum Thema, zum anderen müssen für das Anbieten von Dienstleistungen zu Open Source Software keine Lizenzgebühren an die Lizenzurheber gezahlt werden.

Technische Ziele

Der dritte Bereich für Unternehmensziele in Verbindung mit einem Einsatz von Open Source Software umfasst schließlich technische Aspekte.

Der Softwareentwicklungsprozess in erfolgreichen Open Source Projekten ist geprägt durch eine hohe Anzahl von Entwicklern, die aktiv zur Verbesserung der Projektsoftware beitragen. Dies wirkt sich vor allem im Bereich der Qualitätssicherung positiv aus, da Prozesse wie Fehlersuche, Fehlerkorrektur oder Beurteilung der Benutzerschnittstellen in einem stärkeren Maße skalierbar und parallelisierbar sind [Michlmayr2004] als die Prozesse, die direkt mit der Weiterentwicklung des Quellcodes einer Software in Verbindung stehen [Brooks2000].

---

[85] siehe http://www.jboss.com/services/index und http://www.mysql.com/services/

[86] siehe http://solutions.myql.com/solutions/partners/training

Verstärkt wird dieser Effekt durch die in Open Source Projekten etablierte, sehr effektive Methode der gegenseitigen Verbesserung, das so genannte *„peer review“* [Michlmayr2004][Raymond2001][Clement2002]. Clement-Fontaine *et al.* argumentieren dazu, dass durch die Transparenz des Entwicklungsprozesses der Druck auf die beteiligten Personen steigt, qualitativ hochwertig und detailliert zu arbeiten. Das Resultat ist die Open Source Software häufig zugesprochene hohe Qualität [Wheeler2005][Raymond2001].

Diese äußert sich zumindest in erfolgreichen Open Source Projekten wie dem Linux- oder dem Apache-Projekt in einer hohen Reliabilität und Stabilität der Software [Clement2002][Mockus2002][Wheeler2005]. Die Erfüllung dieser beiden Kriterien ist für Unternehmen, die unter Umständen eine hohe Verfügbarkeit und geringe Ausfallzeiten der Software garantieren müssen, ein entscheidendes Ziel verbunden mit einem Einsatz von Open Source Software. Typische Unternehmensfelder, in denen die Reliabilität und die Stabilität einer Software eine wesentliche Rolle spielen, sind der Finanzdienstleistungssektor oder kritische Anwendungen wie beispielsweise Flugzeugsteuerungen. So kostet der Ausfall eines Bankrechenzentrums für eine Stunde das entsprechende Bankunternehmen laut einer Studie von *„Contingency Research“* etwa 2,5 Millionen U.S. Dollar [Lenz2007]. Bei einer Betriebszeit von 7 x 24 Stunden pro Woche und einer Verfügbarkeit von 99,95% bedeutet dies einen Schaden von 10,95 Millionen U.S. Dollar im Jahr. Bei einer Verfügbarkeit von 99,99% reduziert sich dieser Betrag bereits auf 2,19 Millionen U.S. Dollar.

Ein weiterer technischer Aspekt, der die Ziele eines Unternehmens im Hinblick auf die potentielle Nutzung von Open Source Produkten beeinflusst, ist die Reduzierung von Sicherheitslücken in der Software. Die Softwareentwicklung innerhalb eines offenen Entwicklungsprozesses beinhaltet ein hohes Potential zur Verringerung und Behebung sicherheitsrelevanter Programmfehler [Michlmayr2005b]. Dies hängt zum einen damit zusammen, dass es die Offenheit des Quellcodes einer Vielzahl von Entwicklern ermöglicht, Sicherheitslücken zu entdecken und gegebenenfalls zu schließen [Mitre2001]. Zum anderen argumentieren Entwickler in Open Source Projekten [Michlmayr2005b] und Anhänger der Open Source Bewegung [Raymond2001], dass mit der eigenen Zuteilung auf bestimmte Aufgabenfelder in offenen Softwareprojekten ein motiviertes und damit fokussiertes Arbeiten einhergeht, das sich ebenfalls in der Qualität der Anwendungen widerspiegelt.

Dass das erste Argument dabei allerdings auch eine Kehrseite besitzt, berichten Ebert und Ruffin [Ebert2004]. Da der Quellcode nicht nur Entwicklern offen steht, die an der Beseitigung von Sicherheitslücken interessiert sind, sondern auch denen, die versuchen, diese auszunutzen, kann der Einsatz Open Source Software vor diesem Hintergrund

auch ein Sicherheitsrisiko bedeuten. Allerdings bestätigen Ebert und Ruffin, dass sich durch offene Software zumindest die Gefahr in Verbindung mit *Trojanern*[87] reduziert, da diese leichter aufgespürt werden können. Zudem benutzen Open Source Projekte wie beispielsweise das Apache-Projekt zur Übermittlung sicherheitsrelevanter Fehler nicht das standardmäßige Bug-Tracking-System, sondern eine private Mailingliste [Halloran2002], um bestehende Sicherheitslücken nicht für jedermann kenntlich zu machen.

Ein drittes technisches Ziel von Unternehmen ist schließlich die Etablierung der mit Open Source Software häufig gleichgesetzten offenen Standards *(engl. „Open Standards“)* [Fugetta2003]. Dieses Ziel steht eng in Verbindung mit dem bereits angesprochenen Motiv der Herstellerunabhängigkeit. Offene Standards wirken dabei für Unternehmen auf zwei Arten positiv.

Erstens reduzieren sie wie erwähnt die Abhängigkeit von Anwendungs- und Softwareunternehmen zu anderen Softwareherstellern. Entspricht ein Dateiformat, eine API oder eine gesamte Softwarearchitektur einem offenen Standard, so ist es wahrscheinlich, dass er von mehreren Anbietern unterstützt wird. Dies erhöht die Konkurrenzsituation und ermöglicht es Kunden, gegebenenfalls den Softwarelieferanten zu wechseln. Zudem sichert ein offener Standard damit in gewissem Maße auch die Langlebigkeit einer Softwarelösung, da kein einzelner Hersteller den Lebenszyklus der Software beenden kann [Ebert2004].

Zweitens tragen offene Standards zur Gewährleistung der *Interoperabilität* [88] verschiedener Softwaresysteme bei [Fugetta2003]. Dies gilt insofern, da standardisierte Schnittstellen die Grundlage für eine reibungslose Zusammenarbeit unterschiedlicher Softwareanwendungen bilden. Für Anwendungsunternehmen bedeutet dies, dass die Integration unterschiedlicher Softwareprodukte unterschiedlicher Softwarehersteller möglich wird. Für Softwareunternehmen stellt es einen Mehrwert des eigenen Produktes dar. Anzumerken bleibt in diesem Zusammenhang schließlich, dass offene Standards nicht mit Open Source Software gleichzusetzen sind. So kann auch proprietäre Software offen standardisierte Dateiformate, APIs oder andere, offen standardisierte Komponenten implementieren [Fugetta2003].

---

[87] Ein Trojaner ist ein Programm, das als nützlich getarnt ist, im Hintergrund aber eine versteckte Funktion ausführt

[88] In Verbindung mit Softwaresystemen bezeichnet der Begriff „Interoperabilität“ die Fähigkeit zur Zusammenarbeit unterschiedlicher Anwendungen

4.4.6 Anforderungen der Unternehmen an Open Source Projekte

Verbunden mit ihren Zielen besitzen unterschiedliche Unternehmen auch unterschiedliche Anforderungen an Open Source Projekte und ihre Software. Um diese zu ermitteln, wird im Rahmen der Arbeit eine Unternehmensumfrage zu den Risiken und Vorteilen eines Einsatzes von Open Source Software durchgeführt. Die im Zuge dessen identifizierten relevanten Kriterien von Unternehmen an freie Softwareprojekte werden dann in einem zweiten Schritt zu Anforderungskategorien zusammengefasst, die die Grundlage für die spätere Zerlegung des Evaluationsgegenstandes „Open Source Projekt“ bilden.[89]

Ziele der Umfrage

Die Durchführung der Umfrage verfolgt vor allem zwei Ziele. Zum einen soll sie die Risiken und Vorteile, die Unternehmen mit einem Einsatz von Open Source Software verbinden, ermitteln. Die Abfrage erfolgt dabei offen, d.h. ohne eine Vorgabe von Antwortmöglichkeiten, um zu gewährleisten, dass die Antworten aus den Überlegungen der verantwortlichen Personen der Unternehmen hervorgehen und somit explorativen Charakter besitzen. Zudem sollen die genannten Aspekte hinsichtlich ihrer Wichtigkeit von den Betrieben und Organisationen bewertet werden. Die dadurch ermittelten unternehmensrelevanten Kriterien können dann in einem zweiten Schritt zu Anforderungskategorien zusammengefasst werden, die als *„Zielfaktoren“* in die Zerlegung des Evaluationsgegenstandes innerhalb des GQM-Verfahrens in der Planungsphase eingehen.[87]

Zum anderen sollen gleichzeitig die unterschiedlichen Bedingungen, die in den Unternehmen vorherrschen und die Einschätzungen der verantwortlichen Personen im Hinblick auf einen Einsatz von Open Source Software beeinflussen können, beleuchtet werden. Dies dient der Kontrolle, ob bestimmte unternehmensinterne Rahmenbedingungen mit spezifischen Einstellungen zu Open Source Projekten verbunden sind. Die dabei abgefragten Kontrollbedingungen sind die Unternehmensbranche, die Anzahl der Mitarbeiter, die Art des Einsatzes von Open Source Software und der Grad der Involvierung des Unternehmens in bestehende Open Source Projekte.

Aufbau des Fragebogens

Entsprechend der Unterscheidung von Anwendungsunternehmen, die Open Source Software als reine Anwendung nutzen, und Softwareunternehmen, die sie innerhalb

[89] siehe Kapitel 5 „Planung der Evaluation“

ihres Softwareentwicklungsprozesses einsetzen,[90] existieren auch zwei voneinander abweichende Fassungen des Fragebogens der Umfrage. Dieser Vorgehensweise liegt die Annahme zugrunde, dass mit den zwei unterschiedlichen Anwendungsgebieten auch auseinander gehende Anforderungen an Open Source Projekte und ihre Software verbunden sind.

Während im Fall der reinen Anwendungssoftware vor allem Aspekte wie geringe Migrationskosten, standardisierte Formate und dauerhafter Support im Vordergrund stehen sollten, birgt der Einsatz von Open Source Software in der Softwareentwicklung zusätzliche rechtliche Schwierigkeiten wie versteckte Lizenzbedingungen oder zu erbringende Garantieleistungen auf die eigene Software.

Die daraus resultierenden Abweichungen zwischen den zwei Versionen des Fragebogens sind allerdings nur geringfügig. Der grundlegende Aufbau bleibt identisch. Beide bestehen aus vier inhaltlichen sowie einem zusätzlichen Abschnitt.

Allgemeiner betrieblicher Kontext

Der erste Abschnitt besteht lediglich aus zwei Fragen, die zur Kontrolle der nachfolgenden Erhebung der Risiken und Vorteile von Open Source Software den allgemeinen betrieblichen Kontext des Unternehmens verdeutlichen sollen. Dies beinhaltet zum einen die Frage nach der Branche, in der das jeweilige Unternehmen tätig ist, zum anderen nach der Anzahl der dort beschäftigten Mitarbeiter. Für die letzte Frage sind die vier Antwortmöglichkeiten 1-10, 11-100, 101-1000 und >1000 vorgegeben.

Art des Einsatzes von Open Source Software

Neben der Größe eines Unternehmens und der Branche, in der es tätig ist, bestimmen vor allem auch die Art des Einsatzes und der Grad der Verwendung die betriebliche Wahrnehmung des Phänomens „Open Source Software“. Beide Aspekte bilden zudem selbst einen wesentlichen Bestandteil der gegenwärtigen Situation in den Unternehmen im Hinblick auf Open Source Software.

Folglich zielt die Untersuchung der genannten Punkte zum einen auf die Gewinnung einer weiteren Kontrollgröße, zum anderen soll sie ein Spiegelbild der aktuellen Einsatzgebiete von Open Source Software liefern. Konzeptionell ist dies dabei der Bereich, in dem sich die zwei Versionen des Fragebogens grundsätzlich unterscheiden. Während im Fall der Anwendungsunternehmen eine inhaltliche Unterscheidung zwischen bestimmten Typen von Anwendungen wie Betriebssystemen oder Datenbanken vorgenommen wird, erfolgt für Softwareunternehmen eine eher

[90] siehe Kapitel 4.1 „Beteiligte Gruppen“

strukturelle Zerlegung in Komponenten, die innerhalb des Softwareentwicklungszyklus zur Bildung einer Gesamtanwendung verwendet werden.[91] Beispiele dafür sind Softwarebibliotheken sowie Programmierframeworks.

In beiden Fällen vollzieht sich die Abfrage der Items bewertet, d.h. es wird nicht bloß nach den Einsatzgebieten von Open Source Software selbst gefragt, sondern auch nach der Intensität der Nutzung im jeweiligen Bereich. Dazu ist eine fünfstufige Skala mit den Elementen *„wird nicht eingesetzt“*, *„ein Einsatz ist zukünftig geplant“*, *„wird in Testsystemen eingesetzt“*, *„wird eingeschränkt eingesetzt“* und *„wird signifikant eingesetzt“* vorgegeben.

Schließlich erhalten Softwareunternehmen eine zusätzliche Frage zu ihrer Involvierung in bestehende Open Source Projekte, um diesbezüglich bestehende Zusammenhänge erkennen zu können.

Risiken und Vorteile von Open Source Software

Abschnitt drei und vier sind die Kernabschnitte des Fragebogens, in denen die unternehmensrelevanten Risiken und Vorteile eines Einsatzes von Open Source Software ermittelt werden sollen. Die Befragung erfolgt dabei offen, d.h. ohne eine Vorgabe wählbarer Antwortmöglichkeiten. Dies soll gewährleisten, dass die Antworten den Überlegungen der befragten Personen entspringen und keine Widerspiegelung der Vorstellungen des Autors sind. Unterteilt sind die Abschnitte dabei in die Kategorien *rechtliche, technische, wirtschaftliche* und *sonstige Risiken* bzw. *Vorteile* [Ebert2004], um dennoch eine strukturierte Form der Abfrage zu gewährleisten.

Zusätzlich zur Nennung sollen die Unternehmen die von ihnen angeführten Risiken und Vorteile darüber hinaus in ihrer Stärke bewerten. Dazu ist ebenfalls eine fünfstufige Skala vorgegeben, die von *„sehr gering“* bis *„sehr hoch“* reicht.

Als abschließende inhaltliche Frage werden die Unternehmen schließlich gebeten, konkrete Informationen anzugeben, die sie sich zur Beseitigung ihrer Bedenken gegenüber Open Source Projekten wünschen.

Zusatzinformationen

Die letzten beiden Fragen sind nicht mehr inhaltlicher Natur. Sie dienen lediglich dazu, Anmerkungen zur Form der Datenerhebung sowie eine Email-Adresse bei Interesse an den Umfrageergebnissen zu hinterlassen.[92]

---

[91] Die Komponenten könnten zusätzlich inhaltlich aufgespaltet werden

[92] Für den Fragenkatalog, siehe Anhang B.1

Durchführung der Umfrage

Insgesamt wurden Fragebögen an 605 unterschiedliche deutsche Unternehmen verschickt, darunter 290 Anwendungs- und 315 Softwareunternehmen. Die Benachrichtigung der Firmen erfolgte dabei auf zwei Wegen.

Die erste Gruppe, bestehend aus insgesamt 132 ausgewählten Unternehmen, erhielt ein schriftliches Anschreiben per Post. Der Brief enthielt dabei neben der Umfrage an sich und einem frankierten Rückumschlag auch einen Verweis auf eine angefertigte Online-Umfrage, sodass den verantwortlichen Personen zwei Alternativen zur Beantwortung des Fragebogens zur Verfügung standen. Hintergrund dieser Vorgehensweise war die Hoffnung auf eine höhere Rücklaufquote im Vergleich zu einem direkten Anschreiben per Email. Unter den 132 Unternehmen dieser Gruppe waren 69 Softwarefirmen, darunter die 25 wirtschaftskräftigsten in Deutschland aus dem Jahr 2004[93], und 63 Anwendungsunternehmen, die sich hauptsächlich aus Banken, Behörden, Bibliotheken und Verlagen aus dem Raum Berlin zusammensetzten.

Die zweite Gruppe aus den 473 übrigen Firmen wurde per Email von der Umfrage benachrichtigt. Sie bestand aus 246 Software- sowie 227 Anwendungsunternehmen, deren Kontaktadressen mit Hilfe des Internetportals *www.webadress.de* zur Suche von gewerblichen Email-Adressen zusammengestellt wurden. Dabei wurde versucht, eine möglichst breite Verteilung der Unternehmen auf unterschiedliche Branchen zu gewährleisten.

Die Online-Umfrage war insgesamt drei Wochen lang geöffnet, eingehende Briefe wurden auch nach dieser Frist noch akzeptiert.

Ergebnisse

Aus den 605 insgesamt verschickten Anschreiben resultierten 26 gültig beantwortete Fragebögen. Dies entspricht einer Rücklaufquote von etwa 4%. Unter den betreffenden Unternehmen befanden sich 8 Anwendungs- sowie 18 Softwareunternehmen.

Unternehmensgröße

Die Resultate der Umfrage im Hinblick auf die Größe der Unternehmen, gemessen an der Zahl ihrer Mitarbeiter, unterscheiden sich für die beiden identifizierten Unternehmenskategorien. Von den 18 Softwareunternehmen besitzen neun eine Mitarbeiterzahl zwischen 1 und 10, zwei verfügen über 11-100, sechs über 101-1000 und lediglich ein Unternehmen über mehr als 1000 Angestellte. Demgegenüber liegen die Angestelltenzahlen der meisten Anwendungsunternehmen im Bereich von 11-100 bzw. von über 1000 Mitarbeitern (siehe Tab. 3).

---

[93] siehe http://luenendonk.de/download/press/LUE_SSU_2005_f230505.pdf

Tabelle 3: Anzahl der Mitarbeiter der Unternehmen

	Anz. der Softwareunternehmen	Anz. der Anwendungsunternehmen
1-10	9	1
11-100	2	3
101-1000	6	1
Über 1000	1	3

Unternehmensbranche

Die zweite Kontrollgröße der Umfrage ist die Unternehmensbranche. Diese soll es ermöglichen, gegebenenfalls branchenspezifische Risiken und Vorteile eines Einsatzes von Open Source Software in Unternehmen extrahieren und unterscheiden zu können.

Entsprechend den Ergebnissen lässt sich die Gruppe der Softwareunternehmen in die Branchen *reine Softwareentwicklung*, *Soft- und Hardwareentwicklung*, *Consulting* und *Forschung* unterteilen. Zusätzlich sind ein *Musikverlag* sowie eine *Versicherung* unter den Unternehmen dieser Kategorie. Den größten Teil der Antworten machen dabei mit über 50% die reinen Softwareentwicklungsfirmen aus, wobei dieser Bereich inhaltlich weiter unterteilt werden kann (siehe Tab. 4). Weiterhin sind zwei Unternehmen in der Forschung sowie im Bereich Consulting tätig, während auf die verbleibenden genannten Branchen jeweils nur eine beantwortete Umfrage fällt.

Aus den Resultaten der Gruppe der Anwendungsunternehmen ergeben sich analog fünf Kategorien von betrieblichen Tätigkeitsfeldern. Dies sind die Bereiche *Öffentliche Behörden/Verwaltung*, *Archive*, *Banken*, *Museen* und *Softwareentwicklung*. Der letzte Punkt erscheint dabei kurios innerhalb der Gruppe der Anwendungsunternehmen und resultiert daraus, dass im Zuge der Durchführung der Umfrage auch Dienstleistungsfirmen im Bereich Datenbanken als Softwareanwender angeschrieben wurden, die darüber hinaus möglicherweise eigene Software entwickeln. Der Großteil der beteiligten Unternehmen stammt mit drei bzw. zwei Vertretern allerdings aus den Feldern Öffentliche Behörden und Archive.

Tabelle 4: Branchen der Unternehmen

	Anz. der Softwareunternehmen	Anz. der Anwendungsunternehmen
Softwareentwicklung • Nicht weiter spezifiziert • Automotive SW • Graphische Datenverarbeitung • SW für Vermieter • Individualsoftware	 7 1 1 1 1	1
Soft- und Hardwareentwicklung	1	---
Consulting	2	---
Forschung	2	---
Versicherungen	1	---
Musikverlage	1	---
Öffentliche Behörden/ Verwaltung	---	3
Archive	---	2
Banken	---	1
Museen	---	1

Art des Einsatzes von Open Source Software in Anwendungsunternehmen

Bevor sowohl die Anwendungs- als auch die Softwareunternehmen zu den internen Einsatzgebieten von Open Source Software befragt wurden, sollten sie zunächst angeben, ob sie offene Software im Rahmen ihrer Arbeitsprozesse überhaupt nutzen.

Dies ergab für die beteiligten Anwendungsunternehmen eine Einsatzquote von 100%, d.h. alle acht Organisationen und Betriebe, die die Umfrage beantworteten, bestätigten die Nutzung von Open Source Software. Weiterhin waren zur genauen Klärung der Einsatzgebiete einerseits ausgewählte Kategorien von Softwaretypen vorgegeben, zu denen die Unternehmen sich äußern sollten. Andererseits sollten sie darüber hinaus zusätzliche Anwendungen und Programme spezifizieren, die sie im Rahmen ihrer betrieblichen Arbeitsprozesse verwenden.

Die Resultate beider Punkte ergaben gemeinsam die in Tabelle 24[94] dargestellten Bereiche für einen Einsatz von Open Source Software in Anwendungsunternehmen. Die

[94] siehe Anhang B.2

Durchschnittswerte für die Intensität des Gebrauchs des jeweiligen Softwaretyps beziehen sich dabei nur auf Unternehmen, die angaben, entsprechende Anwendungen auch einzusetzen. Dies bedeutet, die Werte sind nicht gleichzusetzen mit den Durchschnittswerten aller beteiligten Unternehmen.

Für den Bereich der Betriebssysteme gaben fünf von acht Anwendungsunternehmen an, ein Open Source Betriebssystem einzusetzen. Zwar erfolgt der Einsatz im Durchschnitt lediglich eingeschränkt, allerdings ist eine Tendenz in Richtung einer signifikanten Verwendung gegeben. Dieser Trend ist offensichtlich eng verbunden mit dem Aufschwung des Open Source Betriebssystems Linux in den letzten Jahren. So stieg in japanischen Unternehmen von 2001 auf 2002 der Verbreitungsgrad des Betriebssystems um mehr als 28% [Wheeler2005] und eine *„Zona Research"-Studie* belegt darüber hinaus, dass 63% der von ihnen befragten Unternehmen eine 10-50%ige Steigerung der Linux-Nutzung im eigenen Betrieb prognostizieren [Wheeler2005]. Weitere Hinweise für einen solchen Aufschwung auch in Deutschland liefern deutsche Behörden. Neben der Stadt München[95], die ihre 14.000 Arbeitsplatzrechner auf Linux umgestellt hat, hat vor allem auch das Bundesland Niedersachsen mit seiner flächendeckenden Ausrüstung der Polizei[96] mit dem Betriebssystem gezeigt, dass der Rückgriff auf Open Source Software in diesem Bereich mehr als nur eine Alternative ist. Dass die Nutzung von Open Source Betriebssystemen in Unternehmen trotzdem nur eingeschränkt erfolgt, hängt damit zusammen, dass eine Migration auf Open Source Software zum einen mit einem erheblichen Aufwand zu Beginn verbunden und zum anderen auch nicht in jedem Kontext die kostengünstigere Lösung ist.[97]

Ein zweites wesentliches Feld für Open Source Software ist der Bereich der *Webserver*. Hierbei geben sieben von acht Unternehmen an, quelloffene Software zu verwenden. Der Einsatz erfolgt dabei im Gegensatz zu den Betriebssystemen sogar durchschnittlich signifikant. Dies ist jedoch nicht überraschend, da mit dem Apache Webserver das marktführende Produkt aus dem Bereich Open Source Software stammt. Vor allem in Verbindung mit dem Linux-Betriebssystem, der MySQL-Datenbank und den Scriptsprachen *Perl* und *PHP* bildet der Server eine populäre Infrastruktur zur Entwicklung von Webanwendungen.

Ähnliche Resultate liefert die Umfrage für den Bereich der *Applikationsserver*. Dabei geben sechs der acht beteiligten Unternehmen an, in diesem Anwendungsfeld Open Source Software zu verwenden. Analog zur Kategorie der Webserver erfolgt auch im Fall der Applikationsserver die Nutzung im Durchschnitt signifikant. Allerdings

---

[95] siehe http://www.heise.de/newsticker/meldung/37197

[96] siehe http://www.heise.de/newsticker/meldung/40374

[97] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen"

entstammt der alleinige Marktführer in diesem Bereich nicht einem Open Source Projekt. Zwar existiert mit dem JBoss Applikationsserver ein Open Source Produkt, das große Marktanteile besitzt, allerdings teilt es sich diese mit Unternehmen wie IBM, BEA, Oracle und SUN.[98]

Ein weiterer interessanter Aspekt ist die Tatsache, dass die Ergebnisse der Umfrage eine Benutzung von freien *Office-Anwendungen* nur in Testsystemen oder eingeschränkt indizieren. Dies hängt vermutlich mit der Dominanz der Firma Microsoft in diesem Feld zusammen. Da die genauen Dokumentformate Microsofts keine offenen Standards sind, sondern gehütete Betriebsgeheimnisse, wird eine spätere Migration auf andere, möglicherweise offene Produkte erschwert, da eine Konvertierung vorhandener Dokumente eventuell nicht ohne weiteres möglich ist. Dies ist gleichzeitig ein Mittel vieler Softwarehersteller, um Kundenabhängigkeiten zu schaffen.[99]

Schließlich fallen unter den von Unternehmen zusätzlich aufgeführten Einsatzgebieten von Open Source Software unter anderem die Bereiche Entwicklungsumgebungen und Content Management Systeme mit drei bzw. zwei signifikant nutzenden Betrieben auf. Für den Fall der Entwicklungsumgebungen stammt dabei laut einer Studie von QA Systems mit der Java-Plattform Eclipse die in Betrieben am weitesten verbreitete Anwendung aus dem Bereich Open Source Software [QASystems2003]. Dies ist allerdings insofern nicht verwunderlich, als dass Eclipse eine Gemeinschaftsproduktion der Firmen IBM, Borland, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain ist.[100]

Im Bereich der Content Management Systeme bzw. speziell der *Web Content Management Systeme (WCMS)*[101] existiert dagegen ein breites Spektrum an Lösungen sowohl durch proprietäre als auch durch offene Softwaresysteme. Während große Unternehmen im Bereich des Enterprise Content Management (ECMS)[102] wie Open Text, Oracle oder IBM zunehmend Anbieter von WCMS-Anwendungen aufkaufen, um der steigenden Nachfrage zu begegnen, bewegen sich vermehrt auch Hersteller von Open Source Anwendungen in diesem Feld [Gartner2007]. Open Source Projekte wie *Typo3*, *eZ publish*, *OpenCMS*, *Magnolia* oder *Plone* sind nur einige Beispiele für den

---

[98] für Stand Dezember 2005: siehe http://rmh.blogs.com/weblog/2006/05/bz_research_on_.html

[99] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

[100] siehe http://www.eclipse.org/org/index.php

[101] Ein „Web Content Management System“ bezeichnet ein Softwaresystem zur Verwaltung von Webinhalten

[102] Ein „Enterprise Content Management System“ bezeichnet den Zusammenschluss unterschiedlicher Formen der Inhaltsverwaltung in einem Softwaresystem. Web Content Management ist folglich ein Teilaspekt des Enterprise Content Managements.

steigenden Druck auf etablierte Anbieter proprietärer Web Content Management Systeme vor allem aus dem Bereich der offenen Softwareentwicklung.

Art des Einsatzes von Open Source Software in Softwareunternehmen

Analog zur Befragung der Anwendungsunternehmen wurde auch Softwareunternehmen zunächst die Frage gestellt, ob sie offene Software überhaupt verwenden. Im Gegensatz zum ersten Fall bezog sich der Einsatz von Open Source Software dabei allerdings nicht auf eine Nutzung in Form einer Anwendungssoftware, sondern auf eine Verwendung von offenen Softwarekomponenten innerhalb des eigenen Entwicklungsprozesses. Einen solchen Einsatz bestätigten 13 von 18 Unternehmen.

Auffällig ist dabei vor allem die starke Verbreitung von Softwarebibliotheken, Frameworks und kompletten Open Source Anwendungen zur Ausformung der Infrastruktur der Unternehmenssoftware. Alle drei Einsatzbereiche sind in jeweils über 50% der beteiligten Unternehmen etabliert. Während Softwarebibliotheken und Frameworks dabei lediglich eingeschränkt eingesetzt werden, erfolgt die Nutzung vollständiger Open Source Anwendungen wie z.B. Datenbanken sogar durchschnittlich signifikant. In Verbindung mit den Umfrageresultaten der Anwendungsunternehmen kann dies als ein weiterer Indikator für die Etablierung von Open Source Software im Bereich der Infrastruktur gedeutet werden und demonstriert darüber hinaus, dass offene Softwarebibliotheken bereits für ein breites Spektrum von Anwendungsfeldern verfügbar sind.[103]

Zusätzlich bildet die stark eingeschränkte Nutzung von *Open Source Codefragmenten* und *Open Source Codegeneratoren* einen interessanten Aspekt der Umfrageergebnisse. Die Einschränkungen in der Verwendung der beiden Komponenten stehen dabei vermutlich unter anderem im Zusammenhang mit rechtlichen Implikationen, die sich aus einem Einsatz ergeben können. Diese entstehen unter Umständen in Verbindung mit Copyleft-Klauseln der entsprechenden Open Source Lizenz. Eingefügter Quellcode aus Open Source Software, die unter einer solchen Lizenz verbreitet wird, kann dazu führen, dass die eigene Anwendung des Unternehmens ebenfalls unter die Open Source Lizenz gestellt und frei verfügbar gemacht werden muss.[104] Ähnlich verhält es sich mit der Nutzung von Quellcode, der einem Open Source Codegenerator entstammt, wobei hier nur in Sonderfällen mit rechtlichen Problemen dieser Art zu rechnen ist. In der Regel verpflichtet der Einsatz von offenen Codegeneratoren nicht zur Freigabe des entstehenden Codes. Ausnahmen von dieser Regel entstehen dann, wenn mit dem generierten Programmcode gleichzeitig Teile des Quellcodes des Generators kopiert

---

[103] siehe z.B. Xerces, eine Bibliothek zum Parsen von XML-Daten

[104] siehe Kapitel 4.3.5.1 „Copyleft-Lizenzen“

werden. In diesem Fall entspricht das Resultat rechtlich betrachtet einer Bearbeitung der Open Source Anwendung und muss im Zuge einer Copyleft-Bestimmung ebenfalls freigegeben werden. Ein Beispiel dafür bildete bis zur Version 1.24 der Parser *„Bison“*, der vollständig der GPL unterstellt war und Teile seines eigenen Programmcodes mit in den Objektcode des entstehenden Programms kompilierte [Jaeger2002].

Einen Überblick über die Umfrageresultate zum Einsatz von Open Source Komponenten innerhalb des Entwicklungszyklus von Softwareunternehmen liefert Tabelle 25.[105]

Kontakt zu den Open Source Projekten

Zusätzlich zur Art des Einsatzes von Open Source Software wurden Softwareunternehmen auch zu ihrem Kontakt zu den dazugehörigen Projekten befragt. Die Ergebnisse zeigen, dass nur etwa ein Drittel aller beteiligten Softwareunternehmen überhaupt keinen Kontakt zu den Open Source Projekten besitzt. Über 50% aller Beteiligten und damit über 70% der Betriebe, die Open Source Software wirklich einsetzen, nutzen die Foren und Mailinglisten der Projekte, um Hilfestellungen zu eigenen Problemen zu erhalten. Dies zeigt, dass die Mailinglisten, sofern keine Dienstleistungsunternehmen mit den Entwicklungen im Projekt in Verbindung stehen, die wesentlichsten Supportschnittstellen bilden.

Immerhin 7 der 13 Softwareunternehmen, die offene Software im Rahmen ihres eigenen Entwicklungszyklus verwenden, geben darüber hinaus an, entdeckte Fehler an die Projekte weiterzuleiten und tragen damit indirekt zu einem funktionierenden Qualitätssicherungsprozess bei. Die Anzahl der Unternehmen, die aktiv in die Gestaltung der Prozesse sowie die Weiterentwicklung der Projektsoftware involviert sind, ist dagegen relativ gering. So sind lediglich drei Betriebe auch als Antwortgeber in den Mailinglisten aktiv und sogar nur zwei Unternehmen stellen den Projekten Patches für bestehende Fehler zur Verfügung. Zwei Betriebe sind darüber hinaus aktiv mit der Weiterentwicklung von Open Source Software beschäftigt und ein Unternehmen beteiligt sich an der Führung eines Open Source Projektes.

[105] siehe Anhang B.3

Tabelle 5: Kontakt der Softwareunternehmen zu Open Source Projekten

Art des Kontaktes	Anz. der Softwareunternehmen
Wir haben keinen Kontakt zu den Projekten	6
Wir nutzen die Foren und Mailinglisten der Projekte, um Antworten auf sich uns stellende Probleme zu erhalten	10
Wir sind in Foren und Mailinglisten auch als Antwortgeber aktiv	3
Von uns entdeckte Fehler werden an die Projekte weitergeleitet	7
Unsere Entwickler lösen bei uns auftretende Probleme und stellen ihre Lösungen als Patches zur Verfügung	2
Wir sind aktiv an der Beseitigung von Fehlern beteiligt, die bisher nicht bei uns aufgetreten sind	---
Wir sind aktiv an der Weiterentwicklung (und nicht ausschließlich der Fehlerbeseitigung) der Software beteiligt	2
Unsere Entwickler helfen mit administrativen Aufgaben, wie der Veröffentlichung von neuen Versionen und der Wartung der Website	---
Unsere Entwickler sind aktiv an der Führung des Projektes beteiligt	1
Wir beobachten den OSS-Markt genau	1

Anforderungen an Open Source Projekte

Aus den Umfrageresultaten zu den Risiken und Vorteilen, die Unternehmen mit einem Einsatz von Open Source Software verbinden, lassen sich für Anwendungsunternehmen vor allem fünf Kategorien von Anforderungen extrahieren. Diese entstehen durch die Vereinigung der von den Unternehmen genannten Items zu Gruppen, die diese semantisch zusammenfassen. Die fünf Kategorien sind *Zukunftssicherheit*, *Softwarequalität*, *Kostenkontrolle*, *Planbarkeit* und *Rechtssicherheit*. Für Softwareunternehmen ergibt sich zusätzlich die Forderung nach der *Nutzbarkeit für die Erstellung eigener Produkte*. Dabei ist zu beachten, dass die Zusammensetzungen der Anforderungskategorien für Anwendungs- und Softwareunternehmen teilweise variieren, sodass zwar die resultierenden Kategorien größtenteils übereinstimmen, die einzelnen Items aber durchaus auseinander gehen können.

Eine der wesentlichen Anforderungen von Unternehmen an Open Source Projekte ist ihre Zukunftssicherheit. Dies hängt damit zusammen, dass eine Migration auf Open Source Software zunächst mit einem Mehraufwand an Schulungs- und

Migrationskosten verbunden ist, der sich häufig erst langfristig amortisiert.[106] Zukunftssicherheit ist dabei sowohl für Anwendungs- als auch für Softwareunternehmen geknüpft an das Risiko der Einstellung eines Open Source Projektes. Dieses Risiko wird von etwa 25% der Softwareunternehmen sowie einem Anwendungsunternehmen genannt und durchschnittlich als mittelmäßig bis hoch eingestuft. Zudem spielt mangelhafter bzw. fehlender Support eine wesentliche Rolle für die Zukunftssicherheit einer Investition in Open Source Software. Erstaunlicherweise sind es in diesem Bereich gerade die Softwareunternehmen, die diese Gefahr als hoch bis sehr hoch einschätzen. Anwendungsunternehmen dagegen bewerten das Risiko als eher gering. Dies kann damit zusammenhängen, dass sich im Umfeld der meisten großen Open Source Projekte wie Linux[107] oder dem JBoss Application Server[108] zahlreiche Dienstleistungsunternehmen etabliert haben, die Anwendersupport zur Software anbieten, so dass die Anwenderseite im Hinblick auf den Support bei diesen Projekten in der Regel gut abgedeckt ist. Auf der Gegenseite zu den Risiken eines Einsatzes von Open Source Software verbunden mit dem Aspekt der Zukunftssicherheit nennen die Unternehmen in diesem Zusammenhang auch charakteristische Vorteile des offenen Entwicklungsprozesses in Open Source Projekten. Diese beeinflussen zwar nicht direkt die Zukunftssicherheit der einzelnen Projekte, sehr wohl aber die Zukunftssicherheit der Investitionen der beteiligten Unternehmen. Die dabei sowohl von Anwendungs- als auch von Softwareunternehmen angeführten Vorteile sind die Offenheit des Quellcodes, die damit verbundene Anpassbarkeit der Software sowie die Etablierung von offenen Standards. Alle drei genannten Kriterien tragen dazu bei, die Fortentwicklung der Software auch im Fall einer Einstellung des Projektes entweder durch eigene Entwicklungsarbeit oder durch die Beauftragung externer Unternehmen abzusichern. Darüber hinaus nennen Anwendungsunternehmen zusätzlich den Vorteil eines breiten Angebots an Lösungen, das vor allem in Verbindung mit der angesprochenen Standardisierung von Formaten gegebenenfalls eine Migration auf andere offene Softwaresysteme ermöglicht, ohne bereits erstellte Dateien und Dokumente aufwändig konvertieren zu müssen.

Die zweite Anforderung von Unternehmen an Open Source Projekte ist die Softwarequalität. Dieser Aspekt ist vor allem vor dem Hintergrund, dass mit der freien Nutzung von Open Source Software keine Garantien auf die Qualität einer Anwendung einhergehen, ein wichtiges Kriterium und wird von den Unternehmen kontrovers wahrgenommen. Einerseits werden Risiken wie leichter identifizierbare

---

[106] siehe 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

[107] siehe z.B. http://www.mandriva.com

[108] siehe http://www.jboss.com/partners/partners_syst_int

Sicherheitslücken, schlechtes Design oder mangelhafte Dokumentation mit den Entwicklungen in Open Source Projekten in Verbindung gebracht. Andererseits sehen Unternehmen in der offenen Softwareentwicklung gleichzeitig auch Vorteile, die auch die genannten Aspekte positiv beeinflussen. So ermöglicht die Offenheit des Quellcodes eine schnellere Korrektur bestehender Sicherheitsmängel. Darüber hinaus fördert eine große Community die Erstellung einer guten Dokumentation und trägt gleichzeitig zu einer besseren Wartung und damit verbunden zu einer hohen Qualität der Software bei. Auffällig in Verbindung mit der Anforderung der Softwarequalität sind vor allem drei Punkte.

Erstens werden wie erwähnt die Qualität der Softwaredokumentation und die Sicherheit der Anwendung sowohl als Risiko wie auch als Vorteil eines Einsatzes von Open Source Software von Unternehmen genannt. Dies verdeutlicht die große Heterogenität im Feld der Open Source Projekte.

Zweitens sind die von den Softwareunternehmen angeführten Kriterien wie erwartet stärker technisch geprägt als die der Anwendungsunternehmen. Während für Softwareunternehmen Aspekte wie mangelhaftes Design, aufgeblähter Programmcode oder eine hohe Testabdeckung relevant sind, stehen für Anwendungsunternehmen eher funktionale Aspekte wie Inkompatibilität mit bestehender Software oder begrenzte Anwenderfreundlichkeit im Vordergrund.

Drittens schließlich werden im Hinblick auf die Softwarequalität sowohl von den Anwendungs- als auch von den Softwareunternehmen die Vorteile durchschnittlich höher bewertet als die Risiken. Während die assoziierten Risiken dabei im Durchschnitt als mittelmäßig eingeschätzt werden, bewerten Anwendungsunternehmen den positiven Einfluss eines Einsatzes von Open Source Software auf die Softwarequalität hoch, Softwareunternehmen sogar hoch bis sehr hoch. Dies zeigt, dass Open Source Software von Unternehmen trotz der damit verbundenen Risiken häufig als qualitativ hochwertig angesehen wird.

Die nächste aus den Antworten der Unternehmensbefragung extrahierte Anforderung an Open Source Projekte ist die Kostenkontrolle bzw. Kostenreduktion. In diesem Bereich bietet Open Source Software vor allem zwei wesentliche Vorteile, die sowohl von Anwendungs- als auch von Softwareunternehmen genannt werden. Zum einen ist die Nutzung von offener Software in der Regel nicht an die Entrichtung einer Lizenzgebühr geknüpft[109], zum anderen bietet die Offenheit des Quellcodes der Software auch insofern einen Investitionsschutz, als dass selbst im Fall einer Einstellung des Projektes

[109] Eine Ausnahme bietet das Modell des „Dual Licensing“, siehe Kapitel 4.4.2 „Wirtschaftliche Rahmenbedingungen in Open Source Projekten“

oder der Ausmusterung veralteter Versionen eine Anpassung und Wartung der jeweiligen Anwendung selbstständig von den Unternehmen initiiert werden kann. Zusätzlich ergibt sich für Softwareunternehmen der Vorteil der Abgabe eines Teils der Entwicklungsarbeit sowie der aufwändigen Pflege [Balzert2000] der offenen Softwarekomponenten.

Demgegenüber stehen allerdings auch bestimmte Risiken mit einem Einsatz von Open Source Software in Verbindung, die zu einem erhöhten Kostenaufwand führen können. So nennen Anwendungsunternehmen hohe Anpassungskosten bzw. erhöhte Supportkosten durch den Einsatz eigener Mitarbeiter als zu berücksichtigende Faktoren im Zuge einer Betrachtung der Total Cost of Ownership. Zusätzlich erwähnen Softwareunternehmen den hohen Aufwand, um geeignete Bibliotheken zu finden. Dieser Aspekt ist vor allem vor dem Hintergrund wesentlich, dass mit Open Source Software in der Regel keine Garantien auf die Funktionsweise einhergehen, sodass zunächst Zeit und Geld investiert werden müssen, um unterschiedliche Anwendungen hinsichtlich ihrer Brauchbarkeit zu evaluieren. Zusammenfassend lässt sich zur Anforderung der Kostenkontrolle bzw. Kostenreduktion sagen, dass für eine genaue Einschätzung der finanziellen Vor- und Nachteile eines Einsatzes von Open Source Software die individuellen Gegebenheiten in den Unternehmen einbezogen werden müssen.[110] Auffällig im Hinblick auf die Umfrageresultate ist jedoch erneut, dass sowohl Software- wie auch Anwendungsunternehmen die Vorteile höher einschätzen als die korrespondierenden Risiken.

Die vierte Anforderung von Unternehmen an Open Source Projekte ist die Planbarkeit des Entwicklungsprozesses. Der Begriff „Planbarkeit“ bezieht sich dabei nicht auf die im Zusammenhang mit dem Faktor „Zukunftssicherheit“ genannten existenziellen Aspekte, sondern auf das Potential, den zeitlichen und inhaltlichen Ablauf der Entwicklungen in Open Source Projekten abschätzen zu können, unter der Voraussetzung, dass Zukunftssicherheit gegeben ist. Das Risiko, das sowohl Software- als auch Anwendungsunternehmen in diesem Zusammenhang identifizieren, ist die teilweise nicht vorhandene klare *Roadmap*[111] der Projekte. Dies beinhaltet einerseits die fehlende Angabe der zukünftig geplanten Features der Software sowie andererseits die fehlende zeitliche Planung des Entwicklungsverlaufs. Darüber hinaus erwähnen Anwendungsunternehmen die starke Abhängigkeit in der Qualität und der Wartung von den jeweiligen Open Source Projekten und die Schwierigkeit, dass Wartungsverträge nicht direkt mit dem Hersteller der Software abgeschlossen werden können, als zusätzliche Hürden für die Planbarkeit eines Einsatzes von Open Source Software.

---

[110] siehe 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

[111] Eine „Roadmap“ beschreibt einen Plan zu zukünftigen Features einer Software

Softwareunternehmen sehen dagegen in der nicht garantierten Lösung und der geringen Versionskontrolle weitere Schwierigkeiten.

Auf der anderen Seite bringt der offene Entwicklungsprozess in Open Source Projekten auch Stärken mit, die zu einer besseren Planbarkeit der Entwicklungen beitragen. So ermöglichen die Offenheit und die damit verbundene Transparenz des Entwicklungsprozesses eine bessere Nachvollziehbarkeit der Aktivitäten zur Erstellung der Projektsoftware. Dies schließt unter anderem auch die Abschätzung der Größe der Projektcommunity und der Aktivität ihrer Mitglieder über offen zugängliche Datenquellen wie die Mailinglisten oder das Versionsverwaltungssystem mit ein. Zudem beschleunigt eine große und aktive Projektcommunity den Entwicklungs-, Wartungs- und Pflegeprozess der Software und trägt damit zu einer Verkürzung der Releasezyklen bei.

Auffällig an den Umfrageresultaten, die den Faktor „Planbarkeit“ ausmachen, ist einerseits, dass Softwareunternehmen sowohl die Risiken wie auch die Vorteile eines Einsatzes von Open Source Software in Verbindung mit dem Aspekt der Planbarkeit durchschnittlich etwas höher bewerten als Anwendungsunternehmen. Andererseits werden die Risiken und Vorteile dabei im Gegensatz zu den Faktoren Softwarequalität und Kostenkontrolle als etwa genauso hoch eingestuft. Dies verdeutlicht, dass der Faktor „Planbarkeit“ im Zusammenhang mit Open Source Projekten ein Bereich ist, in dem Open Source Projekte auch relevante Schwächen offenbaren. Das wesentliche Problem für Unternehmen besteht dabei in der fehlenden Sicherheit in Bezug auf die Entwicklungsrichtung sowie die Releasetermine.

Der Aspekt der Sicherheit führt auch zur fünften Anforderung von Unternehmen an Open Source Projekte – der Rechtssicherheit. In diesem Zusammenhang ist es zunächst einmal die Haftungsfrage, die sowohl Anwendungs- als auch Softwareunternehmen beschäftigt. Für Anwendungsunternehmen bedeutet dies die Frage, welche Unternehmen oder andere rechtliche Instanzen sind bei groben Mängeln der Software für daraus entstehende zusätzliche Kosten haftbar zu machen? Open Source Lizenzen enthalten einen Haftungsausschluss, sodass in der Regel nicht die Projekte selbst für bestehende Fehler der Software zur Verantwortung gezogen werden können. Hier bietet es sich an, die Software von bestehenden Distributoren im Umfeld der Projekte zu kaufen, die eigene, zertifizierte Versionen vertreiben. Für Softwareunternehmen im Gegenzug beinhaltet die Nutzung von Open Source Software für die Erstellung eigener Produkte das Risiko, selbst für bestehende Fehler in den offenen Komponenten der resultierenden Gesamtanwendung haftbar gemacht zu werden. Da offene Software wie erwähnt ohne eine Gewährleistung auf ihre Qualität verbreitet wird, die Unternehmen selbst aber sehr wohl für ihre Produkte zur Verantwortung gezogen werden können, müssen sie im Zuge dessen auch die Haftung für die offenen Bestandteile der

Anwendung übernehmen. Auch in diesem Fall bietet sich die Nutzung von Softwaredistributoren an.

Der zweite wesentliche Punkt in Verbindung mit dem Aspekt der Rechtssicherheit ist die Berücksichtigung patentrechtlicher Fragen. Dies umfasst einerseits die mögliche Legalisierung von Softwarepatenten in der EU, andererseits auch die potentielle Verletzung bereits bestehender Patente. Für Unternehmen bedeutet der Einsatz von Open Source Software in diesem Zusammenhang das Risiko, für verletzte Patentrechtsansprüche durch die freien Anwendungen im Nachhinein Gebühren entrichten zu müssen. Einige Lizenzen wie die Mozilla Public License gewähren die Nutzungsrechte an von der Software berührten Patenten nur für unveränderte Versionen einer Anwendung. Wird das Programm im Zuge der Weiterverbreitung modifiziert, so verliert der Nutzer unter Umständen auch das Recht, die betroffenen Patente zu nutzen [Jaeger2002]. Insbesondere vor dem Hintergrund, dass solche Vergehen durch die Offenheit des Programmquellcodes leichter nachzuweisen sind, sollten diesbezüglich bestehende Bedenken im Vorfeld eines Einsatzes von Open Source Software geklärt werden. Darüber hinaus könnte eine Legalisierung von Softwarepatenten eine generelle Gefährdung für die offene Softwarebewegung an sich darstellen. Drittparteien würde infolge dessen die Möglichkeit eingeräumt werden, Patente auf fremd entwickelte Open Source Software anzumelden und damit die mit ihr einhergehenden Freiheiten zu limitieren, sodass als Resultat eine zu proprietärer Software vergleichbare Situation entstehen würde.

Zusätzlich zu den beiden genannten Aspekten der Rechtssicherheit erwähnen Anwendungsunternehmen noch das ungeklärte Vertragsverhältnis an sich und Softwareunternehmen die Schwierigkeiten in Verbindung mit der kommerziellen Nutzung von Open Source Software unter bestimmten Lizenzen als kritische Punkte eines Einsatzes offener Software. Während der erste Punkt vermutlich eng mit dem zuvor diskutierten Aspekt der unklaren Haftungsverhältnisse zusammenhängt, bildet der zweite eine neue Kategorie von Risiken. So verpflichten bestimmte Open Source Lizenzen den Nutzer dazu, Veränderungen an der Software sowie Kombinationen mit anderen Programmen wieder zu den Bedingungen der Lizenz zu verbreiten.[112] Dies bedeutet für Softwareunternehmen, die eine eigene, proprietäre Anwendung gemeinsam mit Open Source Software vertreiben, dass sie gegebenenfalls gezwungen werden, auch den Quellcode der eigenen Bestandteile einer resultierenden Gesamtanwendung zu veröffentlichen. Dies kann unter Umständen sogar das gesamte Geschäftsmodell eines Softwareherstellers gefährden.

---

[112] siehe Kapitel 4.3.5.1 „Copyleft“-Lizenzen

Auf der Seite der Vorteile eines Einsatzes von Open Source Software im Hinblick auf den Faktor „Rechtssicherheit“ werden von den Unternehmen lediglich zwei wesentliche Aspekte genannt. Dies sind zum einen sowohl für Software- als auch für Anwendungsunternehmen die meist günstigen Lizenzbedingungen in Bezug auf den Ersterwerb bzw. die resultierende freie Nutzung des Quellcodes der Software. Zum anderen bietet Open Source Software Softwareunternehmen die Möglichkeit eines neuen Geschäftsmodells.[113] Dabei geben sie die Software frei ab und generieren die Unternehmenseinkünfte anschließend aus einem umfangreichen Angebot an Dienstleistungen zur Anwendung. Der rechtliche Vorteil besteht dabei darin, dass im Zuge der Lizenzierung der Software unter einer Open Source Lizenz die Haftung auf das erstellte Produkt entfällt oder zumindest stark eingeschränkt wird.

Zusammenfassend lässt sich zur Anforderung der Rechtssicherheit sagen, dass Unternehmen in diesem Punkt tendenziell mehr Risiken als Vorteile sehen. Dies hängt sicherlich mit den teilweise unklaren Haftungsbedingungen und der fehlenden vertraglichen Fixierung der Rechtslage zusammen. Eine Lösung für Risiken dieser Art bieten Unternehmen und Distributoren, die sich im Umfeld von Open Source Projekten etablieren und Dienstleistungen zur Software anbieten.

Die letzte Anforderung schließlich, die Softwareunternehmen zusätzlich an einen Einsatz von Open Source Software knüpfen, ist die Nutzbarkeit der Software für die Entwicklung eigener Anwendungen. Dies umfasst auf der einen Seite Kriterien zur Beschleunigung des eigenen Entwicklungsprozesses sowie auf der anderen Seite rechtliche und marketingtechnische Einschränkungen.

Zum ersten Aspekt nennen die beteiligten Softwareunternehmen unter anderem den Aspekt der Wiederverwendung. Wiederverwendung bedeutet dabei, dass bereits existierende Komponenten, die eine bestimmte Funktion realisieren, nicht neu implementiert, sondern in bestehende Anwendungen integriert werden, sodass ein Teil der Entwicklungsarbeit gespart werden kann. Dazu ist es jedoch erforderlich, dass eine solche Integration möglichst leicht vollzogen werden kann. In diesem Zusammenhang erwähnen Softwareunternehmen zwei wesentliche Vorteile eines Einsatzes von Open Source Software. Dies ist erstens der freie Zugang zu bestehenden Schnittstellen offener Software, zweitens erleichtert die Etablierung offener Standards die Gewährleistung der Interoperabilität unterschiedlicher Softwaresysteme. Hinzu kommt, dass die Offenheit des Quellcodes es Unternehmen bei Vorhandensein des entsprechenden Know-hows ermöglicht, die jeweiligen Schnittstellen anzupassen. Neben dem Aspekt der Wiederverwendung bietet Open Source Software einen reichhaltigen Fundus an

[113] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

vorhandenem Wissen. Dies bestätigen auch die Antworten der beteiligten Softwareunternehmen. Dabei werden der Aufbau auf vorhandenem Wissen sowie die gegenseitige Ideenbefruchtung als wesentliche Stärken im Zusammenhang mit Open Source Software genannt. Während der erste Punkt erneut in die Richtung des genannten Aspekts der Wiederverwendung tendiert, beschreibt der zweite das Vorhandensein eines Innovationspotentials für Softwareunternehmen.

Tabelle 6: Anforderungen der Anwendungsunternehmen an einen Einsatz von OSS

Anforderung	Assoziiertes Risiko	Assoziierter Vorteil
Zukunftssicherheit	• Einstellung des Projektes • Kein Support	• Breites Angebot an Lösungen • Offenheit • Anpassbarkeit • Standardisierung
Softwarequalität	• Sicherheitslücken • Mangelhafte Dokumentation • Heterogenität in unterschiedlichen Bereichen einer Software • Inkompatibilität mit bestehender Software • Begrenzte Anwenderfreundlichkeit	• Standardisierung • Sicherheit • Anpassbarkeit • schnelles Bugfixing
Kostenkontrolle	• Hohe Anpassungskosten • Hohe Supportkosten durch Einsatz eigener Mitarbeiter	• Keine Lizenzgebühren • Geringere Kosten • Investitionsschutz
Planbarkeit	• Keine klare Roadmap • Qualität und Wartung sind abhängig vom Projekt • Keine Wartungsverträge direkt mit dem Hersteller möglich	• Schnelles Bugfixing • Bessere Planung von Releasezyklen • Offenheit
Rechtssicherheit	• Vertragsverhältnis • Gewährleistung und Haftung • Patentrecht • Rechte Dritter • OS-Lizenzen, die man kennen muss • (Offenlegung des eigenen Quellcodes bei Verwendung der LGPL) • Keine Wartungsverträge direkt mit dem Hersteller möglich	• Lizenzbedingungen sind meist günstig • Einige Lizenzen werden von der OSI überwacht

Den beschriebenen Vorteilen eines Einsatzes von Open Source Software, die die Softwareunternehmen als hoch bis sehr hoch einstufen, stehen allerdings auch Schwierigkeiten gegenüber, die einer Kombination von offener und Unternehmenssoftware im Wege stehen. Diese Schwierigkeiten bestehen in den Lizenzbedingungen einiger Open Source Lizenzen, die einerseits den Vertrieb einer Kombination von offener und Unternehmenssoftware an sich, als auch andererseits ihre Vermarktung erschweren oder sogar verhindern können. So verlangt die GPL beispielsweise im Zuge einer gemeinsamen Verbreitung von GPL-Software und Unternehmenssoftware, dass in der Regel auch die von den Unternehmen entwickelten Bestandteile offen gelegt werden müssen. Andere Lizenzen wie die Originalfassung der BSD Copyright-Lizenz erschweren dagegen die Vermarktung solcher Kombinationen, da sie die Nutzer der Software verpflichten, in allen Werbematerialien stets auch einen Verweis auf den Urheber der freien Software zu veröffentlichen.

Darüber hinaus ist die aufkommende Konkurrenzsituation zu Open Source Projekten für Softwareunternehmen ein weiterer wesentlicher Aspekt im Zusammenhang mit der Vermarktung eigener Software. Im Zuge dessen werden die Betriebe unter Druck gesetzt, im Fall einer Überschneidung des eigenen Portfolios und des Softwareangebots offener Softwareprojekte mit der Qualität und vor allem den Preisen von Open Source Software zu konkurrieren. Basiert das Geschäftsmodell der Unternehmen dabei auf der Generierung von Lizenzgebühren und besitzt die jeweilige Anwendung kein Alleinstellungsmerkmal, das es von den offenen Programmen in diesem Bereich abhebt, so kann dies unter Umständen schwierig sein.

Tabelle 6 und 7 zeigen einen Überblick über die aus den Umfrageergebnissen extrahierten Anforderungen von Unternehmen an Open Source Projekte und ihre Software. Eine Übersicht über alle genannten Risiken und Vorteile sowie die Bewertung der einzelnen Items befindet sich im Anhang.[114]

[114] siehe Anhang B.4 und B.5

Tabelle 7: Anforderungen der Softwareunternehmen an einen Einsatz von OSS

Anforderung	Assoziiertes Risiko	Assoziierter Vorteil
Zukunftssicherheit	• Einstellung des Projektes • Mangelhafte Weiterentwicklung/ Pflege des Projektes • Kein/mangelhafter Support	• Offenheit • Anpassbarkeit • Große Entwicklergemeinde • Standardisierung
Softwarequalität	• Mangelhafte(s) Architektur/Design • Aufgeblähter Programmcode • Sicherheitslücken/Virengefahr • Mangelhafte Dokumentation • Mangelhaftes Fehlerhandling	• Standardisierung • Qualität der Software • Sicherheit • Bessere Wartung • Hohe Testabdeckung • Oftmals sehr gute Dokumentation
Kostenkontrolle	• Kosten durch Updates • Hohe Total Cost of Ownership (TCO) • Hoher Aufwand, um geeignete Bibliotheken zu finden	• Aufbau auf vorhandenem Wissen • Keine Lizenzgebühren • Abgabe der Pflege und Weiterentwicklung • Geringerer Eigenentwicklungsaufwand • Kostenersparnis • Keine Knebelverträge • Investitionsschutz
Planbarkeit	• Keine klare Roadmap • Keine Garantie der Lösung • Geringe Versionskontrolle	• Hohe Entwicklungsgeschwindigkeit • Offenheit • Transparenz
Rechtssicherheit	• Legalisierung von Softwarepatenten • Andere patentrechtliche Fragen • Haftungsrisiko • Einsatz von OS-Lizenzen für eine kommerzielle Nutzung • Urheberrechtsansprüche • Inkompatibiltät der Lizenzen	• Keine eigene Haftung bei der Herstellung von OSS => Verschiebung Richtung Support (2) • Freie Nutzung des Quellcodes
Nutzbarkeit für die Entwicklung eigener Produkte	• Probleme in der kommerziellen Vermarktung der eigenen Software bei der Integration von OS-Bestandteilen • Konkurrenz für Lizenzprodukte • Einsatz von OS-Lizenzen für eine kommerzielle Nutzung	• Flexible Nutzung von Bibliotheken/ Frameworks • Zugang zu Schnittstellen • Wiederverwendung • Offenheit • Ideenbefruchtung • Aufbau auf vorhandenem Wissen • Standardisierung • Anpassbarkeit

Grenzen und Schwächen der Umfrage

Die größte Schwäche der durchgeführten Unternehmensbefragung ist der offensichtlich sehr geringe Rücklauf von lediglich 26 beantworteten Fragebögen. Dies ist gleichzeitig auch der Grund, dass entgegen der ursprünglichen Zielstellung auf eine detaillierte Analyse der Beziehungen zwischen den eingeführten Kontrollbedingungen wie der Unternehmensgröße und den extrahierten Anforderungen verzichtet wurde. Die Größe der Stichprobe ermöglicht keine Formulierung statistisch valider Korrelationen. Darüber hinaus ergeben sich weitere Schwierigkeiten.

Erstens beinhalten sowohl das Anschreiben per Email wie auch per Brief das Problem, dass damit nicht immer die Entscheidungsträger bzw. die kompetenten Personen eines Unternehmens in Bezug auf den Einsatz von Open Source Software erreicht werden. Vor allem in großen Unternehmen kann dies häufiger der Fall sein und erklärt sicherlich teilweise den geringen Rücklauf der Umfrage.

Zweitens besteht die Tendenz, dass eher Unternehmen, die Open Source Software bereits einsetzen, dazu geneigt sind, Fragen zu diesem Thema zu beantworten. Die Tatsache, dass ein Großteil der Antwortgeber bereits Open Source Software verwendet, zeigt einerseits, dass die Nutzung offener Anwendungen in Unternehmen schon etabliert ist, andererseits lässt es auch die Vermutung zu, dass Betriebe, die aus unterschiedlichen Gründen nicht auf Open Source Software zurückgreifen, stärker dazu neigen, Fragen zu diesem Thema nicht zu beantworten. Im Zuge dessen könnte eine Verzerrung der Ergebnisse entstehen, die die überwiegend positive Sichtweise des Phänomens „Open Source Software“ relativieren würde.

Drittens schließlich stellt sich die Frage, inwieweit die Auswahl der Unternehmen die Umfrageresultate beeinflusst. Im Rahmen der Befragung wurden ausschließlich in Deutschland ansässige Unternehmen angeschrieben. Dies beinhaltet natürlich, dass die Resultate ein Spiegelbild der deutschen Wahrnehmung der Open Source Softwarebewegung ist, das zum einen durch kulturelle, zum anderen aber auch durch rechtliche Besonderheiten im Vergleich zu anderen Nationen geprägt ist. Darüber hinaus wurde allerdings versucht, eine möglichst breite Verteilung der Unternehmen auf unterschiedliche Branchen zu gewährleisten, um verschiedene branchenspezifische Unterschiede in der Wahrnehmung zu erfassen.

Trotz der aufgezählten Schwächen und Grenzen der Umfrage ist davon auszugehen, dass das wesentliche Ziel, die Ermittlung unternehmensrelevanter Anforderungen an Open Source Projekte, erreicht wurde. Die identifizierten Risiken und Vorteile eines Einsatzes von Open Source Software mögen unter Berücksichtigung der angeführten Umfragegrenzen unvollständig, aber darüber hinaus nicht falsch sein. Vor dem Hintergrund, dass die Umfrageresultate im Rahmen der Arbeit zum einen zur

allgemeinen Analyse des betrieblichen Umfeldes von Open Source Projekten im Rahmen der Phase der Definition des Evaluationsproblems sowie zum anderen zur Identifizierung relevanter Zielfaktoren[115] innerhalb der Phase der Evaluationsplanung herangezogen werden, bildet keine der beschriebenen Umfragegrenzen eine Grundlage zur Annahme, dass die Resultate dies nicht ermöglichen. Bestehende Tendenzen zur Handhabung des Phänomens „Open Source Software“ in Unternehmen wurden aufgezeigt sowie wichtige Zielfaktoren in Form relevanter Anforderungen extrahiert.

[115] siehe Kapitel 5 „Planung der Evaluation“

5 Planung der Evaluation

Nachdem im ersten inhaltlichen Schritt eine detaillierte Analyse zu Open Source Projekten und ihrer Umgebung durchgeführt wurde, erfolgt in der Planungsphase nun die Zerlegung des Evaluationsgegenstandes in Teilkomponenten. Das Ziel besteht dabei darin, abstrakte Zielfaktoren, wie z.B. Zukunftssicherheit, hohe Qualität oder Plattformunabhängigkeit, auf messbare Metriken, wie z.B. Anzahl der aktiven Entwickler innerhalb der letzten 6 Monate oder mittlere Zeit zur Behebung eines Bugs, herunterzubrechen. Diese Metriken sollen dann eine Evaluation des Untersuchungsgegenstandes „Open Source Projekt" ermöglichen. Für die strukturierte Zerlegung des Evaluationsgegenstandes im Rahmen dieser Arbeit wurde der *„Goal-Question-Metric"-Ansatz (GQM-Ansatz)* [Basili1994], ein iteratives *Top-Down-Verfahren*[116] zur Analyse und Bewertung von Produkten, Prozessen und Ressourcen, gewählt, welcher im folgenden erläutert wird.

5.1 Der „Goal-Question-Metric"-Ansatz (GQM)

Der „Goal-Question-Metric"-Ansatz ist ein iteratives Top-Down-Verfahren. Dies bezieht sich vor allem auf zwei Aspekte.

Erstens erfolgt nach der vollständigen Zerlegung des Untersuchungsgegenstandes eine Überprüfung der erzielten Messergebnisse. Relativieren diese entweder die angenommenen Beziehungen zwischen den definierten Zielen und den identifizierten Teilaspekten oder die Relationen zwischen den Teilaspekten und den zugrunde liegenden Metriken, so wird die Vorgehensweise in einem zweiten Iterationsschritt vollständig wiederholt.

Zweitens basiert die Methodik des Verfahrens auf der Zerlegung eines Problems in Teilprobleme. Dies bedeutet im Gegensatz zu einem *Bottom-Up-Verfahren*, bei dem einfache Probleme nach und nach zu komplexeren zusammengesetzt werden, dass die Grundlage des Verfahrens das identifizierte Ziel ist und nicht die zugrunde liegende Metrik. Dies ist für die Anwendung des Verfahrens auf Open Source Projekte insofern entscheidend, da eine Zerlegung der Zielfaktoren in Teilaspekte zu Beginn der Evaluation nicht bekannt ist. Ein Nachteil des Bottom-Up-Verfahrens ist also, dass die vorhandenen Metriken zu noch unbekannten Teilproblemen zusammengesetzt werden

[116] Die Bezeichnung „Top-Down" bezeichnet im Gegensatz zu „Bottom-Up" eine Vorgehensweise zur Bearbeitung eines Problems, die auf der höchsten Ebene des Problems beginnt und dieses durch Zerlegung in Teilprobleme löst

müssten. Das Top-Down-Verfahren hat aber in umgekehrter Richtung häufig das Problem, dass bei der Zerlegung eines Zielfaktors Teilaspekte entstehen, für die keine brauchbaren Metriken gefunden werden können.

Konzeptuelle Ebene

Inhaltlich setzt sich das GQM-Verfahren aus drei Ebenen zusammen. Die konzeptuelle Ebene als oberste Schicht beinhaltet die formulierten Ziele zum Untersuchungsgegenstand. Diese werden im Rahmen der Arbeit als „Zielfaktoren“ bezeichnet.

Definition: *Der Begriff „Zielfaktor“ bezeichnet die Ziele einer Untersuchung innerhalb der konzeptuellen Ebene des GQM-Verfahrens.*

Ein Zielfaktor besitzt dabei vier Dimensionen, die seine Ausprägung charakterisieren. Dies sind der Untersuchungsgegenstand selbst, der dabei betrachtete Aspekt, die Perspektive des Betrachters und die Absicht der Untersuchung.

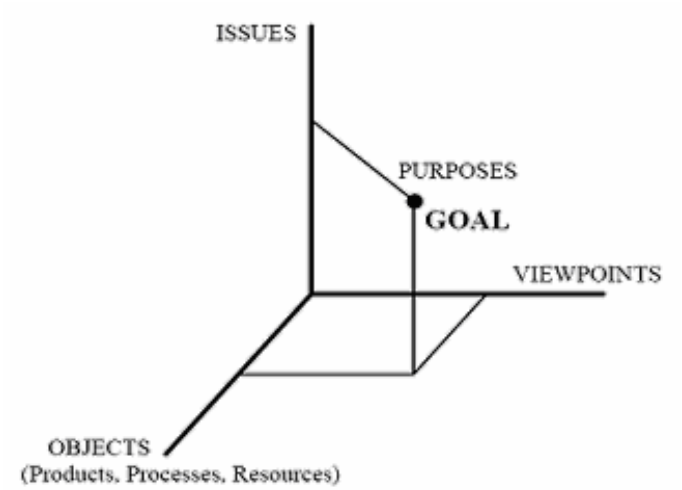

Abbildung 9: Dimensionen eines Zielfaktors innerhalb des GQM-Verfahrens

Operationale Ebene

Die operationale Ebene im zweiten Schritt besteht aus spezifischen Fragestellungen zu den zuvor ermittelten Zielfaktoren. Die Fragen beziehen sich dabei inhaltlich auf den Untersuchungsgegenstand allgemein sowie auf spezielle Eigenschaften, die den zuvor ausgewählten Aspekt betreffen. Darüber hinaus sind Fragen zur Evaluation dieser Eigenschaften zu erheben.

Quantitative Ebene

Im dritten Schritt, der quantitativen Ebene, werden schließlich Metriken zur Beantwortung der aufgestellten Fragestellungen entwickelt. Dies bedeutet, dass zu jeder

formulierten Frage ein Datensatz assoziiert werden muss. Verdeutlicht wird der allgemeine Aufbau eines GQM-Modells noch einmal in Abbildung 10.

5.2 Übertragung des Ansatzes auf Open Source Projekte

Um nun eine Evaluation des Untersuchungsgegenstandes „Open Source Projekt“ mit Hilfe des GQM-Verfahrens realisieren zu können, gilt es, eine entsprechende Zerlegung von Open Source Projekten in die drei Ebenen des Modells zu finden, die die Strukturen und Prozesse in den Projekten widerspiegelt. Cruz *et al.* schlagen dazu eine mögliche Zerlegung vor [Cruz2006].

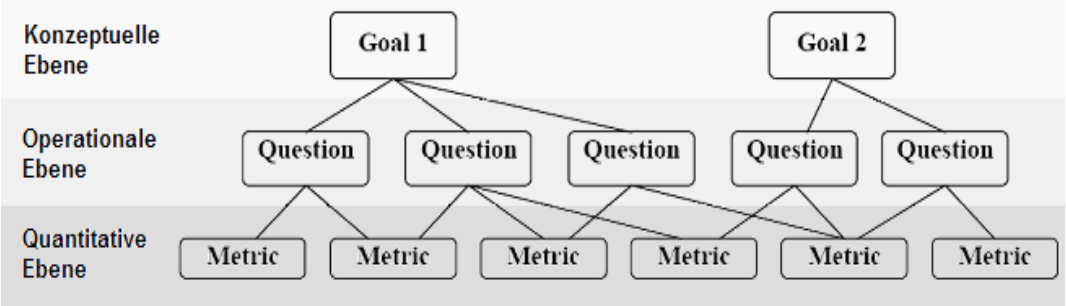

Abbildung 10: Allgemeiner Aufbau eines GQM-Modells

5.2.1 Zerlegung nach Cruz *et al.*

Allgemeine Vorgehensweise

Cruz *et al.* versuchen im Zuge ihrer Untersuchungen des Phänomens „Open Source Software“, einen Weg zur eigenständigen Evaluation sowohl der Software als vor allem auch der dazugehörigen Projekte vorzuzeichnen [Cruz2006]. Dazu nehmen sie allgemein die Perspektive von Unternehmen und speziell die ihrer Manager ein. Ausgehend von unterschiedlichen Nutzungsszenarien für Open Source Software in Organisationen identifizieren Cruz *et al.* spezifische Anforderungen, die im Kontext eines betrieblichen Einsatzes der Software entstehen. Diese kategorisieren sie in die Bereiche *Funktionale*, *Technische*, *Organisatorische*, *Rechtliche*, *Wirtschaftliche* und *Politische Anforderungen*. Parallel dazu entwickeln Cruz *et al.* Kriterien zu Open Source Projekten, die mit den entsprechenden Bedürfnissen der Unternehmen in Verbindung stehen, sowie darüber hinaus Metriken zu deren Messung. In einem letzten Schritt diskutieren sie dann die Zusammenhänge zwischen der Ausprägung der ermittelten Kriterien und der Erfüllung der Anforderungen der Unternehmen (siehe Abb. 11).

Übertragung in ein GQM-Modell

Um die vorgezeichnete Zerlegung des Untersuchungsgegenstandes „Open Source Projekt“ auf das GQM-Verfahren zu übertragen, gilt es, eine Zuordnung zwischen den von Cruz *et al.* identifizierten Untersuchungsinstanzen und den Analyseebenen eines GQM-Modells (siehe Abb. 10) zu finden.

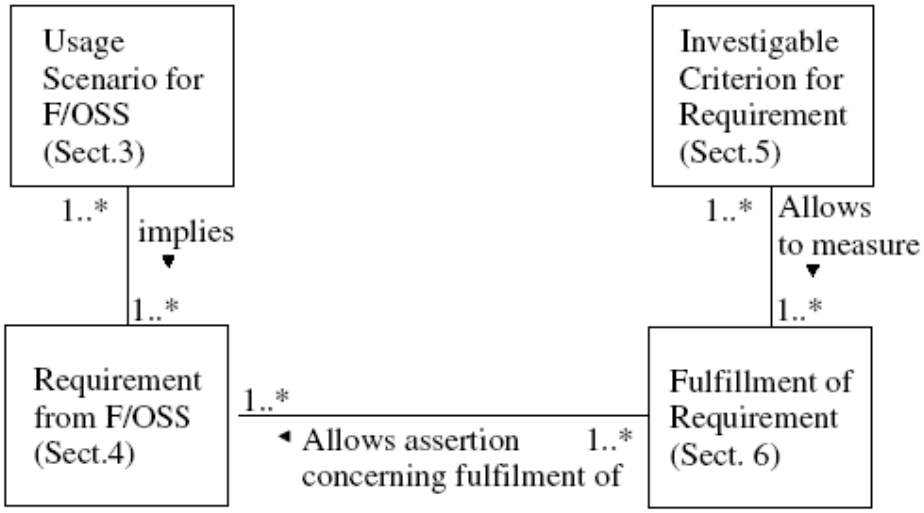

Abbildung 11: Allgemeine Vorgehensweise von Cruz *et al.*

Konzeptuelle Ebene (Zielfaktoren)

Im ersten Schritt erfolgt hierzu das Ausfüllen der konzeptuellen Ebene. Der Perspektive der Evaluation folgend entsprechen die Zielfaktoren innerhalb des GQM-Verfahrens den Anforderungen der Unternehmen nach Cruz *et al.* Dabei ist zu beachten, dass damit nicht die allgemeinen Anforderungskategorien gemeint sind, sondern die konkret formulierten Items.[117]

Operationale Ebene (Fragen)

Der zweite Schritt besteht dann in der Aufteilung der ermittelten Zielfaktoren in charakteristische Fragestellungen, die diese möglichst genau umschreiben. Dies bedeutet für die Anpassung der Ergebnisse von Cruz *et al.* an das GQM-Modell, dass die Resultate der Sektionen 5 und 6 (siehe Abb. 11) transformiert werden müssen. In diesen beiden Sektionen entwickeln Cruz *et al.* zunächst Kriterien, die Open Source Projekte hinsichtlich der sechs genannten inhaltlichen Kategorien beschreiben, und versuchen dann, diese im Hinblick auf die Erfüllung der Unternehmensanforderungen

[117] Einen Überblick über die Anforderungen der Unternehmen nach Cruz *et al.* liefert Anhang C.1

zu interpretieren. Übertragen auf die Operationale Ebene des GQM-Modells folgt daraus zum einen, dass die Kriterien in entsprechende Fragestellungen umgewandelt werden müssen. Zum anderen müssen sie angelehnt an die Interpretationen den jeweiligen Zielen zugeordnet werden.[118]

Tabelle 8: GQM-Zerlegung des Faktors „Reduzierung der Kosten" nach Cruz *et al.*

Goal	
Absicht	Reduzierung
Betrachteter Aspekt	der Kosten
Objekt	durch den Einsatz von OS-Software
Perspektive	aus der Sicht von Unternehmensmanagern

Question	**Metrics**
Q1 Welche Funktionalitäten realisiert die SW?	**M1.1** Angegebene Funktionalitäten auf Featurelisten
Q2 Welchen Aufwand bringt eine Migration auf die Software mit sich?	**M2.1** Eine Kosten-Nutzen-Analyse zur Schätzung der benötigten Mannkraft, des erforderlichen Know-hows, der Zeit und des finanziellen Aufwandes einer Migration sowie des erwarteten Nutzens
Q3 Mit welchen monatlichen Kosten ist zu rechnen?	**M3.1** Eine Analyse der laufenden Kosten sowie der Kosten zur In-House-Pflege der Software
Q4 Existiert Dokumentation zur Software?	**M4.1** Existierende Links auf der Projekthomepage
Q5 Von wie vielen anderen Softwarepaketen hängt das Funktionieren der SW ab?	**M5.1** Angaben in den „Readme"-Dateien
Q6 Herrscht ein konstruktives, fortschrittsorientiertes Klima in den Foren?	**M6.1** Betrachtung der Diskussionen in den Mailinglisten

Quantitative Ebene (Metriken)

Schließlich müssen die entwickelten Fragestellungen in letzter Instanz messbar gemacht werden. Dazu definieren Cruz *et al.* bereits im Zuge der Kriterienentwicklung Metriken

[118] Ein Überblick über die von Cruz *et al.* entwickelten Kriterien sowie die dazugehörigen Fragestellungen befindet sich im Anhang C.2

zu deren Erfassung. Diese können zur Komplettierung des GQM-Modells direkt übernommen werden und bilden die Quantitative Ebene.[119]

Zur Verdeutlichung der Zerlegung von Open Source Projekten nach Cruz *et al.* und der Übertragung in ein GQM-Modell ist in Tabelle 8 beispielhaft die Analyse des Faktors *„Reduzierung der Unternehmenskosten“* dargestellt.

5.2.2 Probleme

Der von Cruz *et al.* aufgezeigte Weg zur Evaluation von Open Source Projekten und die daraus resultierende Zerlegung des Untersuchungsgegenstandes erfassen zwar viele wesentliche Aspekte der Prozesse in Open Source Projekten, allerdings bringt ihr Ansatz auch einige Schwächen mit sich.

Der erste Schwachpunkt ist dabei die fehlende Eingrenzung des Anwendungsgebietes der Software. Während unternehmensseitig eine Unterscheidung der verschiedenen Unternehmensanforderungen in Form der angeführten Nutzungsszenarien erfolgt, existiert keine vergleichbare inhaltliche Fokussierung für die Projekte selbst. Delacretaz [Stuermer2005], Kernentwickler des *„Cocoon“-Projektes*, Michlmayr und Senyard [Michlmayr2004] sowie O’Mahony und West [O’Mahony2005] liefern Hinweise, dass in der offenen Entwicklungsphase verschiedener Open Source Projekte Unterschiede bestehen, die einige der von Cruz *et al.* angeführten Metriken beeinflussen, aber gleichzeitig nicht auf die von ihnen entwickelten Projektkriterien zurückzuführen sind, sondern auf grundsätzliche Differenzen in der Anlage der Projekte. Beispiele für solche grundlegenden Unterschiede sind

- die Art der Anwendung (Delacretaz, Michlmayr und Senyard)
- oder verschiedene Formen der Projektgründung (O’Mahony und West).

Im Kontext anderer Open Source Projekte kann also eine Metrik völlig andere Bedeutung bekommen. Vergleicht man z.B. eine Endbenutzersoftware mit einer Softwarebibliothek, so muss die Bedeutung der Anzahl der Downloads bei der Bibliothek deutlich höher bewertet werden.

In diesem Zusammenhang wird auch die zweite Schwäche des Evaluationskonzeptes von Cruz *et al.* deutlich – die fehlende Validierung. Zwar ist die Herleitung der angeführten Anforderungen, Kriterien und Metriken schlüssig, allerdings fehlt ein empirischer Beleg der Aussagen und Implikationen, sodass ihre Generalisierbarkeit vor allem vor dem Hintergrund der zuvor erwähnten individuellen Gegebenheiten in Open Source Projekten angezweifelt werden muss.

---

[119] Einen Überblick über die von Cruz *et al.* entwickelten Metriken liefert Anhang C.2

Die letzte Schwachstelle der Untersuchungen von Cruz *et al.* bezieht sich auf die Durchführung des vorgezeichneten Evaluationsweges. Zwar wird die Erfassung der entwickelten Metriken von Cruz *et al.* beschrieben, allerdings ist diese Beschreibung teilweise zwangsläufig sehr allgemein gehalten, so dass sie für Personen, die nicht mit den Strukturen in Open Source Projekten vertraut sind, eventuell nur schwer nachvollziehbar ist. Darüber hinaus fehlen Vergleichswerte, um erzielte Resultate einordnen zu können.

5.3 Erweiterung des Ansatzes von Cruz *et al.*

Diese Arbeit soll aufgrund der skizzierten Probleme eine Stärkung der Vorgehensweise von Cruz *et al.* in folgenden Bereichen sein:

1) Exemplarische Fokussierung auf ein Anwendungsfeld (Web Content Management Systeme) und einen plausiblen dazugehörigen Zielfaktor (Zukunftssicherheit),
2) Durchführung einer Evaluation bis auf die konkrete Ebene der Messung der einzelnen Metriken und
3) Konkretisierung ausgewählter Metriken im Bezug auf den gegebenen Kontext.

5.3.1 Der Zielfaktor „Zukunftssicherheit“

Der für die im Rahmen der Arbeit beispielhaft durchgeführten Teilevaluation von Open Source Projekten ausgewählte Zielfaktor ist „*Zukunftssicherheit*“. Dabei beschreibt Zukunftssicherheit die Aussicht auf eine langfristige Etablierung eines Open Source Projektes und seiner Software.

Definition: *Zukunftssicherheit bezeichnet die Wahrscheinlichkeit, mit der ein Open Source Projekt langfristig besteht und seine Software weiterentwickelt wird.*

Gründe für den Zielfaktor „Zukunftssicherheit“

Die Wichtigkeit des Faktors „Zukunftssicherheit“ spiegelt sich dabei in mehreren Argumenten wider.

Zu allererst ist es gerade dieses Kriterium, das eines der großen Fragezeichen in Verbindung mit Open Source Projekten bildet. Die Mehrzahl der freien Softwareprojekte besteht lediglich aus einer kleinen Gruppe von hauptverantwortlichen Entwicklern, die den Großteil des Entwicklungsaufwandes leisten müssen.[120] Dies

[120] siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

bedeutet für diese Projekte eine starke Abhängigkeit von einem kleinen Personenkreis. Besonders in Verbindung mit dem Aspekt der Freiwilligkeit der Beteiligung und dem häufig nur eingeschränkt oder nicht vorhandenen finanziellen Ausgleich stellt sich in diesem Zusammenhang die Frage nach einem Nachlassen der Motivation der Kernentwickler eines Projektes.[121]

Das zweite Argument wird von Cruz *et al.* unter dem Begriff *„Investitionsschutz“* zusammengefasst. Investitionsschutz bedeutet für Unternehmen die Amortisation der investierten Ressourcen. Besonders deutlich wird die Problematik, führt man sich vor Augen, dass eine Migration von proprietärer zu offener Software zunächst mit einem Mehraufwand an Migrations- und Schulungskosten verbunden ist (siehe Abb. 8) und sich erst langfristig in niedrigeren Lizenz- und/oder Betriebskosten rechnet.[122] Zwar reduziert sich das Risiko teilweise aufgrund der Verfügbarkeit des Quellcodes der Software, jedoch haben Unternehmen teilweise weder das Know-how noch die finanziellen Ressourcen, die Wartung einer Open Source Anwendung voranzutreiben. Vor allem große Unternehmen scheuen durch die meist hohe Zahl an umzustellenden Rechnern und Systemen bzw. des zu schulenden Personals das damit verbundene hohe Risiko einer Softwaremigration.

Der dritte Grund für die Wahl des Zielfaktors „Zukunftssicherheit“ ist seine offensichtliche Relevanz innerhalb der Zielgruppe der Unternehmen und Behörden. Sowohl die Untersuchungen von Cruz *et al.* wie auch die im Rahmen der Arbeit durchgeführte Unternehmensbefragung[123] ergaben eine entsprechende Anforderung der Betriebe und Organisationen an Open Source Projekte. Dabei weisen die Umfrageresultate darauf hin, dass sowohl Anwendungs- als auch Softwareunternehmen die Beendigung eines Open Source Projektes fürchten. Dementsprechend ist die Klärung der Einflüsse auf den Zielfaktor ein relevantes Evaluationsziel.

Zerlegung des Zielfaktors „Zukunftssicherheit“

Um nun eine Zerlegung des Zielfaktors „Zukunftssicherheit“ zu entwickeln, ist es zunächst einmal erforderlich, sich die Vorgänge in Open Source Projekten vor Augen zu führen, die ihn direkt bestimmen.

Eine Open Source Community besteht aus unterschiedlichen Gruppen von beteiligten Individuen[124]. Diese zeichnen sich dadurch aus, dass ihre Mitglieder sowohl im Grad der Beteiligung sowie auch in ihrem Wissen über die Software des Projektes variieren.

---

[121] siehe Kapitel 4.4.4 „Resultierende Schwächen von Open Source Projekten“

[122] bzw. in Abhängigkeit von der Situation rechnen kann

[123] siehe Kapitel 4.4.6 „Anforderungen der Unternehmen an Open Source Projekte“

[124] siehe Kapitel 4.3.2 „Rollen“

Eine generelle Unterteilung differenziert dabei im Wesentlichen drei Personengruppen innerhalb eines Open Source Projektes – die Benutzer der Software *(engl. „User“)*, die Entwickler *(engl. „Developer“)* sowie die Kernentwickler *(engl. „Core Developer“).* Alle Mitglieder der drei Gruppen stehen fortwährend vor der Entscheidung, aus dem Projekt auszutreten, auf demselben Level weiterhin aktiv zu bleiben oder den Grad ihrer Beteiligung zu erhöhen. Da die Zahl der aktiv beteiligten Personen in Open Source Projekten die wesentlichste Ressource darstellt [125], beeinflusst jede dieser Entscheidungen auch den Zielfaktor „Zukunftssicherheit“. Dabei muss jedoch zwischen den identifizierten Gruppen einer Community unterschieden werden. Aufgrund ihres Wissens um die Software sowie ihres Engagements stellt der Verlust eines Kernentwicklers die größte Gefährdung für die zukünftige Weiterentwicklung der Software in einem Open Source Projekt dar. Dahinter folgen der Austritt von Co-Entwicklern des Projektes, die überwiegend im Bereich der Programmerweiterungen sowie der Qualitätssicherung aktiv sind, sowie von aktiven Endnutzern, die Fehlermeldungen und Feature Requests liefern. Die Relevanz der beiden letztgenannten Gruppen bezieht sich dabei allerdings nicht ausschließlich auf ihre aktuellen Beiträge, sondern vor allem auch auf ihr zukünftiges Potential im Zuge eines Aufstiegs im Projekt [Rullani2006][Jensen2007].

Beeinflusst werden die Entscheidungen der beteiligten Personen durch drei Ressourcen, über die jeder Beteiligte in unterschiedlichem Maße verfügt.

Der erste Einflussfaktor ist die *Motivation* der Benutzer und Entwickler. Da die Arbeit in Open Source Projekten auf freiwilligem Engagement beruht, spielt dieser Faktor eine entscheidende Rolle. Dabei beginnt die Involvierung in ein Open Source Projekt in der Regel mit dem einfachen Bedarf oder Interesse an einer bestimmten Softwarelösung [Raymond2001][Ghosh2002]. Weitere Motive sind darüber hinaus die Verbesserung der eigenen Fähigkeiten, intellektuelle Stimulation oder die Etablierung eines Rufes innerhalb der Open Source Bewegung [126]. Wichtig dabei ist, dass die Motive der Beteiligten kein statisches Konstrukt sind, sondern sich im Laufe der Zeit ändern können [Rullani2006].

Die zweite Ressource der beteiligten Personen ist ihr *Wissen*. Wissen beeinflusst die Rolle eines beteiligten Benutzers oder Entwicklers dabei auf zwei Weisen. Zum einen bildet es insofern die Voraussetzung für eine tiefere Involvierung der Personen im Projekt, als dass für die Übernahme zentralerer Entwicklungsaufgaben zunehmend auch Kenntnisse über den Aufbau und die Architektur bzw. die Funktionsweise des Kerns einer Anwendung erforderlich sind. Zum anderen bestimmt das Wissen eines

[125] siehe Kapitel 4.4.3 „Resultierende Stärken von Open Source Projekten“

[126] siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

Beteiligten direkt seinen Status innerhalb der Projektcommunity. Da die Hierarchien innerhalb eines Open Source Projektes in der Regel nicht formalisiert sind, sondern aufgrund von wahrgenommener Kompetenz entstehen [Healy2003], ist ein projektinterner Aufstieg eng an das Wissen einer Person gebunden. Dies gilt besonders vor dem Hintergrund, dass die Besetzung verschiedener Rollen in Open Source Projekten nicht nur explizit, d.h. durch die Ernennung durch eine autorisierte Gruppe erfolgen kann, sondern auch implizit durch Durchführung bestimmter Tätigkeiten in dem Projekt oder Hervortun auf der Mailingliste [Jensen2007].[127]

Die dritte Ressource schließlich, die die Entscheidungen eines beteiligten Individuums zum Grad seiner Involvierung in Open Source Projekte beeinflusst, ist die *Zeit*, die sie oder er zur Verfügung hat, bzw. bereit ist zu investieren. Der Großteil der Entwickler von Open Source Projekten arbeitet nicht hauptberuflich an deren Weiterentwicklung [BCG2002], sondern beschäftigt sich in der Freizeit damit.[128] Dies führt dazu, dass vor allem für die Kernentwickler eines Projektes die Ressource „Zeit“ zu einem wesentlichen Faktor werden kann. Dies gilt insbesondere vor dem Hintergrund, dass der Aufwand zur Pflege und Weiterentwicklung einer Software mit zunehmender Komplexität ebenfalls steigt. Einen Hinweis auf dieses Problem bildet das zuvor bereits erwähnte Open Source Projekt „Dillo Web Browser“, dessen beiden Kernentwicklern die notwendige Zeit fehlt, die Software parallel zu ihrem Beruf weiterzuentwickeln.[129]

Abbildung 12 zeigt die aus den genannten Einflussfaktoren resultierende allgemeine Modellierung der Einflüsse auf den Zielfaktor „Zukunftssicherheit“. Die Abläufe sind dabei in mehrerer Hinsicht vereinfacht dargestellt. Erstens stehen die Bewertungen der Entscheidungen der Benutzer, Entwickler und Kernentwickler in keinem realistischen Verhältnis, sondern sollen lediglich die unterschiedliche Relevanz der identifizierten Gruppen einer Open Source Community betonen.

Zweitens beschreibt das Modell einen nur aufwärts gerichteten Prozess der Orientierung der beteiligten Personen innerhalb eines Open Source Projektes. Ein Rückschritt in der Beteiligung wie der Übergang vom Kernentwickler zum Entwickler ist nicht vorgesehen. Obwohl dies natürlich vorkommen kann, deckt sich das Modell in diesem Punkt mit dem bestehenden Trend, dass in einem Open Source Projekt involvierte Personen mit zunehmender Dauer der Beteiligung zentralere Rollen übernehmen [Jensen2007][Rullani2006]. Die dritte Vereinfachung besteht schließlich in der Aufteilung der Community in die drei Gruppen „Benutzer“, „Entwickler“ und

---

[127] siehe 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

[128] Die Mehrheit der Entwickler wenden nicht mehr als 10 Stunden pro Woche auf [Ghosh2002]

[129] siehe http://lists.auriga.wearlab.de/pipermail/dillo-dev/2006-September/003097.html

„Kernentwickler“. Jede der drei Gruppen kann weiter differenziert werden und besitzt gegebenenfalls projektspezifische Ausformungen

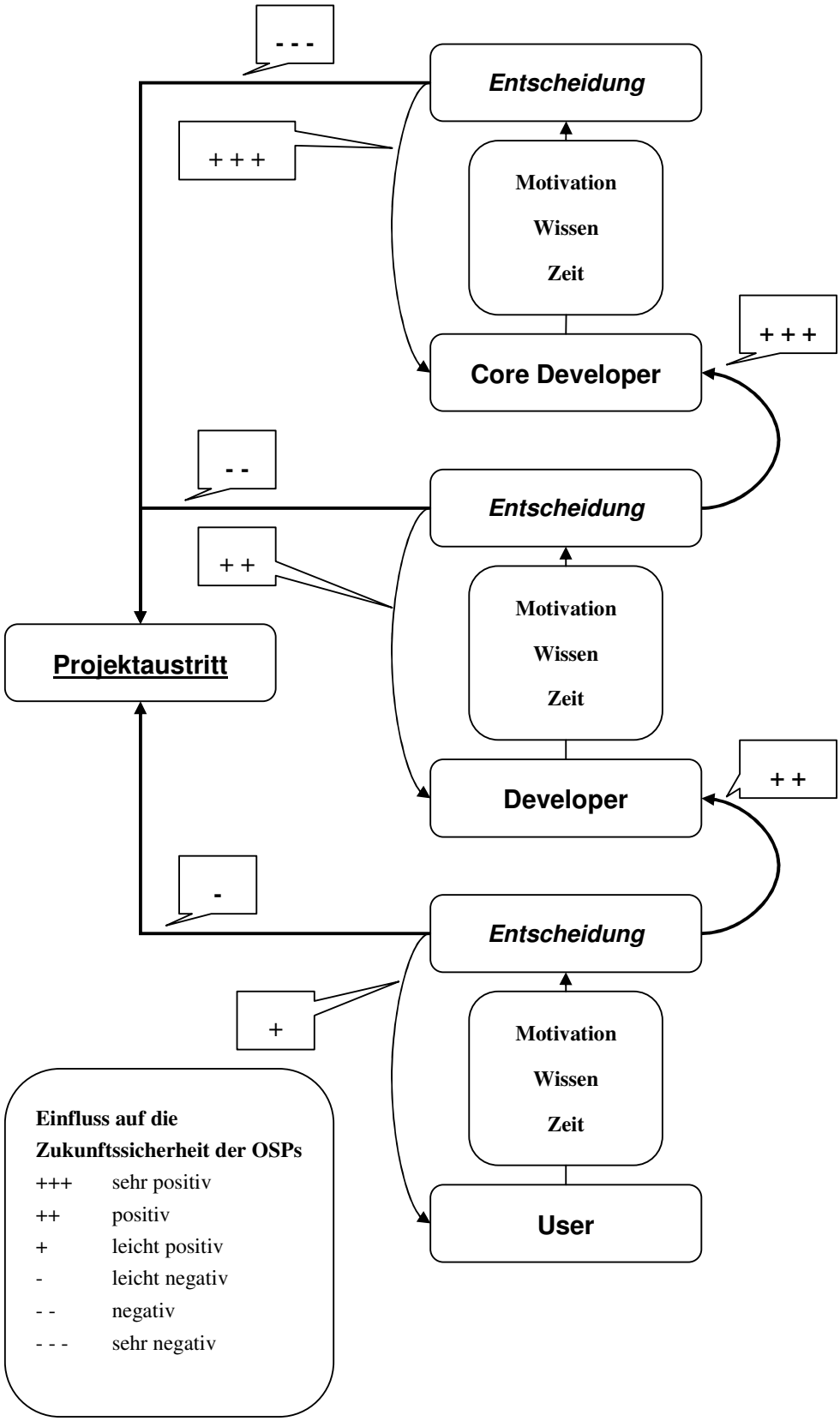

Abbildung 12: Allgemeine Modellierung der Einflüsse auf den Zielfaktor „Zukunftssicherheit“

Nachdem im ersten Schritt der Zerlegung des Zielfaktors „Zukunftssicherheit“ eine allgemeine Modellierung der Einflüsse verbunden mit der Identifikation der drei Ressourcen „Motivation“, „Wissen“ und „Zeit“ aller beteiligten Personen erfolgt ist, stellt sich im zweiten Schritt die Frage nach den bestimmenden Faktoren zur Ausprägung der genannten Ressourcen.

In der ersten Iterationsschleife der Evaluation innerhalb des GQM-Verfahrens werden dazu vier wesentliche Einflussfaktoren angenommen, die die Ausformung der Ressourcen der Benutzer und Entwickler und damit ihre Entscheidungen zum Grad ihrer Beteiligung am jeweiligen Open Source Projekt bestimmen. Die vier Einflussfaktoren sind die *„Kommerzielle Ausrichtung des Projektes“*, ein funktionierender *„Schulungsprozess für Entwickler“*, die *„Aktuelle Größe bzw. Aktivität des Projektes“* sowie eine von der Community getragene *„Projektideologie“* bzw. damit verbundene *„Projektziele“*. Aus ihnen resultieren insgesamt 9 Hypothesen zur Beeinflussung der drei Ressourcen der Entwickler und Benutzer.

1) Die kommerzielle Ausrichtung eines Open Source Projektes beeinflusst die Motivation der beteiligten Personen

Definition: *Die kommerzielle Ausrichtung eines Open Source Projektes beschreibt den Grad, mit dem vor allem das Projekt selbst, aber auch Unternehmen in seinem Umfeld versuchen, Einnahmen aus der entwickelten Software zu generieren.*

Die generelle Fragestellung in diesem Zusammenhang lautet:

„In welchem Maße beeinflussen finanzielle Interessen die Motive der Entwickler in Open Source Projekten, in denen der Entwicklungsprozess in der Regel charakterisiert ist durch eine freiwillige und unbezahlte Teilnahme?“

Dabei können zwei Aspekte voneinander getrennt werden:

Erstens gilt es, die Entwicklung des Motivs der Entwickler, welches zum Einstieg und Verbleib in ein Open Source Projekt führte, zu betrachten. So spielen finanzielle Interessen zu Beginn des Engagements noch eine eher untergeordnete Rolle und das Erlernen neuer Fähigkeiten oder der Bedarf an einer Softwarelösung steht deutlich im Vordergrund.[130] Dies bestätigt auch eine Studie von Rullani zum Verlauf der Motive von Entwicklern in Open Source Projekten über die Zeit [Rullani2006], in der finanzielle Beweggründe zu Beginn eines Engagements auf dem letzten Platz rangieren.

Über die Zeit findet dann allerdings ein Wandel in den Motiven der beteiligten Personen statt. Im Zuge dieses Wandels gewinnen vor allem auch finanzielle Interessen an

[130] siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

Bedeutung. Zwar spiegelt sich dieser Anstieg in der Bedeutung nicht durch vergleichsweise hohe absolute Antwortzahlen wider, sehr wohl aber durch eine hohe relative Steigerung der Wichtigkeit. Rullani begründet diesen Wandel mit einem *„Prozess des Bewusstwerdens“* über die Möglichkeiten in Verbindung mit Open Source Software. Demnach realisieren viele Entwickler erst im Zuge einer intensiveren Beteiligung in Open Source Projekten, dass sich ein finanzielles Potential dahinter verbirgt. Zudem ist darüber hinaus zu vermuten, dass eine Beteiligung der Entwickler an den finanziellen Erlösen eines Open Source Projektes an ein starkes Engagement geknüpft ist, das erst im Rahmen einer längerfristigen Unterstützung der Projekte möglich wird.

Der zweite Aspekt im Zusammenhang mit den finanziellen Motiven der in Open Source Projekten involvierten Personen ist die ansteigende Professionalisierung des Feldes. Große IT-Unternehmen wie IBM, HP, Novell oder SUN [Jensen2007][Clement2002] investieren zunehmend in den Bereich Open Source Software und es bilden sich Firmen wie Red Hat, JBoss oder Alfresco mit auf Open Source Produkten basierten Geschäftsmodellen heraus. Dies äußert sich zum einen in komplett unternehmensinitiierten Projekten [O'Mahony2005] sowie zum anderen in Unternehmen, die Dienstleistungen zu bestehender, offener Software bereitstellen und dafür die Projekte sowohl finanziell als auch in Form von Entwicklungsressourcen unterstützen [Clement2002]. Parallel zu dieser Entwicklung steigt damit auch die Zahl derjenigen Entwickler an, die hauptberuflich mit der Arbeit in Open Source Projekten beschäftigt sind. So belegen die Umfragen von Hars und Ou [Hars2001] sowie der Boston Consulting Group [BCG2002], dass bereits 16 bzw. 17,5% der Gesamtentwickler der untersuchten Projekte aus professionellen Softwareentwicklern bestehen, die ihren Lebensunterhalt mit der Weiterentwicklung von Open Source Software verdienen.[131] Diese Entwickler sind natürlich in besonderem Maße an der langfristigen Weiterentwicklung der entsprechenden Open Source Projekte interessiert, da sie existenzielle finanzielle Motive mit ihnen verbinden.

Zusammengefasst tragen finanzielle Interessen weniger zur initialen Entscheidung zu einem Engagement in Open Source Projekten bei, sondern beeinflussen vor allem die Motivation zu einer langfristigen Beteiligung der Kernentwickler. Darüber hinaus trägt die zunehmende Involvierung von Unternehmen in Open Source Projekten dazu bei, dass mehr und mehr Entwickler hauptberuflich an der Weiterentwicklung der Projekte arbeiten, und bildet somit einen guten Indikator für ihre Zukunftssicherheit.

---

[131] Eine Schätzung für den Linux Kernel 2.6.20 geht davon aus, dass 65% des in diesem Release beigetragenen Codes von bezahlten Entwicklern entwickelt wurde (http://lwn.net/Articles/222773/).

2) *Die kommerzielle Ausrichtung eines Open Source Projektes beeinflusst die Zeit der beteiligten Personen*

Neben dem Einfluss auf die Motivation der beteiligten Personen steht die kommerzielle Ausrichtung eines Open Source Projektes auch im Zusammenhang mit der verfügbaren Zeit der Entwickler. Da die Arbeit in Open Source Projekten speziell für deren Kernentwickler sehr zeitintensiv sein kann, ist es oftmals nicht einfach, die Anstrengungen zur Weiterentwicklung der jeweiligen Software und die Ausübung einer anderen beruflichen Tätigkeit parallel zu kombinieren. Dies gilt besonders vor dem Hintergrund, dass die Mehrzahl der Open Source Projekte lediglich über eine sehr geringe Zahl von Kernentwicklern verfügt [Krishnam2002][Gorling2003]. Die Einrichtung eines kommerziellen Zweiges kann in diesem Zusammenhang dabei helfen, den primär beteiligten Entwicklern wie zuvor erwähnt eine vollzeitliche Involvierung in das jeweilige Open Source Projekt zu ermöglichen.

3) *Ein funktionierender Schulungsprozess beeinflusst die Motivation der beteiligten Personen*

Definition: *Der Begriff „Schulungsprozess“ bezeichnet die Gesamtheit aller Strukturen und Aktivitäten in einem Open Source Projekt, die neuen oder bestehenden Benutzern und Entwicklern das Erlernen der Funktionsweise der Software bzw. der sozialen Regeln innerhalb des Projektes erleichtern.*

Definition: *Der durch eine einzelne Struktur oder Aktivität ausgelöste Wissensgewinn wird als „Lernprozess“ bezeichnet.*

Diesen Definitionen folgend besteht der Zweck eines etablierten Schulungsprozesses in Open Source Projekten vor allem in der Verteilung des im Projekt vorhandenen Wissens auf eine breitere Personengruppe. Aktivitäten, die eine derartige Verteilung unterstützen, sind beispielsweise das Beantworten von Fragen in den Mailinglisten, das Aufstellen von Codierrichtlinien, das Veröffentlichen der Architektur der Software oder die Beschreibung offener Features.

Dass allerdings das Initiieren eines Schulungsprozesses bzw. der damit verbundenen Lernprozesse in Open Source Projekten darüber hinaus motivierenden Charakter für beteiligte Personen besitzen kann, bestätigen empirische Untersuchungen. Die Umfragen von Hars und Ou [Hars2001], von Rullani [Rullani2006] sowie der Boston Consulting Group [BCG2002] belegen alle, dass *„Lernen“* bzw. *das „Entwickeln neuer Fähigkeiten“* zu den zentralen Motiven der Entwickler und aktiven Benutzer von Open Source Software zu ihrer Teilnahme im Projekt gehören. Hars und Ou betonen in diesem Zusammenhang außerdem, dass sich das Motiv „Lernen“ dabei nicht auf eine

bloße Verbesserung der Programmierfähigkeiten bezieht, sondern auch soziale Aspekte wie das Erlernen von Teamwork mit einbezieht [Hars2001].

Demzufolge besitzt ein funktionierender Schulungsprozess in Open Source Projekten einen wesentlichen Einfluss auch auf die Motivation der aktuellen und potentiellen Entwickler.

4) Ein funktionierender Schulungsprozess beeinflusst das Wissen der beteiligten Personen

Die zentrale Funktion eines etablierten Schulungsprozesses in Open Source Projekten besteht in der Verteilung des im Projekt vorhandenen Wissens auf eine breite Personengruppe.

In diesem Zusammenhang stellen Collins und Porras in ihrer Langzeitstudie über visionäre im Vergleich zu lediglich erfolgreichen Unternehmen die beiden Begriffe des *„time teller“* und des *„clock builder“* gegenüber [Collins2002]. Der erste bezeichnet demnach Unternehmensmanager, die die Geschicke ihres Unternehmens zumeist sehr autoritär und zentralisiert leiten und es dabei im Gegensatz zum „clock builder“ versäumen, zukunftsorientierte Maßnahmen wie die Etablierung von Schulungsprozessen zu initiieren, die dem Unternehmen auch nach ihrer Ära Stabilität verleihen.

Übertragen auf Open Source Projekte bedeutet dies, dass zur Sicherung der zukünftigen Weiterentwicklung der Projektsoftware ein Prozess zur Schulung der beteiligten Entwickler notwendig ist, um vorhandene Abhängigkeiten von einzelnen Kernentwicklern zu verringern.

Den potentiellen Erfolg eines instanziierten Schulungsprozesses für ein solches Vorhaben bestätigen Singh *et al.* [Singh2006] durch ihre Studie zu Lernprozessen in Open Source Projekten. Darin analysieren sie im Zuge einer sechsjährigen Beobachtung von 25 Projekten die Einflüsse zweier von ihnen identifizierter Formen des Lernens – *„Learning by doing“* sowie Lernen durch Interaktionen mit anderen Beteiligten.

5) Ein funktionierender Schulungsprozess beeinflusst die Zeit der beteiligten Personen

Schließlich beeinflusst ein funktionierender Schulungsprozess auch die Zeit der in Open Source Projekten beteiligten Personen. Dabei muss allerdings zwischen den drei identifizierten Personengruppen (siehe Abb. 12) unterschieden werden. Für Softwarebenutzer und Nicht-Kernentwickler eines Projektes bringt ein etablierter Schulungsprozess einen direkten Zeitgewinn mit sich. Bestehende Fragen werden in den Mailinglisten beantwortet oder in entsprechenden Dokumenten behandelt, wichtige Aufgaben sind veröffentlicht und der Aufbau sowie die Struktur der Software

dokumentiert, so dass ein Großteil der Einarbeitungszeit in die Software sowie die Prozesse im Projekt entfällt.

Dagegen ist die Einführung und Etablierung von Mechanismen zur Schulung neuer oder bestehender Entwickler und Benutzer für die Kernentwickler eines Projektes zunächst mit einem erheblichen Mehraufwand verbunden. Dieser manifestiert sich zum einen in der Teilnahme an Diskussionen in den Mailinglisten zu Fragen der übrigen Projektteilnehmer sowie zum anderen in der Erstellung entsprechender Dokumente zur Funktionsweise der Software und des Entwicklungsprozesses. Besonders vor dem Hintergrund, dass es vor allem die Kernentwickler sind, die in Open Source Projekten auch den Großteil der darüber hinaus anstehenden Aufgaben bewältigen müssen [Mockus2002], stellt die Schaffung von Strukturen zur Rekrutierung neuer Entwickler eine große Herausforderung dar, an der viele Projekte scheitern [O'Mahony2005].

Sind jedoch Strukturen und Dokumente zur Dokumentation der Software etabliert und ist eine initiale sehr starke Beteiligung in den Mailinglisten erfolgt, so ist davon auszugehen, dass auch hinzukommende Entwickler vermehrt selbst unterstützende Aufgaben wie die Gestaltung der Diskussionen in den Mailinglisten [Stuermer2005] übernehmen. Folglich bildet die Instanziierung und Ausformung eines Schulungsprozesses für Kernentwickler eines Open Source Projektes vor allem eine Investition in die Zukunft, verbunden mit der Hoffnung, dass andere beteiligte Personen im Zuge eines langfristigen Engagements zentralere Aufgaben übernehmen [Rullani2006][Jensen2007] und die Kernentwickler damit entlasten.

6) *Die aktuelle Größe/Aktivität eines Open Source Projektes beeinflusst die Motivation der beteiligten Personen*

Ein wesentliches Motiv von Benutzern und Entwicklern zur Beteiligung in Open Source Projekten ist die Erarbeitung eines Rufes innerhalb der Projektcommunity oder der Open Source Bewegung selbst [Watson2005][Hars2001][Bonaccorsi2005][BCG2002]. Dazu werden gemeinsam mit dem Quellcode einer Software auch die Namen der Codeautoren verbreitet. Dies geschieht in Form von Listen auf der Projekthomepage, so genannten *Credits*-Dateien, oder auch durch direkte Hinweise im Programmcode. Damit im Zuge einer solchen Verbreitung der Ruf und damit der soziale Status eines Entwicklers wirklich verbessert werden kann, ist es neben der Qualität der Beiträge vor allem entscheidend, ein breites Publikum zu erreichen. Dabei sind zwei Aspekte voneinander zu unterscheiden.

Zum einen beeinflusst die Größe eines Open Source Projektes gemessen an der Anzahl seiner Communitymitglieder die Möglichkeit zur Verbreitung des Rufes eines Entwicklers [Bonaccorsi2005]. Je größer demnach eine Projektcommunity ist, desto höher ist automatisch die Zahl der Adressaten, die direkt mit einem Codierbeitrag

konfrontiert werden, und desto größer ist damit auch die Motivation von Entwicklern zur Einsendung solcher Beiträge.

Zum anderen spielt auch die Popularität eines Projektes außerhalb der Projektcommunity eine große Rolle im Hinblick auf die potentielle Zahl der Adressaten eines Codierbeitrages. Eine Möglichkeit zur Beeinflussung der Popularität eines Projektes besteht in den Vitalitätsranglisten bekannter Kollaborationsplattformen wie Sourceforge. Diese messen die Aktivität der Projektmitglieder auf den Mailinglisten, in den Bug-Tracking-Systemen oder den Versionsverwaltungssystemen. Dies bedeutet, dass eine hohe Aktivität innerhalb eines Open Source Projektes zu positiven Marketingeffekten und damit zu einer größeren Verbreitung der Projektsoftware führen kann. Einen solchen Marketingeffekt bestätigt auch Stürmer [Stuermer2005].

Darüber hinaus kann die Größe eines Projektes gemessen an der Gesamtzahl der *„Lines of Code“* (LOC) ein Indikator für die Reife einer Software sein. Eine solche Metrik besitzt zwar entsprechend der Kritik an der Vorgehensweise von Cruz *et al.*[132] nur Aussagekraft im Kontext eines bestimmten Anwendungsfeldes, allerdings identifiziert Michlmayr [Michlmayr2005a] im Rahmen seiner Untersuchungen zur Reife von Open Source Software durchaus einen Zusammenhang zwischen der Größe und dem Erfolg gemessen an der Anzahl von Downloads einer Software. Dies lässt den Schluss zu, dass neben der Größe der Projektcommunity und der Aktivität der Projektmitglieder auch die Größe der Software einen Einfluss auf den Verbreitungsgrad einer Anwendung und damit verbunden auf das Potential zur Etablierung eines Rufes besitzt.

7) *Die aktuelle Größe/Aktivität eines Open Source Projektes beeinflusst die Zeit der beteiligten Personen*

Die zur Verfügung stehende Zeit ist die zweite Ressource der in Open Source Projekten beteiligten Personen, die durch die Größe der Projekte bzw. ihrer Community beeinflusst wird. Je mehr Benutzer und Entwickler ein Projekt umfasst, desto höher ist der Grad der Arbeitsteilung und desto geringer ist damit der zeitliche Aufwand jeder einzelnen Person an einmaligen Tätigkeiten wie die Pflege der Webseite, die Veröffentlichung einer neuen Version der Software, etc. Dieser Zusammenhang muss allerdings mit zwei Einschränkungen versehen werden:

Die erste bezieht sich dabei auf die Aktivität der beitragenden Personen. Nur aktive Benutzer und Entwickler helfen, die im Projekt anstehenden Aufgaben auf eine breitere Gruppe von Personen zu verteilen, und damit den Zeitaufwand für jeden Einzelnen zu reduzieren. Ein großer Kreis von Entwicklern, die einem Open Source Projekt zwar zu seinem aktuellen Status verholfen haben, inzwischen aber nicht mehr aktiv sind, trägt

[132] siehe Kapitel 5.2.2 „Probleme“

nicht zu einer Reduzierung des Zeitaufwandes der aktuell beteiligten Personen bei. Dementsprechend muss neben dem Aspekt der Größe einer Projektcommunity auch die Aktivität seiner Mitglieder betrachtet werden.

Die zweite Einschränkung dagegen ist technischer Natur. Die Skalierbarkeit des Entwicklungsprozesses einer Software ist bezogen auf den Implementierungsvorgang begrenzt durch den Grad der Modularisierung der Software [Michlmayr2004][Brooks2000]. Entsprechend „Brooks Law“ können nur kleine Teams von Entwicklern effektiv an der Realisierung eines einzelnen Moduls arbeiten, da ansonsten der Kommunikations- und Synchronisationsaufwand den eigentlichen Entwicklungsaufwand überschreitet [Brooks2000].[133]

Überträgt man die obigen Überlegungen auf real existierende Open Source Projekte so hat allerdings die zweite Einschränkung wenig praktische Relevanz, da nur eine geringe Anzahl von Open Source Projekten eine hohe Zahl von aktiven Entwicklern besitzt [Krishnam2002] [134], so dass eine Steigerung der Entwicklerzahl nicht zu einer Überschreitung der von Brooks angesprochenen kritischen Schwelle führt.

Zum anderen erledigen hinzukommende Entwickler und aktive Benutzer häufig Aufgaben, die nicht den Einschränkungen von „Brooks Law“ unterworfen sind. Solche Aufgaben sind die Unterstützung von anderen Benutzern der Software in den Mailinglisten [Stuermer2005], Codeinspektionen und Fehlersuchen [Michlmayr2004] sowie andere Tätigkeiten zur Pflege und Wartung der Software. Darüber hinaus ermöglicht eine Plug-In-Architektur die Entwicklung von Programmerweiterungen, ohne Änderungen am Kern der Anwendung vornehmen zu müssen [Stuermer2005].

8) Projektideologie und Projektziele beeinflussen die Motivation der beteiligten Personen

Einer der wesentlichen Bestandteile des Rahmenwerkes von Collins und Porras [Collins2002] zur Unterscheidung von visionären und lediglich erfolgreichen Unternehmen besteht in der Identifikation einer Kernideologie *(engl. „Core ideology“)* in visionären Unternehmen. Eine Kernideologie setzt sich dabei aus zwei Komponenten zusammen.

Zum einen beinhaltet sie eine kleine Menge von grundlegenden Prinzipien, die so genannten „Kernwerte“ *(engl. „Core values“)* eines Unternehmens, zum anderen umfasst sie darüber hinaus die Formulierung einer Daseinsberechtigung *(engl. „Purpose“)* der Organisation, die über den Zweck der Rentabilität hinausgeht.

---

[133] siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

[134] siehe Kapitel 4.4.4 „Resultierende Schwächen von Open Source Projekten“

Das Ziel einer Kernideologie nach Collins und Porras die Wahrung der Identität eines Unternehmens und damit verbunden eine klare Orientierung. Dabei betonen sie deutlich, dass es nicht *die* Kernideologie gibt, sondern jede Formulierung von dauerhaften Werten und Prinzipien eine individuelle, unternehmensspezifische Ausprägung haben muss. Diese formulierten Kernwerte dienen dann auch zur Selektion innerhalb der Mitarbeiter:

> *„Great places to work only for those who buy in to the core ideology; those who don't fit with the ideology are ejected like a virus"*
>
> *- [Collins2002, S.89]*

Die Umsetzung einer solchen Kernideologie in Unternehmen resultiert Collins und Porras zufolge in „kultähnliche Kulturen" *(engl. „cult-like cultures")*, die im Fall einer gemeinschaftlich getragenen Akzeptanz der formulierten Kernwerte mit einer besonderen Motivation der Mitarbeiter einhergehen, da ihnen das Gefühl vermittelt wird, Teil einer besonderen Institution zu sein.

Neben der Fokussierung auf Kernwerte und der damit verbundenen Konzentration auf das Wesen einer Organisation identifizieren Collins und Porras eine zweite Komponente im Erfolg visionärer Unternehmen – fortschrittsorientierte Maßnahmen.

Fortschrittsorientierte Maßnahmen sollen sicherstellen, dass sich ein Betrieb an ständig wechselnde externe Bedingungen anpassen kann oder den Entwicklungen sogar voraus ist. Eine solche Vorgehensweise ist unter anderem das, was Collins und Porras unter dem Begriff *„Big Hairy Audacious Goals"* zusammenfassen. Darunter verstehen sie zukunftsorientierte, fordernde und oftmals sogar riskante Ziele, die visionäre Unternehmen fortwährend neu formulieren und anstreben. Solche Ziele müssen einerseits natürlich mit den Kernwerten einer Organisation übereinstimmen, dürfen andererseits allerdings alle anderen etablierten Prozesse, Produkte, Aktivitäten oder Mechanismen in Frage stellen.

Das Ergebnis ist eine Unternehmensform, die sowohl konservative, stabilisierende Maßnahmen als auch die Fokussierung auf eine Kernideologie und auch progressive, fortschrittsorientierte Bestandteile kombiniert. Collins und Porras wählen zur Verdeutlichung des Schlüsselaspekts ihres Rahmenwerkes das in Abbildung 13 dargestellte Symbol.

Abbildung 13: Schlüsselkonzept visionärer Unternehmen nach Collins und Porras

Im Hinblick auf die Prozesse in Open Source Projekten stellt sich die Frage: „Sind die beiden genannten Aspekte des Rahmenwerkes von Collins und Porras übertragbar?“

Die Beteiligung in freien Softwareprojekten erfolgt in der Regel auf freiwilliger Basis sowie ohne einen finanziellen Ausgleich. Dies bedeutet, dass für ein langfristiges Engagement von Entwicklern in Open Source Projekten eine andere dauerhafte Projektbindung als finanzielle Interessen geschaffen werden muss [Rothfuss2002]. Dabei unterstützt die Existenz einer Kernideologie.

Eine Kernideologie in Open Source Projekten repräsentiert grundlegende Werte, Normen und Prinzipien, die sich einerseits in der Projektsoftware selbst andererseits vor allem auch in den sozialen Strukturen innerhalb einer Projektcommunity widerspiegeln können. Dies ermöglicht eine Identifikation der beteiligten Personen mit dem Projekt, die über ihr Produkt hinausgeht.

Die Wichtigkeit einer solchen Identifikation für ein langfristiges Engagement der Entwickler bestätigt Rullani, der von der Community geteilte Werte und Normen sowie das Agieren innerhalb des sozialen Umfeldes als wesentliche Motive für eine andauernde Beteiligung in Open Source Projekten identifiziert [Rullani2006].

Der zweite genannte Aspekt des Rahmenwerks von Collins und Porras ist die Formulierung fordernder und unter Umständen gewagter Ziele. Eine solche Maßnahme bewirkt in Unternehmen vor allem die Motivation der Mitarbeiter sowie eine fortschrittsorientierte Ausrichtung [Collins2002]. Für Open Source Projekte ist dabei vor allem der erste Punkt von entscheidender Bedeutung. Neben der angeführten Förderung der Identifikation mit der Projektgemeinschaft eines Open Source Projektes ist es darüber hinaus erforderlich, Anreize für Beiträge zu schaffen.

In diesem Zusammenhang nennt die Boston Consulting Group das Motiv der „Intellektuellen Stimulation“ als wesentlichsten Beweggrund zur Partizipation von Entwicklern am Softwareentwicklungsprozess in Open Source Projekten [BCG2002]. Dieses Motiv betont im Wesentlichen den Bedarf der beteiligten Personen an einer programmiertechnischen Herausforderung. Eine Möglichkeit, einem solchen Bedarf in

Open Source Projekten zu begegnen, ist die von Collins und Porras für Unternehmen vorgeschlagene Formulierung von fordernden, antreibenden und gewagten Projektzielen.

Zusammenfassend wird im Rahmen der Arbeit davon ausgegangen, dass die von Collins und Porras entwickelten Konzepte einer Kernideologie sowie herausfordernder Ziele durchaus auf Open Source Projekte übertragbar sind und die Motivation der in den Projekten beteiligten Personen zu einem langfristigen Engagement auf zwei Arten positiv beeinflussen. Zum einen ermöglichen formulierte Kernwerte eine Identifikation der Entwickler mit dem jeweiligen Open Source Projekt über die Projektsoftware hinaus, zum anderen begegnen fordernde Ziele dem empirisch ermittelten Bedarf an einer Herausforderung.

9) Projektideologie und Projektziele beeinflussen das Wissen der beteiligten Personen

Neben dem Einfluss auf die Motivation prägen Projektideologie und Projektziele darüber hinaus teilweise auch das Wissen der beitragenden Personen. Dabei ist weniger das technische Wissen um die Projektsoftware gemeint, als vielmehr ein grundlegendes Verständnis der sozialen Umgebung der Projekte. Dazu beantworten Ideologie und Ziele eines Open Source Projektes allgemein zwei Fragen zum Wesen einer Projektcommunity:

- Wer sind wir?
- Was wollen wir?

Beide Aspekte helfen einerseits bei der initialen Entscheidung zu einer Beteiligung in einem Open Source Projekt und können sich andererseits in konkreten Regeln innerhalb des Entwicklungsprozesses manifestieren. Solche Regeln können Verhaltensrichtlinien für die Kommunikation auf den Mailinglisten, Codierrichtlinien oder auch inhaltliche Einschränkungen wie beispielsweise die Begrenzung der Features im Zuge einer Fixierung auf leistungsspezifische Merkmale sein. Das Kennenlernen solcher Regeln und Normen eines Open Source Projektes bildet gerade zu Beginn eine Hürde für neue Entwickler, die durch die offene Formulierung von Kernwerten sowie Projektzielen verringert werden kann.

Übertragung der Zerlegung in ein GQM-Modell

Nachdem zuvor allgemein die Zerlegung des Zielfaktors *„Zukunftssicherheit von Open Source Projekten“* in die Teilaspekte kommerzielle Ausrichtung, etablierter Schulungsprozess für neue und bestehende Entwickler, Größe und Aktivität des Projektes sowie Projektideologie und Projektziele erläutert wurde, muss diese im zweiten Schritt in ein dreistufiges GQM-Modell integriert werden. Tabelle 11 zeigt dazu die im Rahmen der Arbeit vorgeschlagene Operationalisierung des Zielfaktors nach dem

GQM-Verfahren. Das Ziel besteht dabei in der Evaluation der Zukunftssicherheit von Open Source Projekten aus der Sicht von Unternehmensmanagern. Aufgeteilt wird der Zielfaktor in insgesamt 11 Fragen, die sich auf die vier identifizierten Teilaspekte beziehen. Zur Beantwortung der Fragen werden drei Datenquellen herangezogen – die Webseite des Projektes, das Versionsverwaltungssystem und eine Umfrage an die Projektadministratoren der Open Source Projekte.

Die ersten drei Fragen beschäftigen sich dabei mit der kommerziellen Ausrichtung von Open Source Projekten. Dies umfasst die Identifizierung der Einnahmequellen der Projekte, die Ermittlung der Anzahl der bezahlten Entwickler sowie der Anzahl der Unternehmen im Projektumfeld, die Support zur Software anbieten. Als Metriken zu den drei Fragen dienen zum einen Angaben zum betrieblichen Umfeld auf der Webseite der Open Source Projekte sowie zum anderen die konkrete Befragung der Projektadministratoren zu diesem Thema.

Die nächsten drei Fragen dienen der Operationalisierung des internen Schulungsprozesses, der in Open Source Projekten zur Verteilung des Wissens etabliert ist. Die drei Fragen erfassen dazu drei unterschiedliche Aspekte. Erstens spielt der Umfang des für die Bearbeitung zentraler Entwicklungsaufgaben notwendigen Wissens eine wichtige Rolle. Zweitens müssen zur Verteilung des Wissens Strukturen und Mechanismen vorhanden sein, die einen Schulungsprozess überhaupt ermöglichen. Drittens schließlich stellt sich die Frage, wie viel Zeit die Kernentwickler der Projekte in einen solchen Prozess investieren? Als Erfassungsmetriken dienen erneut die Webseite sowie die Befragung der Projektadministratoren selbst.

Der dritte Abschnitt, die Einordnung der Projektideologie und der Projektziele in das von Collins und Porras vorgeschlagene Modell, muss in zwei Schritten erfolgen. Zum einen werden die Projektadministratoren zur Existenz der zwei Aspekte befragt. Zum anderen sieht das Rahmenwerk von Collins und Porras explizit vor, dass Kernwerte, Daseinsberechtigung sowie Projektziele nicht bloß vorhanden, sondern vor allem auch in konkreten Mechanismen manifestiert sein müssen. Dies wird im zweiten Schritt durch eine Untersuchung der Webseite als primäre Präsentationsinstanz der Projekte überprüft.

Die letzten drei Fragen dienen zur Messung der Größe der Projektsoftware bzw. der Größe und der Aktivität der Projektcommunity. Zur Messung der drei Aspekte werden Daten aus den Versionsverwaltungssystemen der Open Source Projekte herangezogen.[135]

---

[135] Für eine genaue Beschreibung der unterschiedlichen Datenquellen sowie der Erfassung der vier identifizierten Teilaspekte des Zielfaktors „Zukunftssicherheit“ siehe Kapitel 6 „Informationsgewinnung“

Tabelle 9: GQM-Zerlegung des Zielfaktors „Zukunftssicherheit von Open Source Projekten"

Goal	
Absicht	Evaluation
Betrachteter Aspekt	der Zukunftssicherheit
Objekt	von Open Source Projekten
Perspektive	aus der Sicht von Unternehmensmanagern

Question	**Metrics**
Q1 Welche Einnahmequellen besitzt das Open Source Projekt?	**M1.1** Befragung der Kernentwickler **M1.2** Angaben auf der Projekthomepage
Q2 Wie viele bezahlte Entwickler besitzt das Open Source Projekt?	**M2.1** Befragung der Kernentwickler
Q3 Wie viele Unternehmen bieten Support für die Software des Open Source Projektes an?	**M3.1** Befragung der Kernentwickler **M3.2** Angaben auf der Projekthomepage
Q4 Wie lange benötigt ein neuer Entwickler, um das Wissen eines Kernentwicklers zu erlangen?	**M4.1** Einschätzung der Kernentwickler
Q5 Welche Mechanismen sind in dem Open Source Projekt etabliert, um neue Entwickler dabei zu unterstützen, dieses Wissen zu erlangen?	**M5.1** Befragung der Kernentwickler **M5.2** Überprüfung der Projekthomepage auf vorhandene Dokumente und Strukturen zur Unterstützung von Benutzern und Entwicklern
Q6 Wie viel Zeit verbringen die Kernentwickler des Projektes damit, andere Entwickler zu unterstützen?	**M6.1** Befragung der Kernentwickler
Q7 Welche Ideologie besitzt das Projekt und wie ist diese Ideologie manifestiert?	**M7.1** Befragung der Kernentwickler **M7.2** Überprüfung der Projekthomepage auf formulierte Kernwerte oder eine Daseinsberechtigung **M7.3** Analyse der Mailinglisten-Archive auf gemeinschaftlich getragene Prinzipien
Q8 Welche Ziele besitzt das Projekt und wie sind diese Ziele manifestiert?	**M8.1** Befragung der Kernentwickler **M8.2** Überprüfung der Projekthomepage auf formulierte Kernwerte **M8.3** Analyse der Mailinglisten-Archive auf gemeinschaftlich getragene Prinzipien
Q9 Wie groß ist die Projektcommunity?	**M9.1** Angaben zur Entwicklerzahl auf der Projekthomepage **M9.2** Anzahl der Entwickler, die in das Versionsverwaltungssystem eingecheckt haben?
Q10 Wie groß ist die Projektsoftware?	**M10.1** Anzahl der „Lines of Code“ der Software im Versionsverwaltungssystem
Q11 Wie aktiv ist die Projektcommunity?	**M11.1** Anzahl der in den letzten 3, 6 und 12 Monaten aktiven Entwickler in den Versionsverwaltungssystemen **M11.2** Anzahl der in den letzten 3, 6 und 12 Monaten hinzugefügten „Lines of Code“ in den Versionsverwaltungssystemen **M11.3** Aktivität auf den Mailinglisten des Projektes

5.3.2 Das Einsatzgebiet „Web Content Management Systeme“

Der zweite Schritt zur genaueren Festlegung des Kontextes der Evaluation besteht in der Wahl einer Softwarekategorie. Dies ist im Rahmen der Arbeit die Kategorie der *Web Content Management Systeme (WCM-Systeme).*

Definition

Web Content Management Systeme sind allgemein Softwaresysteme zur Erstellung, Gestaltung, Veröffentlichung, Kontrolle und Verwaltung von Inhalten *(engl. „Content“).* Im Gegensatz zu anderen Content Management Systemen konzentrieren sich WCM-Systeme dazu im Wesentlichen auf internetbasierte Formate wie z.B. HTML, XML oder GIF. Die Besonderheit besteht dabei vor allem in der expliziten Trennung von Inhalten, Metadaten[136] wie Layoutinformationen und den hinter einer Webseite stehenden Funktionalitäten. Dies ermöglicht es Unternehmen zum einen, die Form der Präsentation von Inhalten variabel an spezifische Gegebenheiten wie Kundenanforderungen anzupassen. Zum anderen erleichtert eine solche Trennung die Zuordnung der internen Ressourcen in Form der Angestellten auf ihre jeweiligen Kompetenzgebiete. Im Zuge dessen können die Autoren von Inhalten diese veröffentlichen, ohne Kenntnisse des resultierenden Layouts besitzen zu müssen. Umgekehrt ist die Arbeit des Webdesigners nicht an das Vorhandensein spezifischer Inhalte gebunden.

Aufgrund der unklaren Abgrenzung des Umfangs des Begriffs „Content Management“ ist die starke Streuung der Anwendungen in ihrer Komplexität sowie der Anzahl der integrierten Features eine weitere Besonderheit in diesem Feld.

Gründe für das Einsatzgebiet „Web Content Management Systeme“

Die Auswahl des Einsatzgebietes Web Content Management Systeme im Rahmen der Evaluation des Zielfaktors „Zukunftssicherheit“ von Open Source Projekten basiert dabei auf drei Gründen.

Der Erste davon besteht in dem offensichtlichen Aufschwung, den das Feld nach wie vor erlebt. Die beiden Studien der Gartner Group aus den Jahren 2005 und 2007 [Gartner2005][Gartner2007] zum Marktsegment der Web Content Management Systeme bezeichnen die Aussichten als vielversprechend. Im Zuge der zunehmenden Nutzung des Internets durch Unternehmen als Präsentationsmedium für betriebliche Inhalte steigt auch die Nachfrage an Softwaresystemen zur Professionalisierung der Webauftritte. Dabei existieren aktuell vor allem zwei Tendenzen.

---

[136] Als Metadaten werden Daten bezeichnet, die Informationen über andere Daten enthalten

Zum einen beinhalten WCM-Systeme inzwischen zunehmend mehr als nur Funktionen zur Erstellung, Aufbereitung und Darstellung von Inhalten. So ermöglichen fortgeschrittene Anwendungen darüber hinaus bereits auch Formen der Interaktion mit Kunden, die Organisation betrieblicher Workflows oder auch das Abwickeln von Geschäftsprozessen zwischen verschiedenen Unternehmen bzw. zwischen Unternehmen und Kunden. Zum anderen bieten gerade die angesprochene Erweiterung des Funktionsumfangs der Anwendungen sowie die in der Folge zunehmender Standardisierung abnehmenden technischen Eintrittsbarrieren auch für kleine Unternehmen die Chance, durch innovative Ideen und Produkte Marktanteile zu gewinnen. Die Folge davon ist ein breiteres Spektrum an Produkten am Markt sowie eine stärkere Verteilung der Marktanteile. Vor allem im Zuge des zweiten Aspekts sind es dabei unter anderem auch Open Source Projekte, die durch ihre geringen Anschaffungskosten für die Software eine treibende Kraft zur Herausforderung der etablierten Unternehmen bilden können [Gartner2007].

Der zweite Grund zur Auswahl des Einsatzgebietes WCM-Systeme besteht in der Relevanz des Zielfaktors „Zukunftssicherheit“ in diesem Zusammenhang. Eine Investition in ein WCM-System ist in der Regel eine langfristige strategische Entscheidung.

Dies liegt zum einen daran, dass die Anschaffung und Anbindung an die unternehmensinterne Infrastruktur einer entsprechenden Anwendung zunächst mit Mehrkosten verbunden sind, die sich erst langfristig amortisieren. Dieser Aspekt wird verstärkt durch den Umstand, dass auch die Bedienung eines solchen Systems durch die Angestellten erst erlernt werden muss.

Zum anderen ist die Umstellung von einer einmal gewählten Lösung auf ein neues System mit einem erheblichen Aufwand verbunden. Dieser vergrößert sich, je länger eine bestehende Anwendung im Einsatz ist. WCM-Systeme unterscheiden sich in ihrem grundsätzlichen Aufbau, in den Funktionen, die sie realisieren, in der Art, wie sie die Funktionen realisieren oder auch in der Art der Anbindung an externe Anwendungen. Dies kann sich beispielsweise in unterschiedlichen Sprachen zur Auszeichnung von *Templates* [137], in unterschiedlichen Ausprägungen des Rollenkonzepts und der Workflow-Funktionalitäten oder in auseinander gehenden Datenbanken, die vom jeweiligen System unterstützt werden, manifestieren. Die Folge davon ist, je länger bzw. je mehr bestehende Inhalte mit einem bestimmten WCM-System gepflegt werden, desto größer ist auch die Abhängigkeit von der Anwendung und desto schwieriger ist damit verbunden eine Migration auf ein neues Softwaresystem. Vor diesem Hintergrund ist der Zielfaktor „Zukunftssicherheit“ ein wichtiges Entscheidungskriterium.

[137] Ein „Template“ bezeichnet eine Schablone zur Gestaltung von Inhalten

Der dritte Grund schließlich für die Wahl des Einsatzgebietes ist das große, bereits bestehende Spektrum an Open Source Projekten im Feld der WCM-Systeme. Open Source Projekte wie das *Plone-, das Typo3-, das Magnolia-, das Mambo-(bzw. Joomla)* oder das *OpenCMS-Projekt* sind Produzenten von Open Source WCM-Systemen, die qualitativ durchaus mit proprietären Anwendungen mithalten können und sich folglich zu ernstzunehmenden Herausforderern für große Unternehmen in diesem Bereich entwickeln können bzw. entwickelt haben [Gartner2007].

6 Informationsgewinnung

In der Phase der Informationsgewinnung wird die innerhalb der Planungsphase entwickelte Zerlegung von Open Source Projekten in eine konkrete Datensammlung umgesetzt.

Im für diese Arbeit betrachteten Fall werden hierzu insgesamt drei Daten- und Informationsquellen herangezogen, die zur Analyse der Vorgänge in Open Source Projekten benutzt werden können. Dies sind die frei verfügbaren CVS- bzw. SVN-Daten, die Webseiten sowie die Befragung der Administratoren von Open Source Projekten. Die Webseiten liefern meist einen guten Überblick über die Strukturen und Ressourcen von Open Source Projekten. So werden in der Regel die Funktionalitäten der Software beschrieben, Materialien zur Software verlinkt, die Community und ihre Ziele präsentiert und häufig auch historische Daten angegeben. Zudem bilden die Webseiten von Open Source Projekten die Schnittstelle zu anderen Infrastrukturkomponenten wie Foren, Bug-Tracker oder Quellcodeverwaltungs-systemen.

Allerdings erhält der Betrachter keinen detaillierten Überblick über die ablaufenden Prozesse und Aktivitäten in den Projekten. So reicht eine Betrachtung der Webseiten von Open Source Projekten nicht aus, um festzustellen, wie aktiv und regelmäßig die Entwicklung der Software vorangetrieben wird, wie viele Entwickler insgesamt aktiv am Entwicklungsprozess beteiligt sind oder wie viel Zeit die Kernentwickler damit verbringen, bestehende Fragen zur Software zu beantworten. Hierzu dienen die zwei anderen genannten Daten- und Informationsquellen.

Während die CVS- und SVN-Daten dabei quellcodeorientierte Informationen zum genauen Verlauf der Entwicklung der Software in Open Source Projekten liefern, ermöglicht die Befragung der Projektadministratoren die Ermittlung zusätzlicher Informationen zu den Aktivitäten und Prozessen, die nicht über die Webseiten der Projekte ersichtlich sind. Ein Beispiel dafür ist die Anzahl der bezahlten Entwickler im Open Source Projekt.

Einer Befragung der Projektadministratoren sollte allerdings immer eine genaue Betrachtung der Webseite des jeweiligen Projektes vorangehen, um unbeliebte redundante Fragestellungen zu vermeiden.

Als Stichprobe zur Untersuchung der entwickelten Zerlegung des Zielfaktors „Zukunftssicherheit“ in dem ausgewählten Kontext dienen sechs Open Source Projekte aus dem Feld der „Web Content Management Systeme“. Die sechs Projekte sind Mambo, Magnolia, Typo3, OpenCMS, Drupal und TGS CMS. Sie umfassen fünf

etablierte Open Source Projekte, die bereits eine gewisse Benutzergemeinschaft um die Projektsoftware herum etabliert haben sowie TGS CMS, als ein sehr junges Projekt[138], das erst versucht, eine solche Gemeinschaft zu entwickeln.

6.1 CVS/SVN-Daten

Die erste Quelle wichtiger Informationen bilden die Versionsverwaltungssysteme der Projekte. Die meisten Open Source Projekte besitzen zur Organisation des Quellcodes ihrer Software ein solches System. Etabliert haben sich dabei im Wesentlichen CVS („Concurrent Version System“) und SVN („Subversion“).[139] Beide Systeme bieten den Vorteil, dass sie sowohl die Verteilung des Quellcodes ermöglichen als auch die Integration und Koordination neuer Beiträge. Dazu werden die unterschiedlichen Versionen einer Anwendung zentral auf einem Server gespeichert und separat verwaltet. Dies erfolgt in einem so genannten *„Repository“* (Verzeichnisbaum), das es ermöglicht, unterschiedliche Versionen sowie unterschiedliche Fassungen einer Version einer Software unabhängig voneinander zu speichern. Der letzte Fall kann z.B. auftreten, wenn im Nachhinein Änderungen an Version 1.0 eines Programms vorgenommen werden müssen, obwohl bereits Version 2.0 verfügbar ist. Es kommt zu einer Verzweigung der Version 1.0, die sich gegebenenfalls unterschiedlich entwickeln kann (siehe Abb.14).

Da CVS und SVN neben dem Quellcode selbst auch Informationen zu seiner Historie wie die Namen der Autoren sowie den Zeitpunkt oder den Umfang vorgenommener Änderungen speichern, bieten sie einen reichhaltigen Fundus an Informationen zur Historie einer Softwareanwendung.

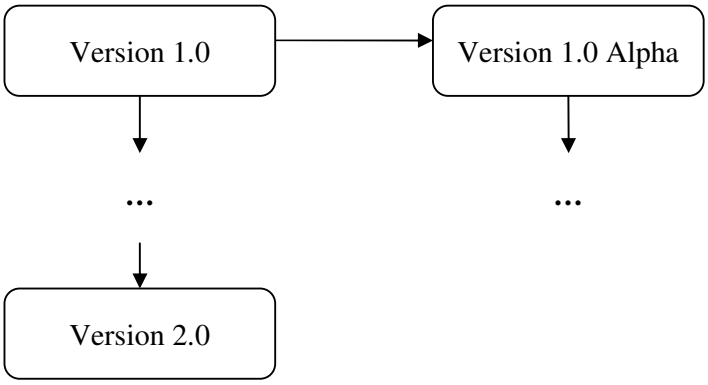

Abbildung 14: Entstehung einer Versionsverzweigung

[138] siehe http://www.tgs-cms.org/index.php

[139] siehe Kapitel 4.3.3 „Infrastrukturkomponenten“

Datensammlung

Die Sammlung der Daten aus dem CVS bzw. dem SVN-System erfolgt im Wesentlichen in drei Schritten.

Zunächst wird im ersten Schritt mit Hilfe eines CVS/SVN-Klienten[140] der Quellcode der Software der Open Source Projekte heruntergeladen. Dabei ist auf folgende Punkte zu achten:

Im Zuge der Entwicklung einer Anwendung im CVS oder SVN sind unterschiedliche Begriffe auseinander zu halten. Der Stamm *(engl. „Trunk“)* einer Software umfasst alle Dateien, die zum Hauptentwicklungszweig einer Anwendung gehören. Entwickeln sich darüber hinaus in der Folge von Verzweigungen unterschiedliche Versionen eines Programms, so werden diese in so genannten *Branches* verwaltet. Da jede einzelne Datei unterschiedliche Versionen, so genannte *Revisions*, besitzen kann, aber nicht jede Version einer Datei mit jeder Version einer anderen Datei zusammenarbeiten kann, hat sich ein System zur Auszeichnung ganzer Module herausgebildet, das so genannte *Tagging*. Dies bedeutet, dass bestimmte Versionen einzelner Dateien, die zueinander kompatibel sind, als ein *Tag*, ein zusammengehöriges Paket ausgezeichnet werden können. Möchte man so beispielsweise die aktuellste Version des Hauptentwicklungszweiges herunterladen, so ist dies die *Head Revision* (aktuellste Version aller Dateien) des Trunks. Im CVS erhält man diese Version durch ein Auschecken der existierenden Dateien ohne eine Revisionsangabe. Im SVN entspricht dies dem Auschecken der Dateien im Verzeichnis „Trunk“[141].

Für die Analyse der sechs WCM-Systeme wird, abgesehen vom Mambo-Projekt, die jeweils aktuellste Version des Hauptentwicklungszweiges der Projektsoftware verwendet. Die grundlegende Annahme zu dieser Vorgehensweise besteht darin, dass dieser Zweig zum einen die Hauptentwicklungsrichtung repräsentiert, zum anderen normalerweise regelmäßig aktualisiert wird, sofern neue Änderungen vorliegen. Da der Hauptentwicklungszweig im Mambo-Projekt allerdings seit etwa 1 Jahr nicht mehr geändert wurde, dient hier die aktuellste Verzweigung (Version 4.6) als Untersuchungsgegenstand.

Eine weitere Fokussierung bildet die Limitierung der Untersuchungen auf den Anwendungskern der untersuchten Programme. Aufgrund des hohen Funktionsumfangs von WCM-Systemen sind diese meist in Form einer Plug-In-Architektur realisiert. Im

---

[140] Anwendung zum Zugriff auf ein CVS/SVN-Serversystem, siehe http://www.nongnu.org/cvs/ bzw. http://subversion.tigris.org

[141] Dies muss nicht der Fall sein, allerdings hat es sich zu einer Konvention entwickelt, die Verzeichnisstruktur im SVN entsprechend der Unterscheidung von Trunk, Branches und Tags anzulegen.

Zuge dessen werden lediglich die Basisfunktionen im Kern einer Anwendung implementiert. Die übrigen Funktionen werden dann über spezifische Schnittstellen mit dem Kern verbunden. Entsprechende Angaben zur Identifizierung der tatsächlichen Kerndateien befinden sich, soweit vorhanden, auf den Webseiten der Open Source Projekte. Der Grund für diese zusätzliche Einschränkung besteht in dem Ziel, mit den CVS- bzw. SVN-Daten vor allem Aussagen über den vorhandenen Kern der Entwickler in den Open Source Projekten treffen zu wollen. Die Schwierigkeit besteht allerdings darin, dass unterschiedliche Open Source Projekte möglicherweise einen unterschiedlichen Modularisierungsgrad der Software besitzen, sodass die erzielten Resultate allein deshalb abweichen könnten.

Der zweite Schritt zur Sammlung der CVS- und SVN-Daten besteht in der Generierung von so genannten *„Log-Dateien“* zur Quellcodebasis. Log-Dateien sind spezielle Protokolldateien, in denen im Allgemeinen Prozessaktivitäten aufgezeichnet werden. Im Zusammenhang mit Versionsverwaltungssystemen beinhalten Log-Dateien eine Zusammenfassung der Entstehungshistorie des Softwarequellcodes. Generiert werden die Log-Dateien mit Hilfe eines speziellen Befehls der CVS/SVN-Klientanwendung:

```
cvs log > Name_der_Logdatei.log
```

bzw.

```
svn log -xml -v > Name_der_Logdatei.log
```

Im dritten Schritt werden die generierten Log-Dateien schließlich ausgewertet. Dies geschieht mit Hilfe zweier Anwendungen zur Erstellung von Statistiken zu CVS- bzw. SVN-Log-Dateien (*StatCVS*[142] bzw. *StatSVN*[143]). Das Ergebnis ist eine Kollektion von HTML-Seiten mit statistischen und graphischen Informationen zur Historie des korrespondierenden Quellcodes.

6.1.1 Alter der Open Source Projekte (1)

Das Alter der Projekte dient als zusätzliche Kontrollgröße zur Einschätzung der anderen identifizierten Metriken zur Evaluation des Zielfaktors „Zukunftssicherheit“. Das Alter eines Open Source Projektes kann dabei insofern mit Hilfe der CVS/SVN-Daten ermittelt, als dass die Informationen in den Log-Dateien der Versionsverwaltungssysteme auch das Datum von eingehenden Beiträgen beinhalten. Dementsprechend ist auch der Tag des ersten *„Commits“* [144] ersichtlich. Da die

[142] siehe http://statcvs.sourceforge.net/

[143] siehe http://www.statsvn.org

[144] Der Begriff „Commit“ bezeichnet den Vorgang des Eincheckens von Quellcode in ein Versionsverwaltungssystem

Einschätzung des Projektalters in dieser Form allerdings an die Annahmen gebunden ist, dass zum einen alle Versionen einer Software auch in das entsprechende System integriert worden sind und zum anderen der Tag des ersten „Commits“ gleichzeitig der Gründungstag des jeweiligen Open Source Projektes ist, sollten die so erhaltenen Werte mit gegebenenfalls vorhandenen Angaben auf der Webseite der Projekte verglichen werden.

6.1.2 Größe der Open Source Projekte

Die Größe eines Open Source Projektes kann an zwei Aspekten festgemacht werden. Dies ist zum einen die Größe der Software, gemessen in der Anzahl der Codezeilen (LOC), zum anderen auch die Größe der Projektcommunity.

Die erste Metrik kann dabei direkt den Ergebnissen der Analysen der Log-Dateien entnommen werden, da sowohl StatCVS wie auch StatSVN die Werte für die im Laufe der Zeit hinzugefügten und gelöschten Codezeilen im Zuge des Parsens der Log-Dateien direkt aufsummieren, sodass als Ergebnis unter anderem die absolute Größe der Anwendung berechnet wird. Problematisch in diesem Zusammenhang ist allerdings die Frage nach der Vergleichbarkeit der erhaltenen Werte. Dies liegt unter anderem daran, dass unterschiedliche Anwendungen wie erwähnt einen unterschiedlichen Modalisierungsgrad aufweisen können. Im Zuge dessen variiert der Umfang der Funktionalitäten, die fest in den Anwendungskern integriert sind. So wird die Umsetzung einer Plug-In-Architektur im Allgemeinen mit dem Ziel verbunden sein, den Umfang des Anwendungskerns so gering wie möglich zu halten. Dies trifft jedoch nicht für alle Anwendungen gleichermaßen zu.

Die zweite Metrik kann dagegen nur teilweise direkt aus den CVS/SVN-Daten ermittelt werden. Zwar beinhalten die Log-Dateien der Versionsverwaltungssysteme Autorinformationen zu den Entwicklern, die die entsprechenden Codefragmente eingecheckt haben. Allerdings ist die Zahl der im CVS oder SVN-System geführten Entwickler nicht unbedingt gleichzusetzen mit der Gesamtentwicklerzahl des Open Source Projektes. In diesem Zusammenhang spielt vor allem das bereits erwähnte Argument des Zentralisierungsgrades der Entwicklung [145] eine wichtige Rolle. Allerdings gibt die Anzahl der im Versionsverwaltungssystem involvierten Personen in der Regel einen guten Überblick über die Größe des Kreises der Kernentwickler eines Open Source Projektes, vertraut man dem empirisch für einige erfolgreiche Projekte ermittelten Zusammenhang, der besagt, dass die insgesamt aktive Entwicklercommunity etwa um eine Größenordnung größer ist als die Zahl der Kernentwickler selbst [Mockus2002].

---

[145] siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

6.1.3 Aktivität der Entwickler

Neben der Größe der Community ist vor allem auch die Aktivität der Entwickler ein entscheidendes Kriterium für die Vitalität eines Open Source Projektes. Ähnlich wie im Fall der Projektgröße lässt sich auch die Aktivität der Entwickler in mehrere auseinander gehende Aspekte unterteilen.

Erstens beeinflusst die Regelmäßigkeit, mit der die beteiligten Personen zur Weiterentwicklung der Software beitragen, die Entwicklung des Projektes. Zweitens spielt zur Einschätzung der Produktivität der Entwickler auch die Größe ihrer geleisteten Beiträge eine wesentliche Rolle. Drittens schließlich ist mit den zwei genannten Aspekten der Entwickleraktivität darüber hinaus die Entwicklung der Gesamtgröße der Software im untersuchten Zeitraum eng verbunden. Da diese unmittelbar von den beiden erstgenannten Metriken abhängt, wird sie im Rahmen der Arbeit ebenfalls zu den Aktivitätsmetriken gerechnet.

Alle drei beschriebenen Größen können aus den CVS/SVN-Log-Dateien ermittelt werden. Betrachtet werden dabei im Rahmen der Arbeit jeweils die letzten 3, 6 und 12 Monate.

6.2 Umfrage an die Projektadministratoren

Der Entwicklungsprozess in Open Source Projekten wird in der Regel von einer kleinen Gruppe von Entwicklern geführt und bestimmt.[143] Diese Gruppe von Kernentwicklern besitzt verbunden mit ihrer starken Involvierung in die Aktivitäten zur Erstellung der Projektsoftware das größte Wissen sowohl über die Software selbst wie auch über die Strukturen im jeweiligen Projekt. Innerhalb dieser Gruppe existieren darüber hinaus ein oder mehrere Personen, denen neben den anstehenden Codier- und Implementierungsaufgaben auch administrative Funktionen zukommen.

Diese so genannten Projektadministratoren bilden aufgrund ihrer umfangreichen Kenntnisse über die Hintergründe in Open Source Projekten einen reichhaltigen Fundus an Informationen zu den Besonderheiten in und um die Projekte selbst. Angaben dazu, welche Entwickler eines Open Source Projektes zudem administrative Verantwortungen besitzen, finden sich zum einen häufig auf den Webseiten von Open Source Projekten. Zum anderen werden die Personen auch auf Open Source Portalen wie Freshmeat oder Sourceforge geführt.

Darüber hinaus besitzen Open Source Projekte aufgrund der bestehenden sozialen Hierarchien[146] häufig einen impliziten Prozess zur Handlung entsprechend der eigenen

[146] siehe Kapitel 4.4.1 „Soziale Rahmenbedingungen in Open Source Projekten“

Kompetenzen, sodass in der Regel lediglich Entwickler auf Fragen antworten, die auch das entsprechende Wissen besitzen.

Bestehende Fragen sollten dabei aufgrund der hohen Auslastung der Projektadministratoren normalerweise über die Mailinglisten gestellt werden. Im Rahmen der Arbeit erfolgte die Kontaktierung der Administratoren allerdings direkt durch eine Email, die sowohl die Fragen selbst als auch einen Link auf eine vorgefertigte Online-Umfrage beinhaltete. Vor dem Hintergrund, dass Offenheit einen der Basiswerte der Open Source Bewegung bildet und der Entwicklungsprozess zudem sowieso transparent gestaltet ist, wird im Rahmen der Arbeit davon ausgegangen, dass die Einschätzungen der Projektadministratoren einen realistischen Überblick über die Geschehnisse in Open Source Projekten geben. Der Fragenkatalog zur Umfrage an die Administratoren der sechs ausgewählten Open Source Projekte befindet sich im Anhang.[147]

6.2.1 Allgemeine Vorgehensweise

Inhaltlich besteht die Umfrage aus insgesamt 15 Fragen, wobei die Beantwortung der Fragen entweder direkt in der gesendeten Email oder in Form einer angefertigten Online-Umfrage erfolgen konnte. Die 15 Fragen sind dabei in vier inhaltliche Abschnitte unterteilt, die jeweils auf die Erfassung eines Aspektes der Zerlegung des Zielfaktors „Zukunftssicherheit“ ausgerichtet sind. Eine Besonderheit bildet dabei die Erfassung des Zielfaktors selbst, der im letzten der vier Abschnitte in Form dreier Fragen operationalisiert wird.

Angeschrieben wurden diejenigen Entwickler der Open Source Projekte, die entweder auf einem der beiden Open Source Portale Sourceforge bzw. Freshmeat als Projektadministratoren bzw. Projektmanager angegeben oder auf der Webseite des jeweiligen Projektes als Mitglied des Kernentwicklerteams ausgeschrieben waren.

6.2.2 Kommerzielle Ausrichtung (1)

Die ersten drei Fragen beziehen sich auf die kommerzielle Ausrichtung der Open Source Projekte. Dies bedeutet zunächst einmal die Frage, welche Einnahmequellen, wenn vorhanden, das jeweilige Projekt überhaupt besitzt. Die Beantwortung dieser Frage ermöglicht eine Einschätzung, inwieweit die Entwicklung der Projektsoftware einen kommerziellen Hintergrund besitzt. Typische Einnahmequellen auch für nicht-kommerzielle Projekte sind Spenden von gemeinnützigen Organisationen oder Mitgliedern der Projekte sowie kleinere Werbeeinnahmen durch Verlinkungen zu

[147] siehe Anhang D.1

anderen Webseiten.[148] Dagegen deuten das Anbieten von Dienstleistungen zur Software oder der Verkauf kompletter Distributionen bereits auf eine stärker unternehmerisch geprägte Communitystruktur hin. In diesem Zusammenhang ermittelt die zweite Frage, wie viele Entwickler tatsächlich von den vorhandenen Einkünften profitieren. Zwar sind in der Regel nur die aktivsten Entwickler eines Open Source Projektes an den Projekteinnahmen beteiligt, was im Zuge der starken Verteilung der Entwicklung und der oftmals unübersichtlichen Situation in Bezug auf die genauen Rechte am Quellcode der Software bereits aus organisatorischen Gründen die einfachste Lösung ist.[149] Dennoch ist eine stärkere Demokratisierung der Projekteinkünfte bzw. eine exaktere Aufteilung entsprechend der geleisteten Beiträge zumindest denkbar. Vor diesem Hintergrund kann der Anschluss des Projektes an eine gemeinnützige Organisation wie die Apache Software Foundation (ASF) bzw. die Organisation der Entwickler innerhalb einer eigenen Unternehmens helfen, einen stärkeren Formalisierungsgrad und damit verbunden eine größere Transparenz zu erreichen.[150]

Die dritte Frage schließlich, die den kommerziellen Hintergrund der Open Source Projekte beleuchten soll, bezieht sich auf die Anzahl der Unternehmen im Umfeld der Projekte, die Support zur Projektsoftware anbieten. Dies kann sowohl die Organisation der Projektentwickler selbst innerhalb eines Unternehmens als auch externe Unternehmen beinhalten und stellt für Endnutzer der Software einen wichtigen Aspekt im Zusammenhang mit der Zukunftssicherheit einer Investition in eine Softwareanwendung dar.[151]

6.2.3 Ausformung des Schulungsprozesses für Entwickler und Benutzer

Genau wie die Umfragesektion zur Erfassung der kommerziellen Ausrichtung der sechs Open Source Projekte besteht auch der Abschnitt zur Operationalisierung des im Projekt etablierten Schulungsprozesses für Entwickler und Benutzer der Software aus drei Fragen. Dabei wird zunächst der Umfang des erforderlichen Wissens abgesteckt, das Entwickler besitzen müssen, um die zentrale Funktion eines Kernentwicklers ausüben zu können. Diese Metrik bildet einerseits ein Maß für die Komplexität der Anwendung bzw. des Prozesses zu ihrer Entwicklung, andererseits liefert sie darüber hinaus die Möglichkeit, den Grad der Abhängigkeit von den bestehenden Kernentwicklern abzuschätzen. Ist der notwendige Aufwand für neue bzw. Co-Entwickler gering, so ist es leichter möglich, den Projektaustritt eines Kernentwicklers gegebenenfalls zu

---

[148] siehe z.B. http://argouml.tigris.org/servlets/ReadMsg?list=dev&msgNo=20303 (ArgoUML-Projekt)

[149] siehe Kapitel 4.4.2 „Wirtschaftliche Rahmenbedingungen in Open Source Projekten“

[150] siehe Kapitel 6.3.2 „Kommerzielle Ausrichtung (2)“

[151] siehe Kapitel 4.4.6 „Anforderungen der Unternehmen an Open Source Projekte“

kompensieren. Im umgekehrten Fall wird die Kompensation rückläufiger Kernentwicklerzahlen dagegen schwieriger sein.

Die zweite Frage bezieht sich dann auf die im Projekt etablierten Prozesse, Aktivitäten und Strukturen, die Co-Entwickler dabei unterstützen, das entsprechende Wissen zu erlangen. Beispiele für solche unterstützenden Maßnahmen sind einerseits relativ statische Strukturen wie die Spezifikation einer API bzw. der Softwarearchitektur oder die schlichte Existenz eines Bug-Tracking-Systems zur Organisation der Aktivitäten zur Beseitigung bestehender Programmdefekte. Andererseits umfasst ein funktionierender Schulungsprozess vor allem auch dynamische Prozesse wie Interaktionen auf den Mailinglisten oder das Auskommentieren von Codierbeiträgen neuer Entwickler durch die Kernentwickler.

Besonders der zweite Aspekt fordert dabei eine intensive Beteiligung der Kernentwickler an den entsprechenden Aktivitäten. Dementsprechend beschäftigt sich die dritte Frage des Abschnittes mit der Ermittlung des Zeitaufwandes, den die Projektadministratoren investieren, um neue bzw. Co-Entwickler des Open Source Projektes bei ihren Lernprozessen zu unterstützen.

6.2.4 Projektideologie und Projektziele (1)

Der dritte wesentliche Abschnitt der Umfrage an die Projektadministratoren beschäftigt sich mit der Erfassung einer gegebenenfalls vorhandenen Kernideologie in den Open Source Projekten sowie der langfristigen Ziele, die die Projektadministratoren bzw. die gesamte Community formulieren. Entsprechend der Differenzierung des Begriffs „Kernideologie“ nach Collins und Porras[152] in Kernwerte sowie eine weitergehende Daseinsberechtigung ist auch die Befragung der Projektadministratoren zu diesem Punkt in die zwei Teile gegliedert, so dass der gesamte Abschnitt insgesamt die Messung dreier Teilaspekte beinhaltet. Dabei erfolgt für den Fall der Kernwerte sowie der Projektziele zunächst die Frage nach der Existenz der jeweiligen Aspekte. Im zweiten Schritt sollen die Projektadministratoren dann die entsprechenden Instanzen benennen bzw. begründen, warum das Projekt keine Instanzen der jeweiligen Aspekte trägt. Dies soll zum einen eine qualitative Einordnung der vorhandenen Werte, Ziele und Absichten ermöglichen, zum anderen soll im negativen Fall ermittelt werden, in wieweit sich die Administratoren der jeweiligen Open Source Projekte überhaupt mit dem Thema auseinandergesetzt haben.

Zusätzlich ist es zur Einordnung der Open Source Projekte in das Rahmenwerk von Collins und Porras erforderlich, dass eine vorhandene Kernideologie nicht nur in den Köpfen der verantwortlichen Personen vorhanden, sondern vor allem auch formuliert

[152] siehe Kapitel 5.3.1 „Der Zielfaktor „Zukunftssicherheit“ “

und in spezifischen Mechanismen manifestiert ist. Dies wird später im Rahmen der Untersuchung der Webseiten der Open Source Projekte überprüft.[153]

6.2.5 Zukunftssicherheit

Den letzten Abschnitt innerhalb der Umfrage an die Projektadministratoren der ausgewählten Open Source Projekte bildet die Erfassung des eigentlichen Zielfaktors selbst. Da eine genaue Messung der Zukunftssicherheit von Open Source Projekten eine Langzeitstudie der Entwicklungen in den entsprechenden Projekten bedingen würde und diese im Rahmen der Arbeit nicht möglich ist, erfolgt die Messung des Zielfaktors auf der Grundlage der Einschätzungen der Projektadministratoren. Da sie die Personen darstellen, die am tiefsten in die Prozesse und Strukturen involviert sind und von denen darüber hinaus die Weiterentwicklung der Software im Projekt am stärksten abhängt, sind auch sie es, die die größte Kompetenz in Bezug auf das zukünftige Gesicht der Projekte besitzen. Vor dem Hintergrund, dass viele der Kernentwickler in vergleichsweise großen Open Source Projekten bereits mehrjährige berufliche Erfahrungen im Bereich der Softwareentwicklung besitzen und Transparenz und Offenheit innerhalb der Open Source Bewegung wesentliche Tugenden sind, verbindet sich mit den Einschätzungen der Administratoren der Open Source Projekte innerhalb der Arbeit die Annahme, dass diese realistisch und begründet sind.

Gemessen wird die Zukunftssicherheit der Open Source Projekte dabei an Hand von drei Fragen. Dies beinhaltet erstens eine Frage nach den möglichen Problemen, die die Weiterentwicklung der Software im Projekt behindern könnten. Zweitens soll die Entwicklung der zukünftigen Communitygröße abgeschätzt werden. Drittens schließlich soll analog zur Abschätzung der Entwicklung der zukünftigen Communitygröße eine Abschätzung der zukünftigen Qualität der jeweiligen Softwareanwendung erfolgen.

6.3 Die Webseite

Die Webseite von Open Source Projekten bildet eine der wesentlichsten Präsentations- und Kommunikationsschnittstellen zu Entwicklern und Endnutzern. Die meisten Open Source Projekte kommen dort der mit dem Ansatz der offenen Softwareentwicklung verbundenen Forderung nach Offenheit und Transparenz nach. Dies geschieht durch detaillierte Angaben und Beschreibungen des Projektumfeldes, der Projektcommunity, etablierter sozialer Regeln oder auch der Entwicklungsziele und Entwicklungshintergründe. Demzufolge stellt die Webseite auch für eine Analyse von Open Source Projekten eine wichtige Daten- und Informationsquelle dar. Dies gilt insbesondere vor dem Hintergrund, dass im Zuge der zunehmenden

[153] siehe Kapitel 6.3.3 „Projektideologie und Projektziele (2)“

Professionalisierung des Feldes[154] die Webseite die wesentlichste Schnittstelle zu vorhandenen Kunden bildet.

Im Rahmen einer Datenerhebung besitzt sie zudem den Vorteil, dass sie leicht auszuwerten ist. Allerdings können mit Hilfe der Webseiten zum einen die im Projekt ablaufenden Prozesse zur Entwicklung der Software und Schulung der Entwickler nicht detailliert erfasst werden. Zum anderen enthalten die Webseiten gelegentlich veraltete Daten, weshalb sie im Rahmen der Arbeit vor allem zur Kontrolle der beiden zuvor genannten Datenquellen verwendet werden. Die damit gewonnene Redundanz ist dabei insbesondere für die Punkte Projektideologie und Projektziele wichtig.

6.3.1 Alter der Open Source Projekte (2)

Das Alter der Open Source Projekte dient, wie erwähnt, der besseren Einordnung der anderen identifizierten Metriken zu Open Source Projekten. Da die Ermittlung des Projektalters nur aufgrund der Versionsverwaltungssysteme ungenau sein kann[155], wird zusätzlich die Webseite auf diesbezügliche Angaben überprüft.

6.3.2 Kommerzielle Ausrichtung (2)

Nachdem in Bezug auf die kommerzielle Ausrichtung durch die durchgeführte Umfrage an die Administratoren der fünf ausgewählten Open Source Projekte die drei Fragen

- Welche Einnahmequellen besitzt das Projekt?
- Wie viele bezahlte Entwickler besitzt das Projekt?
- Wie viele Unternehmen haben sich im Umfeld des Projektes entwickelt?

geklärt werden, soll die Betrachtung der Webseite der Projekte darüber hinaus einen Einblick vermitteln, in wieweit das jeweilige Open Source Projekt selbst in Form eines Unternehmens organisiert ist. Da sich in Verbindung mit Open Source Projekten mehrere Geschäftsmodelle etabliert haben, die den Ansatz der offenen Softwareentwicklung auch finanziell nutzbar machen[156], ist ein betrieblicher Hintergrund offener Softwareprojekte nicht mehr ungewöhnlich.

Zusätzlich ist es selbst im Fall rein community-orientierter Open Source Projekte interessant, ob die Projekte an eine gemeinnützige Organisation zur Förderung der offenen Softwareentwicklung angeschlossen sind, da dies zum einen eine

---

[154] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

[155] siehe Kapitel 6.1.1 „Alter der Open Source Projekte (1)“

[156] siehe Kapitel 4.4.5 „Ziele eines Einsatzes von Open Source Software in Unternehmen“

Formalisierung der Verteilung der Projekteinkünfte ermöglicht. Zum anderen verringert die Zugehörigkeit zu gemeinnützigen Organisationen wie der Apache Software Foundation die Abhängigkeit der Projekte von einzelnen Personen, da von ihnen häufig eine Vielzahl von Open Source Projekten gemeinschaftlich verwaltet wird.

Sowohl die Angaben zur betrieblichen Ausrichtung als auch zu Organisationen im Umfeld der Open Source Projekte sind in der Regel auf den Webseiten zu finden.

6.3.3 Projektideologie und Projektziele (2)

Dem Rahmenwerk von Collins und Porras folgend ist der Erfolg einer Kernideologie in Unternehmen und übertragen auch für Open Source Projekte gebunden an eine Manifestierung der vorhandenen Werte und Normen in konkrete Mechanismen. Während die durchgeführte Befragung der Projektadministratoren in Bezug auf den Aspekt einer Kernideologie im Wesentlichen darauf zielt, zu erfahren, ob der Gedanke in den entsprechenden Projekten überhaupt aufgegriffen wird, soll nun die Untersuchung der Webseite dazu dienen, konkrete Instanzen bzw. Manifestierungen zu entdecken. Da die Webseite wie erwähnt die zentrale Präsentationsschnittstelle der Open Source Projekte zu den Benutzern und Entwicklern der Software ist, müssten entsprechende Angaben dort zu finden sein.

7 Informationsauswertung

Nach der Auswahl der Datenquellen und der Durchführung der Datenerhebung in der Phase der Informationsgewinnung erfolgt in der Phase der Informationsauswertung schließlich die Analyse und Interpretation der gewonnenen Daten hinsichtlich der entwickelten Zerlegung des Zielfaktors „Zukunftssicherheit“. Dafür werden zunächst die im Rahmen der GQM-Zerlegung erhobenen Fragestellungen für die sechs ausgewählten WCM-Systeme beantwortet. Anschließend erfolgt dann die Interpretation der Ergebnisse in Bezug auf die vier angenommenen Teilaspekte der Zukunftssicherheit von Open Source Projekten.

7.1 Ergebnisse der Datenerhebung

Im Folgenden werden die Resultate der Datenerhebung anhand der entwickelten Fragestellungen [157] zu den vier angenommenen Teilaspekten des Zielfaktors „Zukunftssicherheit“ von Open Source Projekten präsentiert. Dabei erfolgt die Präsentation der Ergebnisse für jedes Projekt einzeln und in jeweils zwei Abschnitte unterteilt.

Die ersten 12 Fragen entsprechen den Fragestellungen der zu Open Source Projekten entwickelten GQM-Zerlegung und dienen der Erfassung der kommerziellen Ausrichtung, der Art und des Umfanges der im Projekt etablierten Mechanismen zur Unterstützung neuer und bestehender Entwickler, der Größe bzw. der Aktivität der Projektcommunity und der möglicherweise im Projekt etablierten Kernideologie sowie ambitionierter Ziele nach Collins und Porras.

Die Fragen 13 bis 16 dagegen wurden den beteiligten Projektadministratoren im Zuge der durchgeführten Umfrage zusätzlich gestellt und dienen der direkten Erfassung der Zukunftssicherheit der jeweiligen Open Source Projekte, sodass eine Validierung der vorgeschlagenen Zerlegung möglich wird.

7.1.1 Drupal

1) Welche Einnahmequellen besitzt das OSP?

Die wesentlichste Einnahmequelle von Drupal besteht in Spendengeldern von Mitgliedern des Open Source Projektes oder Unternehmen in seinem Umfeld. Die Spendengelder werden genutzt, um zum einen die Infrastruktur für die verteilte

[157] siehe Kapitel 5.3.1 „Der Zielfaktor ‚Zukunftssicherheit’ “

Entwicklung der Projektsoftware zu finanzieren. Zum anderen werden damit auch verdiente Entwickler der Projektsoftware für ihre Arbeiten entlohnt.

Darüber hinaus bietet die Webseite des Open Source Projektes sowohl externen Unternehmen wie auch Einzelpersonen die Möglichkeit, professionellen Support zur Software oder darauf aufbauenden Anwendungen anzubieten. Voraussetzung für eine Nennung auf den entsprechenden Beraterlisten ist allerdings, dass die aufgeführten Personen oder Unternehmen selbst Beiträge zum Projekt oder zur Projektsoftware geleistet haben. Eine mögliche Form der Beteiligung ist dabei neben dem Einsenden von Codierbeiträgen auch das Leisten finanzieller Unterstützung. Auf diese Weise profitiert das Drupal-Projekt einerseits durch die erbrachten Leistungen, andererseits bietet sich für die Entwickler des Projektes die Möglichkeit, durch die Erbringung von Dienstleistungen ein Einkommen aus ihrem Wissen über die Software zu generieren.

Eine Anlehnung von Drupal an ein kommerzielles Unternehmen in Verbindung mit eigenen Serviceleistungen erfolgt allerdings nicht. Das Open Source Projekt selbst ist nicht-kommerziell orientiert. *[Administrator][Webseite]*

2) Wie viele bezahlte Entwickler besitzt das Open Source Projekt?

Drupal besitzt mehr als 10 bezahlte Entwickler. *[Administrator]*

Diese werden einerseits durch Spendengelder finanziert, andererseits bieten die Entwickler selbst professionellen Support zur Software an. Vor dem Hintergrund, dass Drupal selbst nicht-kommerziell ist, ist allerdings anzuzweifeln, dass zumindest die Verteilung der Spendengelder mit einem vollzeitlichen Gehalt zu vergleichen ist. *[Webseite]*

3) Wie viele Unternehmen bieten Support für die Software des OSPs an?

Seit dem Jahr 2006 wird Drupal von der *Drupal Association*, einer gemeinnützigen Organisation im Umfeld des Open Source Projektes, betreut. Diese bietet zwar keinen eigenen Softwaresupport an, zeichnet dafür aber Unternehmen im Umfeld des Projektes aus, die dies tun. Den Angaben auf der Webseite folgend sind dies etwa 20 Unternehmen. *[Administrator][Webseite]*

Diese Zahl deckt sich auch mit den Ergebnissen der Administratorbefragung. *[Administrator]*

4) Wie lange benötigt ein neuer Entwickler, um das Wissen eines Kernentwicklers zu erlangen?

Ein neu hinzukommender Entwickler benötigt etwa 1-2 Jahre, um das Wissen eines Kernentwicklers zu erlangen. *[Administrator]*

5) Welche Mechanismen sind in dem OSP etabliert, um neue Entwickler dabei zu unterstützen, dieses Wissen zu erlangen?

Die Strukturen, die zur Unterstützung neuer und bestehender Entwickler im Drupal-Projekt etabliert sind, umfassen ein Bug-Tracking-System, das es Entwicklern ermöglicht, einen Überblick über die bestehenden Aufgaben zu erlangen, sowie eine Beschreibung der API zur Erläuterung der Programmierschnittstellen. Darüber hinaus beantwortet der Projektadministrator sowohl technische Fragen wie auch Fragen hinsichtlich der Entwicklungsrichtung in den Mailinglisten. *[Administrator]*

6) Wie viel Zeit verbringen die Kernentwickler des OSP damit, andere Entwickler zu unterstützen?

Der Projektadministrator verbringt etwa 3-5 Stunden pro Woche mit der Unterstützung von anderen Entwicklern und Benutzern der Software. *[Administrator]*

7) Welche Kernwerte besitzt das OSP und in welcher Form werden sie formuliert?

Die Kernwerte von Drupal bestehen in der offenen Softwareentwicklung selbst sowie in der Anlehnung der Software an bestehende Webstandards. *[Administrator]*

Zudem formuliert Drupal insgesamt sieben Kernwerte auf der Webseite, an denen sich die Entwicklung der Software innerhalb des Open Source Projektes orientiert. Diese werden auf einer getrennten Seite innerhalb des Bereiches *„About Drupal“* genannt und im Einzelnen erklärt. Die sieben Kernwerte sind

- Modularität und Erweiterbarkeit der Software,
- hohe Codierqualität,
- standardbasierte Entwicklung,
- geringer Ressourcenverbrauch,
- offene Softwareentwicklung,
- einfache Bedienbarkeit
- und Zusammenarbeit. *[Webseite]*

8) Welche Daseinsberechtigung über finanzielle Aspekte hinaus besitzt das OSP?

Die Daseinsberechtigung von Drupal besteht in dem Bedarf an einem erstklassigen Content Management System. *[Administrator]*

Zudem formuliert das Projekt auf seiner Webseite eine Mission. Diese Mission besteht in der Erweiterung des Mediums „Internet“ zur Gewährleistung eines gemeinschaftlichen Informationsaustausches. *[Webseite]*

9) Welche langfristigen herausfordernden Ziele besitzt das OSP und in welcher Form werden sie formuliert?

Ziel von Drupal ist die Entwicklung des besten WCM-Systems weltweit. *[Administrator]*

Demgegenüber existiert allerdings existiert keine offizielle Roadmap, die dieses Ziel in konkrete, durchführbare Maßnahmen und Vorhaben gliedert. *[Webseite]*

10) Wie groß ist die Projektcommunity?

Es haben insgesamt 8 Entwickler die Entwicklung der Projektsoftware im CVS getragen. Diese Zahl bezieht sich dabei allerdings nur auf die Entwicklung des Anwendungskerns. Die darüber hinaus entwickelten Plug-Ins sind nicht berücksichtigt. Der Entwicklungszeitraum umfasst dabei insgesamt 7 Jahre. *[CVS]*

11) Wie groß ist die Projektsoftware?

Die CVS-Head-Revision des Drupal-Kerns (Stand: 13.04.2007) umfasst insgesamt 59.131 Lines of Code. *[CVS]*

Dies ist im Vergleich zu den restlichen untersuchten Open Source Projekten vergleichsweise wenig. Allerdings belegen die Angaben auf der Webseite des Projektes, dass das WCM-System sehr modular aufgebaut ist, wodurch ein Großteil der Funktionalität in den Bereich der Plug-Ins verlagert wird. *[Webseite]*

12) Wie aktiv ist die Projektcommunity?

Die Entwicklung der Software von Drupal wurde von insgesamt 8 Entwicklern getragen, wobei zwei der Entwickler dabei fast 90% des gesamten Quellcodes implementiert haben. Insgesamt waren in den letzten 3, 6 und 12 Monaten 3, 4 bzw. 5 Entwickler im CVS aktiv. Der größte Teil der Aktivitäten fällt aber auf die zwei aktivsten Entwickler zurück.

Betrachtet man den Verlauf der Entwicklung im CVS, so fällt vor allem die vergleichsweise hohe Zahl an durchgeführten Commits in den letzten 12 Monaten auf. Demgegenüber steht allerdings eine relativ niedrige Zahl an in diesem Zeitraum hinzugefügten Quellcodezeilen.

Folglich nehmen die Entwickler zwar einerseits häufig Änderungen an der Software vor, andererseits sind die Änderungen aber nicht sehr umfangreich. Dies deutet darauf hin, dass es sich vorwiegend um verbessernde im Gegensatz zu erweiternden Änderungen handelt. Da sich diese Entwicklung über den gesamten Untersuchungszeitraum von 12 Monaten hinweg erstreckt, weist dies möglicherweise darauf hin, dass der Kern der Anwendung einen gewissen Reifegrad erreicht hat und es sich nicht um intensive Veränderungen im Vorfeld oder in der Folge eines neuen Release der Software handelt. *[CVS]*

13) Wie entwickelt sich die Größe der Projektcommunity in fünf Jahren?

Die Projektcommunity wird in fünf Jahren wesentlich größer sein. *[Administrator]*

14) Wie entwickelt sich die Qualität der Software in fünf Jahren?

Die Software wird in fünf Jahren „mature“ sein. *[Administrator]*

15) Welche Probleme können die zukünftige Weiterentwicklung der Projektsoftware behindern?

Die zukünftige Weiterentwicklung der Software von Drupal kann durch Probleme in der Ausbildung sowie durch nicht ausreichende Fähigkeiten neuer Entwickler beeinträchtigt werden. *[Administrator]*

16) Was wird den Erfolg des Open Source Projektes ausmachen?

Ausschlaggebend für den zukünftigen Erfolg von Drupal ist die starke Führung im Open Source Projekt. *[Administrator]*

7.1.2 Magnolia

1) Welche Einnahmequellen besitzt das OSP?

Im Gegensatz zu Drupal ist Magnolia sehr stark kommerziell ausgerichtet. Die Projektsoftware wurde zunächst proprietär entwickelt und erst im Jahr 2003 offen lizenziert. Seit 2006 bildet die *Magnolia International Ltd.* die zentrale Organisation im Hinblick auf die Weiterentwicklung und Vermarktung der Software des Open Source Projektes. Das Unternehmen umfasst dabei unter anderem die Kernentwickler des Projektes und besitzt zudem die Rechte am Markennamen „Magnolia“. Das Unternehmensportfolio basiert auf zwei wesentlichen Pfeilern.

Der Erste besteht analog zu Drupal im Anbieten bestimmter Dienstleistungen. Diese werden allerdings direkt von der Magnolia International Ltd. angeboten und nicht wie im Fall von Drupal von Unternehmen und Einzelpersonen im Projektumfeld. Die angebotenen Leistungen umfassen dabei Schulungen zur Projektsoftware, technische Unterstützung sowie Beratungsleistungen zur Erstellung einer Softwarestrategie oder zur Durchführung einer Migration auf das Magnolia WCM-System.

Der zweite Pfeiler besteht im Dual Licensing. Magnolia International Ltd. vertreibt dazu zwei Versionen des Content Management Systems. Die so genannte *Community Edition* wird frei unter der LGPL verbreitet, während die *Enterprise Edition* in Verbindung mit zusätzlichen Funktionen und unbegrenztem Support proprietär vertrieben wird.

Neben den beiden genannten Stützpfeilern zählen die Zertifizierung von Partnerbetrieben sowie auch Werbeeinkünfte zu den Einnahmequellen des Unternehmens. *[Administratoren][Webseite]*

2) Wie viele bezahlte Entwickler besitzt das Open Source Projekt?

Das Magnolia-Projekt besitzt etwa 3-5 bezahlte Softwareentwickler. *[Administratoren]*

Diese sind innerhalb der Magnolia International Ltd. organisiert und beziehen ihr Einkommen durch den Verkauf von Dienstleistungen bzw. lizenzgebührenpflichtigen Programmerweiterungen. *[Webseite]*

3) Wie viele Unternehmen bieten Support für die Software des OSPs an?

Der wesentlichste Dienstleistungslieferant ist im Hinblick auf den Support zum Magnolia CMS die Magnolia International Ltd. Darüber hinaus sind 13 zertifizierte weltweite Partnerunternehmen der Magnolia International Ltd. auf der Webseite des Unternehmens genannt, die ebenfalls Dienstleistungen zur Software anbieten. *[Administratoren][Webseite]*

4) Wie lange benötigt ein neuer Entwickler, um das Wissen eines Kernentwicklers zu erlangen?

Ein neuer Entwickler benötigt etwa 1-2 Jahre, um das Wissen eines Kernentwicklers zu erlangen. *[Administratoren]*

5) Welche Mechanismen sind in dem OSP etabliert, um neue Entwickler dabei zu unterstützen, dieses Wissen zu erlangen?

Die im Magnolia-Projekt etablierten Strukturen zur Unterstützung neuer und bestehender Entwickler umfassen ein Bug-Tracking-System, die Beschreibung der API sowie Tutorials, die eine Einführung in die Funktionsweise des Systems bieten. Darüber hinaus beantwortet der Projektadministrator Fragen zu technischen Problemen, zur Entwicklungsrichtung sowie zu neuen Codierbeiträgen. *[Administratoren]*

6) Wie viel Zeit verbringen die Kernentwickler des OSP damit, andere Entwickler zu unterstützen?

Die beiden an der Umfrage beteiligten Projektadministratoren von Magnolia investieren wöchentlich etwa 2 bzw. sogar über 10 Stunden. *[Administratoren]*

7) Welche Kernwerte besitzt das OSP und in welcher Form werden sie formuliert?

Magnolia besitzt keine Kernwerte. *[Administrator]*[158]

Allerdings besitzt Magnolia einen eigenständigen Bereich innerhalb der Webseite, in dem das Projekt Basiswerte formuliert und erläutert, die die Grundlage für die

[158] Einer der beiden an der Umfrage beteiligten Projektadministratoren von Magnolia gibt die Fokussierung auf die Benutzbarkeit als Kernwert an. Da der zweite Projektadministrator die Existenz von Kernwerten allerdings verneint, besteht die Annahme, dass es sich zumindest nicht um einen in der Community tief verwurzelten Basiswert handelt

Entwicklung der Software bilden sollen. Diese Werte spiegeln sich in dem Leitsatz des Open Source Projektes *„Simple is beautiful“* wider und repräsentieren analog zu Drupal vor allem technische Grundsätze. Die formulierten Kernwerte sind

- Unterstützung der Kunden während des gesamten Softwarelebenszyklus,
- Einfache Benutzung der Anwendung,
- Erweiterbarkeit der Anwendung,
- Skalierbarkeit der Anwendung,
- Orientierung an Standards
- und Verbindung von Open Source Software mit kommerziellem Support. *[Webseite]*

8) Welche Daseinsberechtigung über finanzielle Aspekte hinaus besitzt das OSP?

Die Daseinsberechtigung von Magnolia über finanzielle Interessen hinaus besteht im Bedarf an einem Java-basierten Content Management System bzw. darin, das meist genutzte ECM-System zu entwickeln. *[Administratoren]*

Darüber hinaus formuliert Magnolia allerdings keine vergleichbaren Aussagen auf der Webseite. *[Webseite]*

9) Welche langfristigen herausfordernden Ziele besitzt das OSP und in welcher Form werden sie formuliert?

Das langfristige Ziel von Magnolia besteht in der Entwicklung des besten und vor allem am leichtesten benutzbaren Content Management System. *[Administratoren]*

Darüber hinaus beinhaltet die Webseite des Projektes eine Roadmap mit Features, die in zukünftigen Versionen der Software integriert werden sollen, ohne jedoch einen genauen Überblick über die Abfolge oder den zeitlichen Rahmen zu geben. *[Webseite]*

10) Wie groß ist die Projektcommunity?

Die Entwicklung der Projektsoftware im SVN-System ist von insgesamt 16 Entwicklern realisiert worden. Im Unterverzeichnis „Magnolia-Core“ waren insgesamt 8 Entwickler aktiv. Der Entwicklungszeitraum im SVN-System umfasst etwa 2,5 Jahre. *[SVN]*

11) Wie groß ist die Projektsoftware?

Insgesamt umfasst der SVN-Trunk (Stand: 13.04.2007) 218.706 Lines of Code. Fraglich ist jedoch, inwieweit es sich lediglich um den Kern der Anwendung handelt oder bestimmte Plug-Ins bereits integriert sind. Zwar sind auf der Webseite keine expliziten Kerndateien ausgeschrieben, allerdings beinhaltet der SVN-Trunk ein

Unterverzeichnis mit diesem Namen, das vergleichbar zum Drupal-Projekt insgesamt 36.582 Lines of Code umfasst. *[Webseite][SVN]*

12) Wie aktiv ist die Projektcommunity?

Die Software von Magnolia wurde von insgesamt 16 Entwicklern realisiert. Von diesen 16 Entwicklern haben drei Entwickler zusammen über 80% des bestehenden Quellcodes implementiert. Zwei weitere Entwickler sind darüber hinaus noch relevant (über 6,9%) an der Entwicklung beteiligt. Von den 16 insgesamt beteiligten Entwicklern waren sechs auch in den letzten drei Monaten aktiv, wobei sich in diesem Zeitraum vor allem drei der fünf aktivsten Entwickler aktiv beteiligt haben.

Betrachtet man den Verlauf der Entwicklung des Quellcodes von Magnolia in den letzten 12 Monaten, so gehören die Entwickler des Projektes sowohl in der Anzahl der durchgeführten Commits als auch in der Zahl der hinzugefügten bzw. geänderten Codezeilen zu den Aktivsten innerhalb der Gruppe der untersuchten Open Source Projekte. Zieht man allerdings lediglich die letzten drei Monate heran, so rangieren sie zumindest in Bezug auf die Anzahl der hinzugefügten Codezeilen auf dem letzten Platz. Dies hängt vermutlich mit der Freigabe einer neuen Magnolia-Version im November 2006 zusammen. Im Zuge dessen wurden zum einen in der Phase unmittelbar vor der Veröffentlichung überwiegend Korrekturmaßnahmen vorgenommen. Zum anderen indizieren die SVN-Daten auch eine schöpferische Pause der Entwickler unmittelbar nach der Freigabe der neuen Version. *[Webseite][SVN]*

13) Wie entwickelt sich die Größe der Projektcommunity in fünf Jahren?

Die Projektcommunity wird etwas bzw. wesentlich größer sein. *[Administratoren]*

14) Wie entwickelt sich die Qualität der Software in fünf Jahren?

Die Softwarequalität höher bzw. die Software „mature“ sein. *[Administratoren]*

15) Welche Probleme können die zukünftige Weiterentwicklung der Projektsoftware behindern?

Probleme, die die zukünftige Entwicklung von Magnolia behindern können, sind das Akquirieren neuer Entwickler und die Finanzierung des Entwicklungsaufwandes. *[Administratoren]*

16) Was wird den Erfolg des Open Source Projektes ausmachen?

Den zukünftigen Erfolg von Magnolia werden die gute Benutzbarkeit sowie der Fokus auf Unternehmen als Zielgruppe ausmachen. *[Administratoren]*

7.1.3 Mambo

1) Welche Einnahmequellen besitzt das OSP?

Mambo besitzt inzwischen keinen eigenen kommerziellen Zweig mehr. Ähnlich wie Drupal begann allerdings auch Mambo zunächst als kommerzielles Softwareprojekt, bevor die Firma *Miro* die Projektsoftware 2001 dann der GPL unterstellte. Heute wird das Projekt von der *Mambo Foundation Inc.*, einer gemeinnützigen Organisation im Umfeld des Open Source Projektes geleitet, die auch die Verwaltung von Spendengeldern übernimmt. Spendengelder sind gleichzeitig auch die wesentlichste Einnahmequelle des Open Source Projektes und stammen vor allem von vier Unternehmen, die explizit auf der Webseite der Organisation genannt werden.

Darüber hinaus dienen dem Projekt Beiträge für die Mitgliedschaft von externen Unternehmen in der Mambo Foundation Inc. als zusätzliche Einnahmequellen. Der Vorteil für diese Unternehmen besteht dabei darin, dass sie im Zuge einer Mitgliedschaft in der gemeinnützigen Organisation auch ein Stimmrecht im Hinblick auf die Entwicklungsrichtung der Projektsoftware erhalten. Weitere Einnahmequellen existieren nicht. *[Administrator][Webseite]*

2) Wie viele bezahlte Entwickler besitzt das Open Source Projekt?

Weder die Entwickler im Open Source Projekt noch die Mitglieder der gemeinnützigen Organisation werden für ihre Tätigkeiten bezahlt. *[Administrator][Webseite]*

3) Wie viele Unternehmen bieten Support für die Software des OSPs an?

Da das Mambo-Projekt im Gegensatz zu Drupal und Magnolia keine Unternehmen auszeichnet, die Dienstleistungen zur Software anbieten, existieren auch keine Informationen zur Anzahl der Unternehmen, die professionellen Support zum Mambo WCM-System anbieten. *[Administrator][Webseite]*

4) Wie lange benötigt ein neuer Entwickler, um das Wissen eines Kernentwicklers zu erlangen?

Ein neu hinzukommender Entwickler benötigt etwa 1-2 Jahre, um das Wissen eines Kernentwicklers zu erlangen. *[Administrator]*

5) Welche Mechanismen sind in dem OSP etabliert, um neue Entwickler dabei zu unterstützen, dieses Wissen zu erlangen?

Im Mambo-Projekt etablierte Strukturen zur Unterstützung von Entwicklern sind die Beschreibung der API, ein Bug-Tracking-System, Tutorials, ein FAQ-Bereich zur Klärung häufiger Fragestellungen sowie Codierrichtlinien zur Vereinheitlichung der Gestalt des Quellcodes. Zusätzlich beantwortet der Projektadministrator technische

Fragen auf den Mailinglisten und gibt ein Feedback zu neuen Codierbeiträgen. *[Administrator]*

6) Wie viel Zeit verbringen die Kernentwickler des OSP damit, andere Entwickler zu unterstützen?

Der Projektadministrator verbringt etwa 3-5 Stunden wöchentlich mit der Unterstützung anderer Entwickler. *[Administrator]*

7) Welche Kernwerte besitzt das OSP und in welcher Form werden sie formuliert?

Die Kernwerte von Mambo liegen in ihrem Werbeslogan *„Power in Simplicity“* und bestehen in

- der einfachen Benutzbarkeit,
- der Flexibilität
- und der trotzdem vorhandenen Mächtigkeit der Anwendung. *[Administrator]*

Allerdings werden diese Kernwerte nicht über den Slogan hinaus erfolgt auf der Webseite des Projektes formuliert und erläutert. *[Webseite]*

8) Welche Daseinsberechtigung über finanzielle Aspekte hinaus besitzt das OSP?

Die Daseinsberechtigung von Mambo besteht in der Entwicklung des besten Open Source Content Management Systems der Welt. *[Administrator]*

Auf der Webseite des Projektes wird lediglich eine Daseinsberechtigung für die Mambo Foundation Inc. formuliert, nicht aber für das Open Source Projekt selbst. *[Webseite]*

9) Welche langfristigen herausfordernden Ziele besitzt das OSP und in welcher Form werden sie formuliert?

Mambo hat sich zum Ziel gesetzt, das beste Open Source CMS weltweit zu entwickeln. *[Administrator]*

Zudem besitzt das Projekt eine sehr professionell organisierte Roadmap, die die zukünftig geplanten Features von der aktuellen Entwicklungsversion 4.7 bis zur Version 5.0 nennt und erläutert. *[Webseite]*

10) Wie groß ist die Projektcommunity?

Das Team, das innerhalb der letzten ca. 1,5 Jahre die Entwicklung der Projektsoftware im SVN-System realisiert hat, besteht vergleichbar zum Magnolia-Projekt aus insgesamt 15 Entwicklern. *[SVN]*

11) Wie groß ist die Projektsoftware?

Die SVN-Version der Mambo-Projektsoftware umfasst insgesamt 261.516 Lines of Code. Als einziges der untersuchten Open Source Projekte entstammt die analysierte

Version dabei allerdings nicht dem SVN-Trunk, dem üblichen Hauptentwicklungszweig, sondern der aktuellsten Entwicklungsverzweigung, der Mambo-Version 4.6. Dies liegt daran, dass der Trunk seit etwa 1 Jahr nicht mehr aktualisiert wurde. *[SVN]*

12) Wie aktiv ist die Projektcommunity?

Die Mambo-Software im SVN-System wurde von insgesamt 15 Entwicklern realisiert. Von diesen 15 Entwicklern haben vier Entwickler kumulativ mehr als 80% der bestehenden Codezeilen implementiert. Dennoch kann das Projekt auf eine vergleichsweise hohe Zahl von relevant beteiligten Entwicklern verweisen. So haben insgesamt acht Entwickler mehr als 10.000 Quellcodezeilen in das Versionsverwaltungssystem integriert. In den letzten drei Monaten waren darüber hinaus insgesamt 9 Entwickler aktiv.

Betrachtet man die Entwicklung des Quellcodes von Mambo in den letzten 12 Monaten, so zählt das Projekt ähnlich wie Magnolia zu den aktivsten Projekten sowohl in der Zahl der hinzugefügten Quellcodezeilen als auch in der Zahl der durchgeführten Commits. Anders als im Magnolia-Projekt setzt sich dieser Trend auch fort, werden lediglich die letzten sechs bzw. die letzten drei Monate berücksichtigt.

Betrachtet man die Entwicklung allerdings genauer, so fällt auf, dass die durchschnittliche Zahl der hinzugefügten Codezeilen innerhalb des letzten Jahres dennoch rückläufig ist, während die durchschnittliche Zahl der durchgeführten Commits konstant bleibt bzw. sogar steigt. Dieser Umstand hängt vermutlich mit der Entwicklung der Mambo-Version 4.6 zusammen. Die Erstellung dieser Version beinhaltete eine komplette Neuentwicklung des Anwendungskerns, die sich von Januar 2006 bis September 2006 über insgesamt neun Monate erstreckte. Im Zuge dessen wurden von Januar bis Juni über 100.000 neue Quellcodezeilen in das SVN-System integriert. Ab Juni erfolgte dann eine Korrektur- und Anpassungsphase, im Zuge derer bis zum Releasetermin im September hauptsächlich Korrekturen vorgenommen wurden. Gedeckt wird diese Annahme durch die im Gegensatz zur Zahl der hinzugefügten nicht rückläufigen Zahl der geänderten Codezeilen.

Nach dem Releasetermin im September ist eine etwa einmonatige Phase geringerer Aktivität zu beobachten, bevor die Weiterentwicklung der Software wieder anläuft. *[Webseite][SVN]*

13) Wie entwickelt sich die Größe der Projektcommunity in fünf Jahren?

Die Projektcommunity wird wesentlich größer sein. *[Administrator]*

14) Wie entwickelt sich die Qualität der Software in fünf Jahren?

Die Software wird „mature“ sein. *[Administrator]*

15) Welche Probleme können die zukünftige Weiterentwicklung der Projektsoftware behindern?

Die zukünftige Entwicklung von Mambo kann durch Probleme in der Akquirierung neuer Entwickler sowie durch eine geringe Anzahl an Kernentwicklern behindert werden. Zudem muss der Entwicklungsaufwand finanziert werden. *[Administrator]*

16) Was wird den Erfolg des Open Source Projektes ausmachen?

Der zukünftige Erfolg des Mambo-Projektes wird durch eine aktive Community, motivierte und talentierte Entwickler sowie klare Ziele geprägt. *[Administrator]*

7.1.4 OpenCms

1) Welche Einnahmequellen besitzt das OSP?

OpenCms ist angelehnt an die Firma Alkacon, die die Entwicklung der Projektsoftware zentral koordiniert und in der die Kernentwickler des Open Source Projektes organisiert sind. Alkacons Einnahmen entstehen dabei aus professionellen Dienstleistungen zur Software von OpenCms, die das Unternehmen anbietet. Diese Dienstleistungen umfassen technische Unterstützung, Schulungen, die Erstellung individueller Erweiterungen, die Beratung durch Alkacon-Mitarbeiter sowie das komplette *Outsourcing*[159] unternehmensinterner Softwareprojekte. Zudem bietet Alkacon eine vorgefertigte Kollektion spezieller Erweiterungen des OpenCms-Kerns, die so genannten *Enterprise Extensions*, unter einer proprietären Lizenz an. *[Administrator][Webseite]*

2) Wie viele bezahlte Entwickler besitzt das Open Source Projekt?

OpenCms besitzt über 10 bezahlte Entwickler. *[Administrator]*

3) Wie viele Unternehmen bieten Support für die Software des OSPs an?

Es existieren etwa 120 bei Alkacon registrierte Unternehmen, die Support zur Projektsoftware anbieten. *[Administrator][Webseite]*

4) Wie lange benötigt ein neuer Entwickler, um das Wissen eines Kernentwicklers zu erlangen?

Ein neuer Entwickler benötigt etwa 1-2 Jahre, um das Wissen eines Kernentwicklers zu erlangen. *[Administrator]*

---

[159] Der Begriff „Outsourcing“ bezeichnet das Auslagern bestimmter Arbeiten, speziell der Erstellung von Softwarelösungen, in andere Unternehmen, die sich auf diese Aufgaben spezialisiert haben

5) Welche Mechanismen sind in dem OSP etabliert, um neue Entwickler dabei zu unterstützen, dieses Wissen zu erlangen?

Die Strukturen, die im OpenCms-Projekt zur Unterstützung neuer und bestehender Entwickler etabliert sind, umfassen die Beschreibung der API sowie ein Bug-Tracker-System zur Verwaltung von Programmdefekten und Feature Requests. Zudem beantwortet der Projektadministrator technische Fragen auf der Entwicklermailingliste. *[Administrator]*

6) Wie viel Zeit verbringen die Kernentwickler des OSP damit, andere Entwickler zu unterstützen?

Der Administrator von OpenCms investiert wöchentlich etwa 1 Stunde in die Unterstützung neuer und bestehender Entwickler. *[Administrator]*

7) Welche Kernwerte besitzt das OSP und in welcher Form werden sie formuliert?

OpenCms formuliert keine speziellen Kernwerte des Open Source Projektes. *[Administrator][Webseite]*

8) Welche Daseinsberechtigung über finanzielle Aspekte hinaus besitzt das OSP?

Arbeiten sollte Spaß machen. *[Administrator]*

Darüber hinaus existieren keine vergleichbaren Aussagen auf der Webseite von OpenCms. *[Webseite]*

9) Welche langfristigen herausfordernden Ziele besitzt das OSP und in welcher Form werden sie formuliert?

Das Ziel von OpenCms besteht darin, das weltweit beste WCM-System zu erstellen. *[Administrator]*

Zusätzlich formuliert Alkacon eine Roadmap für das Projekt, die im Vergleich zu Mambo zwar weniger umfangreich ist, dafür allerdings einen konkreteren zeitlichen Ablauf beinhaltet. *[Webseite]*

10) Wie groß ist die Projektcommunity?

Die Entwicklung des Anwendungskerns von OpenCms im CVS wurde von insgesamt 25 Entwicklern getragen. Der Entwicklungszeitraum umfasst dabei etwa 7 Jahre. *[CVS]*

11) Wie groß ist die Projektsoftware?

Die CVS-Head-Revision (Stand: 13.04.2007) des OpenCms-Kerns umfasst insgesamt 678.326 Lines of Code. *[CVS]*

12) Wie aktiv ist die Projektcommunity?

Im Vergleich zu den anderen analysierten Open Source Projekten besitzt OpenCms die breiteste Verteilung der Entwicklungsarbeit auf die beteiligte Entwicklergruppe. So sind es immerhin fünf der insgesamt beteiligten 25 Entwickler, die mehr als 80% des Quellcodes der Anwendung implementiert haben. Zudem haben fünf weitere mehr als 10.000 und insgesamt 15 Entwickler mehr als 1.000 Zeilen Quellcode zur Software beigesteuert.

Der Verlauf der Entwicklung des Quellcodes im CVS innerhalb der letzten 12 Monate zeigt allerdings, dass das Entwicklungsteam von OpenCms im Vergleich zu den anderen untersuchten Open Source Projekten in diesem Zeitraum weniger aktiv war. Dies manifestiert sich sowohl in der vergleichsweise niedrigen Zahl der durchgeführten Commits als vor allem auch in der geringen Anzahl hinzugefügter Quellcodezeilen. Die letzte umfangreichere Änderung des Quellcodes wurde im März 2006 kurz vor der Freigabe der OpenCms-Version 6.2 durchgeführt. Seitdem sind die Entwickler zwar nach wie vor regelmäßig[160] aktiv, allerdings nicht in gleichem Maße wie z.B. die Entwickler des Mambo-Projektes.

Betrachtet man diesen Umstand vor dem Hintergrund, dass OpenCms, gemessen an der Gesamtzahl der Quellcodezeilen, die insgesamt umfangreichste Anwendung innerhalb der Untersuchungsgruppe darstellt, so suggeriert der vergleichsweise geringe Umfang der in den letzten 12 Monaten durchgeführten Änderungen, dass die Projektsoftware funktional betrachtet bereits einen gewissen Reifegrad erreicht hat, sodass lediglich korrigierende Maßnahmen vorgenommen werden müssen. *[Webseite][SVN]*

13) Wie entwickelt sich die Größe der Projektcommunity in fünf Jahren?

Die Community wird wesentlich größer sein. *[Administrator]*

14) Wie entwickelt sich die Qualität der Software in fünf Jahren?

Die Qualität der Software wird höher sein. *[Administrator]*

15) Welche Probleme können die zukünftige Weiterentwicklung der Projektsoftware behindern?

Die zukünftige Weiterentwicklung der Software kann lediglich durch die Finanzierung des Entwicklungsaufwandes beeinträchtigt werden. *[Administrator]*

16) Was wird den Erfolg des Open Source Projektes ausmachen?

Entscheidend für den zukünftigen Erfolg von OpenCms wird die Fokussierung auf den Endnutzer im Gegensatz zum Entwickler sein. *[Administrator]*

---

[160] mit Ausnahme des WM-Monats Juni

7.1.5 TGS CMS

1) Welche Einnahmequellen besitzt das OSP?

Das Open Source Projekt TGS CMS besitzt keine Einnahmequellen. Im Gegensatz zu den drei anderen nicht-kommerziellen Open Source Projekten ist das TGS CMS darüber hinaus auch an keine gemeinnützige Organisation angelehnt. *[Webseite][Administrator]*

2) Wie viele bezahlte Entwickler besitzt das Open Source Projekt?

TGS CMS besitzt keine bezahlten Entwickler. *[Administrator]*

3) Wie viele Unternehmen bieten Support für die Software des OSPs an?

Es existiert ein Unternehmen im Umfeld von TGS CMS, das Support zur Projektsoftware anbietet. *[Administrator]*

4) Wie lange benötigt ein neuer Entwickler, um das Wissen eines Kernentwicklers zu erlangen?

Neue Entwickler benötigen weniger als 1 Jahr, um das Wissen eines Kernentwicklers zu erlangen. *[Administrator]*

5) Welche Mechanismen sind in dem OSP etabliert, um neue Entwickler dabei zu unterstützen, dieses Wissen zu erlangen?

Die zur Unterstützung neuer Entwickler etablierten Strukturen im TGS-CMS-Projekt umfassen ein Bug-Tracking-System, einen FAQ-Bereich sowie Codierrichtlinien. Zudem gibt der Administrator des Open Source Projektes Feedback zu eingehenden Codierbeiträgen und pflegt bei Fragen einen direkten Kontakt zu potentiellen Neuentwicklern. *[Administrator]*

6) Wie viel Zeit verbringen die Kernentwickler des OSP damit, andere Entwickler zu unterstützen?

Der Projektadministrator von TGS CMS investiert wöchentlich mehr als 10 Stunden in die Unterstützung von Benutzern und potentiellen Entwicklern. *[Administrator]*

7) Welche Kernwerte besitzt das OSP und in welcher Form werden sie formuliert?

TGS CMS formuliert keine Kernwerte. *[Administrator][Webseite]*

8) Welche Daseinsberechtigung über finanzielle Aspekte hinaus besitzt das OSP?

Die Daseinsberechtigung von TGS CMS besteht in Entwicklung eines neuen WCM-Systems, das Geschäftslogik, umfangreiche Funktionalität und einfache Bedienung umfasst. *[Administrator]*

Darüber hinaus wird keine entsprechende Daseinsberechtigung auf der Webseite des Open Source Projektes formuliert. *[Webseite]*

9) Welche langfristigen herausfordernden Ziele besitzt das OSP und in welcher Form werden sie formuliert?

Das Ziel von TGS CMS besteht in der Schaffung eines vermarktungsfähigen WCM-Systems. *[Administrator]*

Darüber hinaus werden keine Ziele auf der Webseite des Open Source Projektes formuliert. *[Webseite]*

10) Wie groß ist die Projektcommunity?

Im Laufe der Entwicklung der Projektsoftware von TGS CMS haben insgesamt drei Entwickler Beiträge im SVN-System geleistet. *[SVN]*

11) Wie groß ist die Projektsoftware?

Der SVN-Trunk (Stand: 13.04.2007) von TGS CMS umfasst insgesamt 99.165 Lines of Code. *[SVN]*

12) Wie aktiv ist die Projektcommunity?

In den letzten fünf Monaten waren insgesamt drei Entwickler im Quellcodeverzeichnis von TGS CMS aktiv, wobei über 95% des Entwicklungsaufwandes von den zwei aktivsten Entwicklern geleistet wurde.

In diesen fünf Monaten wurde in zwei Schüben der gesamte existierende Quellcode der Projektsoftware in das SVN-System integriert. Entsprechend hoch sind auch die Anzahlen der in dieser Zeit hinzugefügten bzw. geänderten Quellcodezeilen. Allerdings äußert sich dieses Bild nicht in einer vergleichbar hohen Anzahl von durchgeführten Commits, sodass der Entwicklungsprozess von TGS CMS bislang durch schubweise hohe Aktivitätsphasen gekennzeichnet ist. *[SVN]*

13) Wie entwickelt sich die Größe der Projektcommunity in fünf Jahren?

Die Projektcommunity wird wesentlich größer sein. *[Administrator]*

14) Wie entwickelt sich die Qualität der Software in fünf Jahren?

Die Qualität der Software wird höher sein. *[Administrator]*

15) Welche Probleme können die zukünftige Weiterentwicklung der Projektsoftware behindern?

TGS CMS besitzt nur wenige Benutzer. Zudem könnten unterschiedliche Ziele der Entwickler sowie die Finanzierung des Entwicklungsaufwandes die Weiterentwicklung des Open Source Projektes und seiner Software behindern.

16) Was wird den Erfolg des Open Source Projektes ausmachen?

Der Erfolg von TGS CMS wird geprägt durch die Kombination von einfacher Benutzbarkeit und einem hohen Funktionsumfang. *[Administrator]*

7.1.6 Typo3

1) Welche Einnahmequellen besitzt das OSP?

Das Typo3-Projekt finanziert die Entwicklung der Projektsoftware vor allem durch zwei Quellen von Einnahmen. Diese bestehen in Spendengeldern von Unternehmen und Einzelpersonen sowie in Beiträgen für die Mitgliedschaft in der *Typo3 Association*. Die Typo3-Association ist eine gemeinnützige Organisation, die seit dem Jahr 2004 die Entwicklung der Software im Typo3-Projekt fördert und steuert. Die Vorteile, die Unternehmen und Einzelpersonen mit einer Mitgliedschaft in der Typo3 Association erwerben können, liegen in der Einflussnahme auf die Geschehnisse im Open Source Projekt. Damit verfolgt das Typo3-Projekt die analoge Vorgehensweise wie das Mambo-Projekt, das ebenfalls einen Erlös aus den Mitgliedsbeiträgen zu der an das Projekt angelehnten gemeinnützigen Organisation bezieht. Zusätzlich zu den beiden genannten Einnahmequellen, erhält die Typo3 Association zudem einen Teilerlös aus dem Verkauf von Merchandising-Artikeln wie T-Shirts oder Rucksäcken. *[Webseite][Administrator]*

2) Wie viele bezahlte Entwickler besitzt das Open Source Projekt?

Typo3 besitzt etwa 3-5 bezahlte Entwickler. Der Projektgründer und aktivste Kernentwickler erhält eine jährliche Unterstützung, die allerdings nicht mit einem vollzeitlichen Einkommen vergleichbar ist. Zudem wird aktuell ein dreiköpfiges Team von Entwicklern auf stündlicher Basis bezahlt, um für die Typo3-Version 5.0 eine totale Neuimplementierung der aktuellen Version zu realisieren. *[Administrator]*

3) Wie viele Unternehmen bieten Support für die Software des OSPs an?

Es existieren mehr als 1000 Unternehmen im Umfeld des Open Source Projektes, die Support für die Projektsoftware anbieten. Allerdings differenziert die Typo3 Association zwischen zertifizierten und nicht-zertifizierten Unternehmen. Unternehmen beider Gruppen sind auf der Webseite des Open Source Projektes aufgelistet, wobei die etwa 120 ausgezeichneten Unternehmen auf einer gesonderten Liste geführt werden. Dazu müssen sie allerdings sowohl Leistungen zur Weiterentwicklung der Projektsoftware erbracht haben, als auch Mitglied der Typo3 Association sein. Dies erfordert die Zahlung eines Mitgliedsbeitrages. *[Webseite][Administrator]*

4) Wie lange benötigt ein neuer Entwickler, um das Wissen eines Kernentwicklers zu erlangen?

Ein neuer Entwickler benötigt etwa 1-2 Jahre, um das Wissen eines Kernentwicklers zu erlangen. *[Administrator]*

5) Welche Mechanismen sind in dem OSP etabliert, um neue Entwickler dabei zu unterstützen, dieses Wissen zu erlangen?

Die Strukturen, die zur Unterstützung neuer und bestehender Entwickler im Typo3-Projekt etabliert sind, umfassen einer Beschreibung der API, einen Bug-Tracker zur Verwaltung von Programmdefekten und Feature Requests, Codierrichtlinien zur Vereinheitlichung der Gestalt des Quellcodes sowie einen Commiter-Guide zur Einführung neuer Entwickler in die projektspezifischen Abläufe zur Einsendung von Codierbeiträgen. Zudem beantwortet der Projektadministrator technische Fragen, Fragen zur Entwicklungsrichtung, Fragen zu neuen Beiträgen und gibt außerdem ein Feedback zu eingehenden Codierbeiträgen. Schließlich ist der sehr detailliert dokumentierte Quellcode eine zusätzliche Unterstützung für neue und bestehende Entwickler. *[Administrator]*

6) Wie viel Zeit verbringen die Kernentwickler des OSP damit, andere Entwickler zu unterstützen?

Der Projektadministrator verbringt etwa 2 Stunden wöchentlich mit der Unterstützung anderer Entwickler. *[Administrator]*

7) Wie groß ist die Projektcommunity?

Die Entwicklung der Projektsoftware im SVN-System wurde von insgesamt 25 Entwicklern vorangetrieben. Der Entwicklungszeitraum im SVN-System umfasst dabei etwa 3,5 Jahre, wobei das Open Source Projekt selbst seit etwa 8 Jahren besteht. *[Webseite][SVN]*

8) Wie groß ist die Projektsoftware?

Der SVN-Trunk (Stand:13.04.2007) des Typo3-Kerns umfasst insgesamt 389.242 Lines of Code. *[SVN]*

9) Wie aktiv ist die Projektcommunity?

Die Entwicklung der Software von Typo3 wurde von insgesamt 25 Entwicklern realisiert. Dabei fällt vor allem ein Entwickler auf, der über 70% aller insgesamt eingecheckten Codezeilen auf sich vereint. Allerdings haben darüber hinaus vergleichbar zu OpenCms 10 Entwickler mehr als 1.000 Quellcodezeilen implementiert, sodass trotz der dominanten Aktivität des aktivsten Entwicklers eine breite Personengruppe relevante Beiträge leistet. Zudem besitzt Typo3, gemessen an der Zahl der aktiven Entwickler, die aktivste Community in den letzten 3, 6 und 12 Monaten.

Betrachtet man dagegen den Verlauf der Entwicklung des Quellcodes in den letzten 12 Monaten, so gehört die Typo3-Community sowohl in der Zahl der Commits als auch in

der Zahl der hinzugefügten Quellcodezeilen nicht zu den aktivsten innerhalb der Vergleichsgruppe.

Diese vergleichsweise geringe Aktivität in diesem Zeitraum erklärt sich wahrscheinlich durch die Freigabe der Typo3-Version 4.0 im April 2006. Im Vorfeld der Freigabe der neuen Softwareversion wurden zwei umfangreichere Änderungen an der Software vorgenommen, sodass nach der Freigabe des Quellcodes der Typo3-Version 4.0 voraussichtlich vor allem Korrekturmaßnahmen vorgenommen wurden. Diese Annahme wird durch zwei Punkte bestätigt.

Zum einen deutet die im Gegensatz zur Zahl der hinzugefügten Quellcodezeilen steigende Zahl der Commits in den letzten drei Monaten auf eine Pflege- und Wartungsphase hin, im Zuge derer zwar viele, allerdings wenig umfangreiche Korrekturen vorgenommen werden.

Zum anderen belegt die Aussage des Projektadministrators, dass parallel zur Weiterentwicklung der Typo3-Version 4.x an der Neuimplementierung des Anwendungskerns für Typo3 5.0 gearbeitet wird, sodass umfangreichere funktionale Erweiterungen möglicherweise auf die Version 5.x verschoben werden. *[Webseite][Administrator][SVN]*

10) Welche Kernwerte besitzt das OSP?

Die Kernwerte sowie eine Daseinsberechtigung über finanzielle Interessen hinaus formuliert Typo3 auf seiner Webseite in Form eines Leitsatzes:

> *„Our vision: inspiring people to share."*
>
> *"Our mission: to jointly innovate excellent free software enabling people to communicate"*

Allerdings sind diese Werte nicht vergleichbar zu Drupal und Magnolia weiter ausgeführt.[161] *[Webseite]*

11) Welche Daseinsberechtigung über finanzielle Aspekte hinaus besitzt das OSP?

Siehe oben.[161]

12) Welche langfristigen herausfordernden Ziele besitzt das OSP?

Typo3 arbeitet aktuell an der Neuimplementierung des Anwendungskerns für die Typo3-Version 5.0, um dauerhaft konkurrenzfähig zu bleiben. *[Administrator]*

Zudem besitzt das Projekt eine offizielle Roadmap, die den aktuellen Entwicklungsstand sowie geplante Features im Einzelnen beschreibt. *[Webseite]*

---

[161] Die entsprechende Frage wurde vom Projektadministrator nicht beantwortet

13) Wie entwickelt sich die Größe der Projektcommunity in fünf Jahren?

Die Projektcommunity wird wesentlich größer sein. *[Administrator]*

14) Wie entwickelt sich die Qualität der Software in fünf Jahren?

Die Qualität der Software wird höher sein. *[Administrator]*

15) Welche Probleme können die zukünftige Weiterentwicklung der Projektsoftware behindern?

Die zukünftige Entwicklung von Typo3 kann durch nachlassende Entwicklermotivation sowie die Koordination der Entwickler behindert werden. Der letzte Aspekt bezieht sich dabei auf den Umstand, dass zur Einhaltung der Qualitätsstandards von Typo3 jeder eingehende Codierbeitrag zum Anwendungskern von zwei zusätzlichen Entwicklern kontrolliert werden muss. Da die meisten Entwickler eher Codieraufgaben erledigen, fällt diese Arbeit meist auf die Kernentwickler zurück, was zukünftig zu Motivationsproblemen der Kernentwickler führen kann. *[Administrator]*

16) Was wird den Erfolg des Open Source Projektes ausmachen?

Der Erfolg des Typo3-Projektes wird im Wesentlichen durch vier Komponenten bestimmt. Dies sind die große, aber zentral koordinierte Community, der *Extension Manager* zur Integration von Programmerweiterungen, der große Umfang an vorhandener Dokumentation sowie die Stabilität und Reliabilität des WCM-Systems. *[Administrator]*

7.2 Einflüsse auf den Zielfaktor „Zukunftssicherheit“

7.2.1 Kommerzielle Ausrichtung

Die Ergebnisse der Datenerhebung im Kontext der fünf analysierten Open Source Projekte lassen klar zwei Tendenzen bzgl. der wahrgenommenen zukünftigen Herausforderungen erkennen.

Die Administratoren der beiden kommerziell orientierten Projekte, des Magnolia- und des OpenCms-Projektes, nennen insgesamt zwei Probleme, die die Weiterentwicklung der Projektsoftware zukünftig behindern könnten. Dies sind die Finanzierung des Entwicklungsaufwandes sowie die Gewinnung neuer Entwickler.

Tabelle 10: Erfassung der Zukunftssicherheit des Magnolia- und des OpenCms-Projektes

OSP	*Zuk.Com.Gr.*[162]	*Zuk.SW.Qualität*[163]	*Mögliche Probleme*
Magnolia	much larger/ little larger	higher/mature	• Finanzierung des Entwicklungsaufwandes • Gewinnung neuer Entwickler
OpenCms	much larger	higher	• Finanzierung des Entwicklungsaufwandes

Diese Ergebnisse decken sich mit den Herausforderungen, die kommerzielle Open Source Projekte in der Regel bewältigen müssen.

Ein Open Source Projekt, das verbunden mit der Anlehnung der Softwareentwicklung an ein zentrales Unternehmen einen kommerziellen Zweig eröffnet, steht vor dem Problem, innerhalb der Open Source Bewegung Akzeptanz gewinnen zu müssen. Dies hängt damit zusammen, dass die verantwortlichen Kernentwickler und Mitarbeiter des Unternehmens im Zuge der kommerziellen Orientierung des jeweiligen Open Source Projektes meist an einer stärkeren Kontrolle der Entwicklungsrichtung sowie des Entwicklungsprozesses insgesamt interessiert sind. Eine solche Haltung steht allerdings im Gegensatz zu den Grundwerten der offenen Softwarebewegung, die vor allem auch die freie, unbegrenzte Entfaltung mit einschließen. Zudem entwickelte sich historisch[164] betrachtet erst durch die Kommerzialisierung des Produktes „Software“ die heute verbreitete Limitierung der Freiheitsrechte an dem Quellcode einer Software, weswegen viele Entwickler innerhalb der Open Source Bewegung bereits aus diesem Grund einem kommerziellen Open Source Projekt kritisch gegenüberstehen. Folglich besteht eines der zentralen Probleme kommerzieller Open Source Projekte in der Akquirierung und Bindung neuer und auch bestehender Entwickler.

Die zweite Herausforderung, die im Zuge einer kommerziellen Orientierung von Open Source Projekten entsteht, hängt weniger mit den Belangen der offenen Softwareentwicklung selbst zusammen. Vielmehr beinhaltet eine kommerzielle Ausrichtung das Problem, die Entwicklungs- und Mitarbeiterkosten für die Weiterentwicklung und Pflege der Software decken zu müssen. Während in nicht-

[162] Zukünftige Communitygröße in 5 Jahren

[163] Zukünftige Qualität der Software in 5 Jahren

[164] siehe Kapitel 4.3.6 „Geschichte“

kommerziellen Open Source Projekten in der Regel keine oder nur geringe Kosten für die Entwickler der Software anfallen, bewirkt die Kommerzialisierung verbunden mit der hauptberuflichen Involvierung der Kernentwickler der Projekte auch ein höheres wirtschaftliches Risiko. Dies gilt insbesondere, da es die Offenheit des Quellcodes auch Drittparteien ermöglicht, professionelle Dienstleistungen anzubieten.

Im Gegensatz dazu bestehen die Probleme nicht-kommerzieller Open Source Projekte neben der Akquirierung vor allem in der Koordination sowie in der Ausbildung neuer und bestehender Entwickler.

Tabelle 11: Erfassung der Zukunftssicherheit von Drupal, Mambo, Typo3 und TGS CMS

OSP	*Zuk.Com.Gr.*[165]	*Zuk.SW.Qualität*[166]	*Mögliche Probleme*
Drupal	much larger	mature	• Fähigkeiten neuer Entwickler • Schulung neuer Entwickler
Mambo	much larger	mature	• Gewinnung neuer Entwickler • nur wenige Kernentwickler • Finanzierung des Entwicklungsaufwandes
Typo3	much larger	higher	• sinkende Motivation der Entwickler • Koordinierung der Entwicklungsarbeiten
TGS CMS	much larger	higher	• nur wenige Benutzer • Finanzierung des Entwicklungsaufwandes • unterschiedliche Ziele der Entwickler

[165] Zukünftige Communitygröße in 5 Jahren

[166] Zukünftige Qualität der Software in 5 Jahren

Dies liegt in den abweichenden Eigenschaften des Entwicklungsprozesses begründet. Während in kommerziellen Open Source Projekten die Entwicklung der Software durch ein Unternehmen, das bereits aus finanziellen Gründen den Kern der bezahlten Entwickler gering halten möchte, zentral gesteuert und koordiniert wird, beinhaltet der Entwicklungsprozess in nicht-kommerziellen Open Source Projekten eine stärkere Abhängigkeit von der Motivation der freiwilligen Teilnehmer. Vor dem Hintergrund, dass diese Motivation im Laufe der Zeit auch innerhalb der Kernentwicklergruppe nachlassen kann, muss es das Ziel der Open Source Projekte sein, in Abhängigkeit von der Komplexität der Anwendung eine angemessene Zahl von beteiligten Entwicklern an den Kern der jeweiligen Software heranzuführen, um die Abhängigkeit von einzelnen, stark involvierten Personen zu verringern. Die drei wesentlichen Elemente in diesem Zusammenhang sind erstens die Gewinnung von Entwicklern, zweitens die Schulung der Entwickler zu Kernentwicklern und drittens die Koordination des Entwicklungsprozesses.

Alle drei genannten Aspekte finden sich auch in den Resultaten der Datenerhebung zu den vier nicht-kommerziell ausgerichteten Open Source Projekten wieder. So nennt der Administrator des Mambo-Projektes die Gewinnung neuer Entwickler sowie den Umstand, dass das Projekt nur wenige Kernentwickler umfasst als Risiken für die zukünftige Weiterentwicklung der Projektsoftware. Der Administrator des Drupal-Projektes sieht dagegen nicht die Anzahl, sondern die Fähigkeiten bzw. die Schulung neuer Entwickler als zentrale Probleme. Schließlich belegen die Umfrageresultate zum Typo3-Projekt, dass auch nachlassende Motivation der Entwickler bzw. die Koordination der bestehenden Community zu den Herausforderungen in Open Source Projekten im Hinblick auf die Sicherung der zukünftigen Weiterentwicklung der Projektsoftware zählen. Der Administrator von TGS CMS nennt darüber hinaus die geringe Anzahl von Benutzern als Problem des Open Source Projektes. Dies hängt damit zusammen, dass das Projekt noch relativ jung ist, sodass sich bislang noch keine aktive Benutzergemeinschaft um das Projekt herum bilden konnte. Dies ist insofern für den zukünftigen Erfolg eines Open Source Projektes entscheidend, als dass sich eine aktive Entwicklercommunity meist aus ursprünglichen Benutzern der Software bildet. TGS CMS formuliert zudem genau wie das Mambo-Projekt die Finanzierung des Entwicklungsaufwand als zusätzliches mögliches Problem für die zukünftige Entwicklung des Projektes. Dies widerspricht zunächst der nicht-kommerziellen Ausrichtung der beiden Projekte, muss aber vor dem Hintergrund, dass beide Projekte keine Entwickler aktiv finanzieren, eher mit der Finanzierung der Infrastruktur oder anderen anfallenden Kosten in Verbindung gebracht werden.

Der unterschiedliche Grad der Abhängigkeit von der freiwilligen Entwicklercommunity, der aus den genannten Aspekten zwischen kommerziellen und nicht-kommerziellen Open Source Projekten besteht, spiegelt sich auch in den Angaben der

Projektadministratoren zu den Kriterien, die den Erfolg des jeweiligen Open Source Projektes ausmachen werden, wider. Während die beiden kommerziellen Projekte dabei vor allem kundenorientierte Kriterien angeben, rücken die nicht-kommerziellen Projekte eher community-orientierte Aspekte in den Vordergrund. Dies lässt darauf schließen, dass nicht-kommerzielle Open Source Projekte primär die Community selbst als wesentlichstes Potential wahrnehmen, während für kommerzielle Projekte darüber hinaus verständlicherweise auch das Kundenpotential entscheidend ist.

Tabelle 12: Kriterien für den zukünftigen Erfolg der untersuchten Open Source Projekte

	Was wird den Erfolg des Projektes ausmachen?
TGS CMS	• Kombination von Benutzbarkeit und hohem Funktionsumfang
Mambo	• Klare Ziele und objectives • ein motiviertes Team • Talentierte Entwickler • eine aktive Community
Typo3	• große Community aber zentral organisiert (via typo3.org platform +newsgroups / mailing lists) • Extension Manager zur einfachen Integration neuer Funktionalitäten • Umfangreiche Dokumentation (Online References / Manuals, Books, Videos, a Magazine etc.) • Stabilität and Reliabilität
OpenCMS	• Auslieferung eines guten Produktes für den Kunden, nicht für den Entwickler
Magnolia	• Benutzbarkeit • Unternehmensfokus • JCR • Ajax
Drupal	• Starke Führung

Zusammenfassend lässt sich zum Einfluss der kommerziellen Ausrichtung auf die Zukunftssicherheit der Open Source Projekte sagen, dass sie die Art der zu bewältigenden Herausforderungen bestimmt.

Kommerzielle Open Source Projekte sollten demnach einen stabilen Kern von bezahlten Entwicklern umfassen, dessen Größe der Größe der Anwendung entspricht. Darüber hinaus ist es wichtig, eine aktive Gemeinschaft von Co-Entwicklern im Open Source Projekt zu besitzen, die den ansonsten hohen Aufwand zur Pflege und Wartung einer

Software verringern. Schließlich ist es zur Finanzierung des Entwicklungsprozesses entscheidend, dass das Open Source Projekt eine möglichst hohe Verbreitung besitzt. Ein Indiz kann in diesem Zusammenhang im Feld der WCM-Systeme die häufig geführte Rubrik der Referenzseiten sein, zu deren Erstellung das entsprechende System eingesetzt wird.

Demgegenüber ist es für nicht-kommerzielle Open Source Projekte wichtig, bestehende Abhängigkeiten von einzelnen Entwicklern zu verringern sowie Mechanismen zur Steuerung des Entwicklungsprozesses zu etablieren. Zum ersten Aspekt gehören dabei eine stabile Anzahl von Kernentwicklern, eine aktive Community sowie Mechanismen und Aktivitäten zur Schulung[167] neuer und bestehender Co-Entwickler der Software. Im Zusammenhang mit dem zweiten Aspekt weisen die Resultate der Datenerhebung darauf hin, dass Open Source Projekte, die eine gewisse Größe überschreiten, in der Regel Anlehnung an eine bekannte gemeinnützige Organisation suchen oder eine solche gründen. Diese Organisationen dienen dabei meist der Erfüllung dreier Ziele. Erstens dient eine solche Anbindung des Open Source Projektes an eine Organisation der Formalisierung der Rechtslage hinsichtlich der Urheberrechte an der jeweiligen Softwareanwendung. Zweitens wird darüber hinaus eine zentrale Verwaltung von Spendengeldern und anderen Einkünften ermöglicht. Drittens schließlich geht mit der Gründung einer zentralen Organisation im Projektumfeld meist der Wunsch der Projektadministratoren einher, die Koordination des Entwicklungsprozesses und damit verbunden die Strukturen im Projekt stärker zu formalisieren und zu zentralisieren, um eine koordinierte Entwicklung der Software zu ermöglichen.

7.2.2 Schulungsprozess

Nachdem zuvor im Zuge der Differenzierung in kommerzielle und nicht-kommerzielle Open Source Projekte unterschiedliche Arten von Herauforderungen an die Projekte identifiziert wurden, soll die Betrachtung der Mechanismen und Aktivitäten zur Schulung neuer und bestehender Entwickler dazu dienen, gegebenenfalls unterschiedliche Vorgehensweisen im Umgang mit diesen Herausforderungen zu beleuchten.

Die Annahme, dass mit der kommerziellen Orientierung gleichzeitig auch eine prototypische Ausformung des Schulungsprozesses in Open Source Projekten einhergeht, wird dabei von den gewonnenen Daten allerdings nicht bestätigt.

So unterscheiden sich die beiden untersuchten kommerziellen Open Source Projekte, das Magnolia- und das OpenCms-Projekt, erheblich im Umfang der Maßnahmen zum Support ihrer Benutzer und Entwickler. Während der Projektadministrator des

[167] siehe Kapitel 7.2.2 „Schulungsprozess“

OpenCms-Projektes wöchentlich lediglich etwa 1 Stunde mit der Beantwortung von technischen Fragen in den Entwicklermailinglisten verbringt, investieren die beiden an der Umfrage beteiligten Projektadministratoren des Magnolia-Projektes dagegen etwa 2 bzw. sogar über 10 Stunden wöchentlich mit der Beantwortung von technischen Fragen, Fragen zur Entwicklungsrichtung sowie zu Beiträgen der beteiligten Co-Entwickler. Zudem bietet das Magnolia-Projekt den Benutzern und Entwicklern neben einem Bug-Tracking-System zur Auszeichnung benötigter Hilfestellungen sowie einer Beschreibung der API auch weiterführende Tutorials als Hilfestellung an.

Tabelle 13: Etablierte Schulungsmechanismen im OpenCms- und im Magnolia-Projekt

	Inv.Zt.	*API*	*B.T.*	*T.F.*	*E.R.*	*N.K.*	*Tut.*
OpenCms	1	ja	ja	ja	nein	nein	nein
Magnolia	>10/2	ja	ja	ja	ja	ja	ja

Inv.Zt. = Wöchentlich investierte Zeit der Projektadministratoren (in Std.)
API = API-Spezifikation
B.T. = Bug-Tracker
T.F. = Beantwortung technischer Fragen
E.R. = Beantwortung von Fragen zur Entwicklungsrichtung
N.K. = Beantwortung von Fragen zu neuen Beiträgen
Tut. = Tutorials

Dies zeigt, dass der in Open Source Projekten etablierte Schulungsprozess eine individuelle Ausprägung besitzt, die nicht allein durch die kommerzielle Ausrichtung der Projekte erklärt werden kann. Eine mögliche Interpretation der Unterschiede zwischen den beiden Open Source Projekten besteht darin, dass das Magnolia-Projekt versucht, dem in der Umfrage genannte Zukunftsrisiko der Gewinnung neuer Entwickler bewusst entgegenzusteuern, während das OpenCms-Projekt bereits einen größeren Kern an Entwicklern besitzt.

Neben der Verdeutlichung der Unterschiede zwischen den beiden kommerziellen Open Source Projekten liefern die Umfrageresultate auch eine mögliche Erklärung für die Einschätzung des Administrators des Drupal-Projektes, dass die Fähigkeiten bzw. die Schulung neuer Entwickler die zukünftige Weiterentwicklung der Projektsoftware hemmen könnten. Vergleicht man das Spektrum der unterstützenden Maßnahmen der vier nicht-kommerziellen Open Source Projekte, die zur Verringerung der Abhängigkeiten von einzelnen Kernentwicklern besonders an der Heranführung neuer Entwickler an den Anwendungskern interessiert sein müssen, so besitzt Drupal zumindest im Verhältnis zu Mambo und Typo3 eine schmale Bandbreite an Strukturen und Aktivitäten zur Schulung von Benutzern und Entwicklern.

Tabelle 14: Etablierte Schulungsmechanismen in den nicht-kommerziellen OSPs

	Drupal	*Mambo*	*TGS CMS*	*Typo3*
Inv.Zt.	3-5	3-5	>10	2
API	ja	ja	nein	ja
B.T.	ja	ja	ja	ja
T.F.	ja	ja	nein	ja
E.R.	ja	nein	nein	ja
N.K.	nein	nein	nein	ja
FAQ	nein	ja	ja	nein
Cod.R.	**nein**	**ja**	**ja**	**ja**
F.K.	**nein**	**ja**	**ja**	**ja**
Tut.	nein	ja	nein	nein
Com.G.	nein	nein	nein	ja

Inv.Zt. = Wöchentlich investierte Zeit der Projektadministratoren (in Std.)
API = API-Spezifikation
B.T. = Bug-Tracker
T.F. = Beantwortung technischer Fragen
E.R. = Beantwortung von Fragen zur Entwicklungsrichtung
N.K. = Beantwortung von Fragen zu neuen Beiträgen
FAQ = Frequently Asked Questions
Cod.R. = Codierrichtlinien
F.K. = Feedback zu eingegangenen Codierbeiträgen
Tut. = Tutorials
Com.G. = Commiter Guide

Dabei scheinen insbesondere quellcodeorientierte Hilfestellungen wie Codierrichtlinien oder das Anbieten eines Feedbacks zu eingehenden Codierbeiträgen wichtige unterstützende Maßnahmen für die Co-Entwickler der Software zu sein. Dies wird deutlich, da das Drupal-Projekt das einzige der vier nicht-kommerziellen Open Source Projekte ist, dass zum einen sowohl die Fähigkeiten als auch die Ausbildung neuer Entwickler als potentielles Problem für die zukünftige Entwicklung des Open Source Projektes nennt, und zum anderen keine der beiden genannten quellcodeorientierten Hilfestellungen implementiert. Dementsprechend bilden solche Maßnahmen eine wichtige Komponente im Hinblick auf die Zukunftssicherheit von Open Source Projekten.

Die dritte Besonderheit, die aus den Umfrageresultaten im Hinblick auf den Schulungsprozess in Open Source Projekten deutlich wird, besteht darin, dass das

Anbieten von Hilfestellungen sowie das Kommunizieren mit potentiellen Entwicklern und Benutzern insbesondere in der Startphase eines Open Source Projektes für die Projektadministratoren sehr aufwendig ist.

Dies wird deutlich an den Umfrageresultaten von TGS-CMS. Entsprechend der SVN-Daten besteht das Open Source Projekt seit November 2006. Dies bedeutet im Wesentlichen zwei Dinge. Erstens muss das Open Source Projekt zunächst eine Gemeinschaft von Benutzern um sich herum etablieren, die im Zuge einer zunehmenden Involvierung in die Software sowie in die Prozesse des Projektes auch aktiv selbst anstehende Aufgaben übernehmen. Zweitens muss der Aufwand zur Akquirierung und Unterstützung potentieller Teilnehmer zu Beginn ausschließlich von den Projektadministratoren selbst geleistet werden.

Beide Aspekte spiegeln sich auch in den Ergebnissen der Datenerhebung zum TGS-CMS-Projekt wider. Als mögliches Problem, das die zukünftige Entwicklung des Open Source Projektes behindern könnte, nennt der Projektadministrator die geringe Anzahl an Benutzern der Projektsoftware. Um dem entgegenzuwirken investiert er wöchentlich mehr als 10 Stunden für die Unterstützung bestehender Benutzer und potentieller Entwickler. Ungewöhnlich ist in diesem Zusammenhang, dass dies nicht über das sonst übliche Medium der Mailinglisten geschieht, sondern durch direkten Kontakt zu den Kernentwicklern des Projektes.

Zusammenfassend sind, basierend auf den Resultaten der Datenerhebung, drei wesentliche Punkte zum Einfluss eines etablierten Schulungsprozesses auf die Zukunftssicherheit von Open Source Projekten zu nennen.

Erstens besteht in Abhängigkeit von der kommerziellen Ausrichtung von Open Source Projekten keine prototypische inhaltliche oder umfangsbezogene Ausformung des Schulungsprozesses, sondern die Ausformung bildet ein individuelles Projektmerkmal.

Zweitens verringert ein breites Spektrum an etablierten Strukturen und Aktivitäten zur Unterstützung der Benutzer und Entwickler das Risiko, dass die Fähigkeiten oder die Ausbildung neuer Entwickler zu einem Problem werden können, das die zukünftige Weiterentwicklung der Projektsoftware behindert. In diesem Zusammenhang sind vor allem quellcodebezogene Hilfestellungen wie Codierrichtlinien oder ein Feedback zu eingehenden Codierbeiträgen zu nennen.

Drittens schließlich ist insbesondere in der Startphase eines Open Source Projektes ein intensives Engagement der Projektadministratoren erforderlich, um neue Benutzer und damit verbunden potentielle Entwickler der Software zu akquirieren und zu unterstützen.

7.2.3 Projektgröße/Projektaktivität

Der erste Zusammenhang, der deutlich wird, betrachtet man die Resultate der Analysen der Quellcodeverwaltungssysteme, ist, dass die Anzahl der Entwickler und die Größe der Software in einem direkten Zusammenhang stehen.

Tabelle 15: Verhältnis der Entwicklerzahl zum resultierenden Output

Open Source Projekt	*Lines of Code*	*# Entwickler*
TGS CMS	99.165	3
Mambo	261.516	15
Typo3	389.242	25
OpenCms	678.326	25
Magnolia	218.706	16
Magnolia-Core	36.582	8
Drupal	59.131	8

Darüber hinaus belegen die ermittelten Daten der Projekte allerdings keinen weiteren Zusammenhang zwischen der aktuellen Community- bzw. Anwendungsgröße und den Einschätzungen der Projektadministratoren in Bezug auf die zukünftige Projektgröße bzw. die zukünftige Softwarequalität.

Die Angaben aller beteiligten Projektadministratoren ergeben, dass sich die Community in fünf Jahren wesentlich vergrößern sowie die Qualität der Projektsoftware verbessern wird. Dies steht offensichtlich in einem Zusammenhang mit der insgesamt kontinuierlichen Entwicklung in den beteiligten Open Source Projekten. Bis auf TGS CMS, das noch ein sehr junges Content Management System ist, bestehen alle beteiligten WCM-Systeme seit mindestens 15 (Mambo) und durchschnittlich etwa 50 Monaten, betrachtet man lediglich die Aktivitäten im CVS bzw. SVN-System. Zieht man darüber hinaus die historischen Angaben auf den Webseiten der fünf Open Source Projekte hinzu, so ergibt sich sogar ein Mindestalter von 40 Monaten. Eine präzise Angabe ist bei einigen der Projekte allerdings schwierig, da Mambo, Magnolia und Typo3 vor ihrer Zeit als offenes Softwareprojekt proprietär von Unternehmen entwickelt wurden.

Verfolgt man in den entsprechenden Zeiträumen den Verlauf der Entwicklung der jeweiligen Projektsoftware im CVS bzw. SVN-System, so weisen alle fünf historisch betrachteten Open Source Projekte ein stetiges Wachstum auf. Zudem signalisieren die Ergebnisse der Datenerhebung der letzten 12 Monate eine kontinuierliche Aktivität der

beteiligten Entwickler. Ausnahmen bestehen in der Regel nur in der Folge neuer Releaseveröffentlichungen, im Zuge derer teilweise Phasen geringerer Aktivität zu verzeichnen sind.

Die durchschnittliche Anzahl der Commits/Tag in den letzten 12 Monaten beträgt in den fünf Open Source Projekten dabei mindestens 0,6 (OpenCms) im Mittel sogar über 2,3. Dies bedeutet, dass alle fünf der in den letzten 12 Monaten existierenden untersuchten Open Source Projekte durchschnittlich fast täglich (OpenCms) bzw. sogar mehr als einmal täglich Quellcode in das Quellcodeverzeichnis integriert haben. Diese Tendenz scheint ein Indiz für die durchgängig positive Zukunftseinschätzung der Projektadministratoren zu sein.

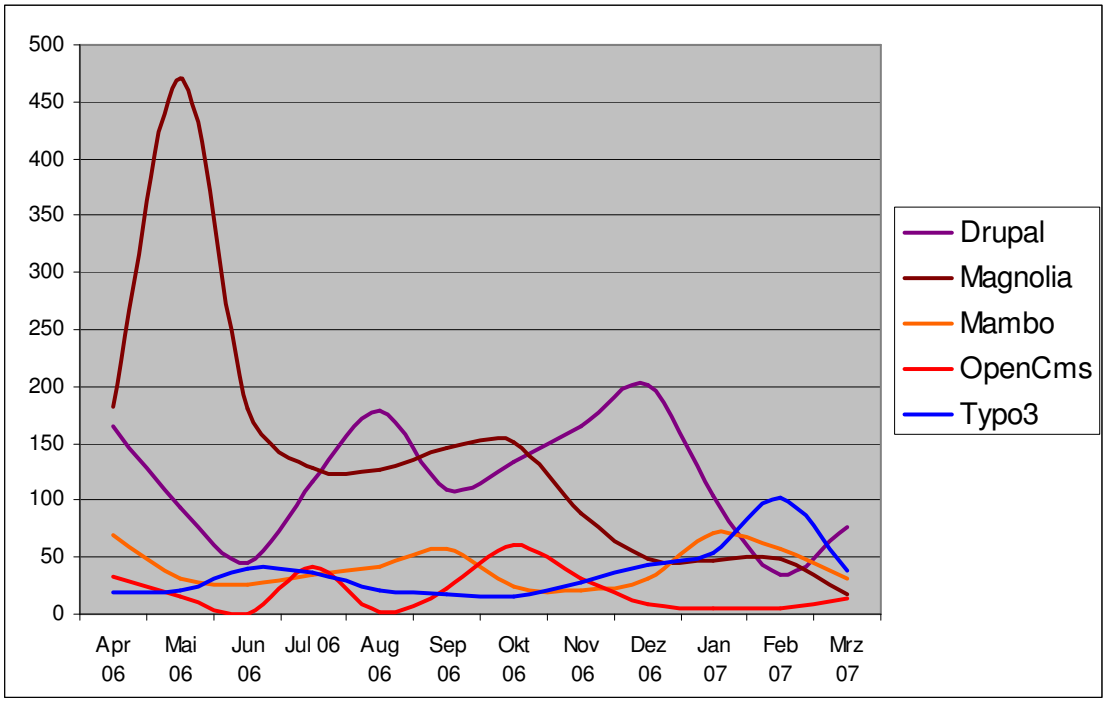

Abbildung 15: Verlauf der Commitzahlen im Quellcodeverzeichnis von April 2006 bis März 2007

Ein dritter Zusammenhang, den die CVS- bzw. SVN-Daten belegen, besteht in der klar vorhandenen Beziehung zwischen dem Verhältnis von geänderten bzw. hinzugefügten Quellcodezeilen und der zukünftigen Softwarequalität. Drei der sechs insgesamt beteiligten Open Source Projekte signalisieren, dass ihre Software in fünf Jahren „mature“ sein wird. Dies sind Drupal, Mambo und Magnolia. Diese drei Projekte besitzen zudem den höchsten Quotienten aus geänderten und hinzugefügten Codezeilen in den letzten drei Monaten. Daraus folgt, dass die Projekte vor allem Änderungen im Gegensatz zu Programmerweiterungen vornehmen, also im Wesentlichen Korrekturen leisten. Eine solche Tendenz scheint dementsprechend ein klares Zeichen für den Reifegrad einer Software zu sein.

Tabelle 16: Zusammenhang zwischen dem Quotienten LOC changed/LOC added in den letzten drei Monaten und der zukünftigen Softwarequalität

	LOC changed	*LOC added*	*changed/added*	*Zuk.SW.Qualität*
Drupal	**7.841**	**1.611**	**4,87**	**mature**
Magnolia	**4.695**	**1.226**	**3,83**	**higher/mature**
Mambo	**84.050**	**18.652**	**4,51**	**mature**
OpenCms	9.331	7.341	1,27	higher
TGS CMS	58.674	54.307	1,08	higher
Typo3	11.131	6.767	1,64	higher

Die erhaltenen Resultate von Drupal und Magnolia suggerieren außerdem, dass auch der Quotient aus der Anzahl der hinzugefügten Quellcodezeilen und der Zahl der durchgeführten Commits ein vergleichbarer Indikator sein kann. Dementsprechend deutet eine geringe Zahl hinzugefügter Codezeilen bei einer vergleichsweise hohen Anzahl von Commits ebenfalls auf eine Tendenz, dass vor allem Programmkorrekturen vorgenommen werden. Diese Ergebnisse werden allerdings nicht durch die gewonnenen Daten zum Mambo-Projekt bestätigt.

Tabelle 17: Zusammenhang zwischen dem Quotienten LOC added/#Commits in den letzten drei Monaten und der zukünftigen Softwarequalität

	LOC added	*# Commits*	*LOC ad./Com.*	*Zuk.SW.Qualität*
Drupal	**1.611**	**215**	**7,49**	**mature**
Magnolia	**1.226**	**114**	**10,75**	**higher/mature**
Mambo	**18.652**	**161**	**115,85**	**mature**
OpenCms	7.341	26	282,35	higher
TGS CMS	54.307	63	862,02	higher
Typo3	6.767	193	35,06	higher

Zusammenfassend belegen die Ergebnisse der CVS- und SVN-Analysen zu den sechs untersuchten Open Source Projekten, dass ein Zusammenhang zwischen der Anzahl der beteiligten Entwickler und der tatsächlich produzierten Zahl der Quellcodezeilen eines Open Source Projektes besteht. Eine Beziehung zur Zukunftssicherheit der Open Source Projekte belegen die Resultate allerdings nicht. Es zeigt sich jedoch, dass eine

vergleichsweise hohe Zahl von Entwicklern zumindest ein Indiz für den Umfang der entstehenden Anwendung bildet.

Darüber hinaus weisen alle untersuchten Projekte einen kontinuierlichen Prozess der Weiterentwicklung der Software auf. Abgesehen vom TGS-CMS-Projekt streckt sich dieser Prozess dabei über einen Zeitraum von mindestens 15 Monaten (CVS/SVN) hinaus. Parallel dazu schätzen die Administratoren der beteiligten Projekte die zukünftige Entwicklung der Community sowie der Software selbst als sehr positiv ein, sodass im Rahmen der Arbeit davon ausgegangen wird, dass kontinuierliche Aktivität von Open Source Projekten ein weiteres Indiz für die Zukunftssicherheit derselben ist.

Schließlich belegen die Ergebnisse der Datenerhebung, dass eine hohe Zahl von geänderten im Vergleich zu hinzugefügten Quellcodezeilen einen Indikator für den zukünftigen Reifegrad einer Software darstellt.

7.2.4 Projektideologie/Projektziele

Im Hinblick auf bestehende Verbindungen zwischen der Existenz einer Kernideologie und der Zukunftssicherheit von Open Source Projekten belegen die Resultate der Datenerhebung keine existierenden Zusammenhänge.

Der Grundgedanke des Rahmenwerkes von Collins und Porras, an dem sich die vorgenommene Einordnung der Kernwerte sowie der Daseinsberechtigung der Projekte orientiert, besteht in der Trennung der bloßen Existenz der beiden genannten Aspekte von ihrer konkreten Manifestierung. Vor diesem Hintergrund zielt die getrennte Erfassung der Kernideologie von Open Source Projekten durch die Befragung der Projektadministratoren einerseits bzw. die Untersuchung der Webseite andererseits auf die Berücksichtigung genau dieser Differenzierung. Während die Befragung der Projektadministratoren ermittelt, inwieweit überhaupt eine Kernideologie in dem jeweiligen Open Source Projekt existiert, ist mit der Untersuchung der Webseite das Ziel verbunden, den Grad der Verwurzelung der entsprechenden Werte und Normen in der Community abzuschätzen.

Obwohl alle sechs analysierten Open Source Projekte basierend auf den Einschätzungen der Projektadministratoren eine hohe Zukunftssicherheit aufweisen, formulieren lediglich Magnolia und Drupal auf ihrer Webseite eine Menge von Kernwerten, die mit der von Collins und Porras intendierten Semantik von überdauernden, zeitlosen Leitwerten zumindest vergleichbar sind. Dies deutet bereits darauf hin, dass das Prinzip der Kernideologie in der Mehrzahl der untersuchten Open Source Projekte nicht etabliert ist. Zudem bestehen teilweise Differenzen zwischen den Angaben der Projektadministratoren und den auf den Webseiten gefundenen Informationen, sodass darüber hinaus in Frage gestellt werden muss, ob eine Nennung von Werten auf der Webseite automatisch mit einer tiefen Verwurzelung der Prinzipien in der Community

einhergeht. In diesem Zusammenhang wird die limitierte Aussagekraft der Erfassungsmethode der Kernideologie über die Befragung der Projektadministratoren bzw. die Untersuchung der Webseiten deutlich. Zwar sind sowohl die Aussagen der Projektadministratoren als auch weitergehende Formulierungen einer Kernideologie auf der Webseite Indizien für den Grad der Verwurzelung der Prinzipien in der Community, allerdings keine hinreichenden Kriterien.

Ähnliches gilt auch für die Erfassung der Ziele der Open Source Projekte. Neben der Fokussierung auf das Wesen einer Organisation beinhaltet das Rahmenwerk von Collins und Porras darüber hinaus fortschrittsorientierte Maßnahmen als zweiten wichtigen Pfeiler visionärer Unternehmen. Übertragen auf Open Source Projekte ist eine solche Maßnahme vor allem das kontinuierliche Aufstellen ambitionierter Ziele. Auch hier gilt, dass die Erfassung der Ziele eines Open Source Projektes durch die Befragung der Projektadministratoren sowie die Untersuchung der Webseite lediglich Indizien dafür liefern, inwieweit sich das jeweilige Open Source Projekt diesen Zielen verpflichtet fühlt.

Allerdings belegen die Resultate der Datenerhebung im Hinblick auf ambitionierte Ziele im Gegensatz zur Existenz einer Kernideologie eine wesentlich stärkere Verbreitung in Open Source Projekten. So geben die Administratoren von vier der sechs beteiligten Open Source Projekten an, dass eines der langfristigen Ziele des Projektes darin besteht, das jeweilige Content Management System zum weltweiten Standardsystem zu entwickeln. Zudem arbeitet das Typo3-Projekt aktuell an einer völligen Neuentwicklung des Anwendungskerns, um langfristig konkurrenzfähig zu bleiben. Einen ähnlichen Schritt hat das Mambo-Projekt bereits im letzten Jahr vollzogen. Dies zeigt, dass die Projekte einerseits ambitioniert, andererseits aber auch wie von Collins und Porras gefordert bereit sind, eine völlige Neuorientierung zu wagen. Schließlich besitzen vier der sechs Open Source Projekte eine offizielle Roadmap, deren Umfang und Detailgrad sich allerdings erheblich unterscheidet.

Zusammenfassend belegen die ermittelten Daten keine Beziehungen zwischen der Zukunftssicherheit von Open Source Projekten und der expliziten Formulierung einer Kernideologie.

Das Problem besteht in diesem Zusammenhang allerdings auch darin, dass die Art der Erfassung des Aspektes durch die genannten Datenquellen keine Messung der tatsächlichen Verwurzelung der jeweiligen Community mit den von ihr formulierten Werten und Normen ermöglicht. Hierfür ist eine genauere Betrachtung des gesamten Entwicklungsprozesses über einen längeren Zeitraum erforderlich. Eine mögliche Methode hierzu kann die Analyse der Mailinglistenarchive über einen längeren Zeitraum sein, da die Mailinglisten in der Regel das zentrale Mittel der Kommunikation und Koordination in Open Source Projekten bilden.

Im Hinblick auf die Ziele von Open Source Projekten signalisieren die Projektdaten dagegen, dass gewachsene Projekte durchaus ambitionierte Ziele besitzen und diese auch verfolgen, wie die Neuimplementierungen des Mambo- sowie des Typo3-Kerns zeigen. Darüber hinaus besteht aber auch im Bereich der Projektziele das Problem, mit den genutzten Datenquellen die tatsächliche Verwurzelung der Community mit diesen Zielen aufzuzeigen.

Unabhängig von den Ergebnissen der Datenerhebung geht der Autor der Arbeit davon aus, dass die Formulierung und Etablierung einer Kernideologie in Open Source Projekten langfristige Unterschiede in der Entwicklung der Projekte bewirken.

8 Zusammenfassung und Ausblick

Die hier vorliegende Arbeit stellt eine methodische Vorgehensweise zur Evaluation von Open Source Projekten vor. Die Zielgruppe der Evaluation besteht dabei aus Unternehmen, die Open Source Software bereits einsetzen oder planen dies zu tun.

Strukturell orientiert sich die vorgeschlagene Evaluationsmethode dazu an dem Rahmenwerk der Deutschen Gesellschaft für Evaluation (DeGEval). Dies bedeutet inhaltlich im ersten Schritt die Durchführung einer detaillierten Analyse der Situation in den Open Source Projekten selbst als auch in den Unternehmen als intendierte Zielgruppe. Im Zuge dieser Analyse wird auch eine Umfrage zu den Risiken und Vorteilen eines Einsatzes von Open Source Software in Unternehmen durchgeführt. Aus den Ergebnissen dieser Umfrage werden dann unternehmensrelevante Anforderungen extrahiert, die als Grundlage für die weiteren Evaluationsschritte dienen.

Im zweiten Schritt der Evaluation erfolgt die Zerlegung des Evaluationsgegenstandes auf messbare Metriken. Dazu wird im Rahmen der Arbeit nach dem Goal-Question-Metric-Ansatz, einer iterativen Top-Down-Methode, verfahren. Als Grundlage dient dabei eine bereits bestehende allgemeine Zerlegung von Open Source Projekten nach Cruz *et al.* Diese wird aufgrund bestehender Schwachstellen um einen konkreteren Kontext erweitert. Dieser Kontext besteht zum einen in dem Zielfaktor „Zukunftssicherheit“, der als relevante Anforderung von Unternehmen aus der durchgeführten Umfrage hervorgeht. Zum anderen erfolgt die Fokussierung auf den Anwendungsbereich „Web Content Management Systeme“.

Im Zuge der Zerlegung des ausgewählten Zielfaktors werden dabei vier wesentliche Einflussfaktoren identifiziert, die die Ausprägung der Zukunftssicherheit in Open Source Projekten beeinflussen. Die vier Faktoren sind die kommerzielle Ausrichtung eines Open Source Projektes, ein etablierter Schulungsprozess zur Ausbildung neuer Entwickler, die Projektgröße und Projektaktivität sowie die Existenz einer Kernideologie.

Im dritten Schritt erfolgt schließlich die Validierung der entwickelten Zerlegung anhand einer Fallstudie von sechs ausgewählten offenen Web Content Management Systemen. Die sechs Open Source Projekte sind Drupal, Magnolia, Mambo, OpenCms, TGS CMS und Typo3. Als Daten- und Informationsquellen dienen dabei die Webseiten der Projekte als wesentlichste Präsentationsschnittstelle zu Benutzern und Entwicklern, CVS- bzw. SVN-Daten zur Analyse der Entwicklung des Quellcodes und eine Umfrage an die Administratoren der Projekte zur Ermittlung zusätzlicher Informationen, die nicht auf den Webseiten der Projekte ersichtlich sind. Die Resultate der Datenerhebung

belegen vor allem vier Beziehungen zwischen den vier identifizierten Teilaspekten und der Zukunftssicherheit von Open Source Projekten:

Erstens wird die Art der zu bewältigenden Herausforderungen zu einem großen Teil durch die kommerzielle Orientierung von Open Source Projekten bestimmt. Demnach bestehen die Ziele kommerzieller Open Source Projekte in einer großen Benutzer- bzw. potentiellen Kundengemeinschaft, einer breiten Co-Entwicklercommunity zur Verringerung des Pflege- und Wartungsaufwandes für die Kernentwickler sowie in der Etablierung einer stabilen Zahl von Kernentwicklern, deren Anzahl der Komplexität der Anwendung gerecht wird, aber darüber hinaus aus finanziellen Aspekten nicht zu groß sein sollte. Nicht-kommerzielle Open Source Projekte müssen dagegen im Zuge der Freiwilligkeit der Beteiligung vor allem auf die Verringerung der Abhängigkeiten von einzelnen Personen sowie die Koordinierung des Entwicklungsaufwandes achten. Vor diesem Hintergrund muss es das Ziel nicht-kommerzieller Open Source Projekte sein, Co-Entwickler an den Kern einer Anwendung heranzuführen sowie Mechanismen zur Koordinierung des Entwicklungsprozesses zu etablieren. Im Zusammenhang mit dem zweiten Aspekt suchen vor allem größere Open Source Projekte häufig Anlehnung an eine gemeinnützige Organisation im Projektumfeld.

Zweitens belegen die Ergebnisse der Datenerhebung die Wichtigkeit quellcodeorientierter Unterstützungsmaßnahmen für Benutzer und Entwickler. Demnach verringern vor allem Maßnahmen wie die Aufstellung von Codierrichtlinien und das Feedback der Kernentwickler zu eingehenden Codierbeiträgen das Risiko, dass die Ausbildung bzw. die Fähigkeiten neuer Entwickler zu einem zukünftigen Problem werden können.

Drittens scheint die langfristige kontinuierliche Weiterentwicklung einer Projektsoftware mit positiven Zukunftsaussichten verbunden zu sein. Bis auf TGS CMS bestehen alle der beteiligten Open Source Projekte bereits seit mindestens 15 (CVS/SVN) bzw. 40 Monaten (Webseite). In diesem Zeitraum hat sich die Software aller fünf Projekte, gemessen an der Zahl der Codezeilen, kontinuierlich weiterentwickelt. Parallel dazu deuten die Einschätzungen aller Projektadministratoren auf eine positive zukünftige Entwicklung der Projekte hin.

Viertens schließlich besteht ein deutlicher Zusammenhang zwischen dem Verhältnis von geänderten und hinzugefügten Codezeilen in den letzten drei Monaten und der zukünftigen Qualität der Software. Drei der sechs beteiligten Open Source Projekte geben an, dass die Projektsoftware in fünf Jahren „mature“ sein wird. Diese fünf Projekte besitzen zudem den höchsten Quotienten aus geänderten und hinzugefügten Quellcodezeilen. Folglich ist ein hoher Wert dieses Koeffizienten ofensichtlich ein starkes Indiz, dass die Software vorwiegend geändert im Gegensatz zu erweitert wird, und signalisiert somit den Reifegrad der Anwendung.

Nicht bestätigt wurde dagegen die vermutete Verbindung zwischen der Existenz und Ausprägung einer Kernideologie und der Zukunftssicherheit von Open Source Projekten. Allerdings ist in diesem Zusammenhang die Erfassung der Kernideologie über die Befragung der Projektadministratoren sowie die Webseiten der Projekte zu oberflächlich und belegt nicht ausreichend die geforderte starke Verwurzelung der Open Source Projekte mit den gegebenenfalls vorhanden Kernwerten und Prinzipien.

Da der Autor dennoch davon ausgeht, dass die Existenz einer Kernideologie die Zukunftssicherheit von Open Source Projekten beeinflusst, ist eine genauere Analyse und Erfassung des Aspektes eine wesentliche Erweiterung der vorgenommenen Zerlegung des Zielfaktors „Zukunftssicherheit“ für den zweiten Iterationsschritt des GQM-Verfahrens. Eine solche Erfassung muss einerseits längerfristige Daten zur Ausprägung einer Kernideologie liefern. Andererseits muss sie auch den Grad der Verwurzelung in den bestehenden Prozessen messbar machen. Eine Möglichkeit zur Durchführung einer solchen Analyse ist die Untersuchung der Mailinglistenarchive über einen längeren Zeitraum. Da die Mailinglisten in der Regel das wesentlichste Medium zur Kommunikation in Open Source Projekten sind, könnte folglich zumindest der Grad der Verwurzelung innerhalb der Entscheidungsprozesse abgebildet werden.

Anhang A: Eigenschaften der unterschiedlichen Lizenzkategorien

In der Folge werden tabellarisch die Eigenschaften der vier wesentlichsten Lizenzkategorien nach Jaeger und Metzger [Jaeger2002] sowie einige der dazugehörigen Lizenzen aufgeführt.

A.1 „Copyleft“-Lizenzen

„Copyleft“-Lizenzen sind geprägt durch die Verpflichtung der Benutzer, im Fall einer Weiterverteilung der Originalsoftware oder einer Modifikationen diese erneut zu den Bedingungen der entsprechenden „Copyleft“-Lizenz zu verbreiten.

Tabelle 18: Eigenschaften von „Copyleft“-Lizenzen

	„Copyleft“-Lizenzen
Einfaches Nutzungsrecht	Die Benutzer der Software erhalten das Recht, die Software zu nutzen, zu vervielfältigen und unveränderte Versionen zu denselben Lizenzbedingungen weiterzuverbreiten. Dieses Recht entstammt nach deutschem Urheberrecht in jedem Fall direkt vom Urheber und wird als „Einfaches Nutzungsrecht an jedermann“ bezeichnet.
Verbreitung im Objektcode	Eine Verbreitung der Software im Objektcode ist gestattet, allerdings an die Mitlieferung des Softwarequellcodes direkt oder in Form eines schriftlichen Angebots zur Lieferung des Quellcodes gebunden.
Verbreitung modifizierter Versionen	Die Verbreitung veränderter Versionen der Software ist gestattet, muss aber zu den Bedingungen der „Copyleft“-Lizenz erfolgen. Eine Verbreitung der Software liegt vor, sobald der Kreis der Empfänger nicht durch gegenseitige Beziehungen oder durch Beziehungen zum Urheber miteinander verbunden ist. Eine Veränderung der Software liegt vor, wenn zum einen die hinzugefügten Bestandteile gemeinsam mit dem Originalquellcode verbreitet werden und zum anderen technische und funktionale Kriterien darüber hinaus auf ein abgeleitetes Werk hindeuten. Technische Kriterien, die für ein abgeleitetes Werk sprechen, sind • Gemeinsames Laden von bestehendem und modifiziertem Quellcode • Ausführung im gleichen Adressraum Funktionale Kriterien, die für ein abgeleitetes Werk sprechen, sind • die Erfüllung der Funktionen eines Programms entsprechend der Betrachtungsweise eines Durchschnittsfachmanns (komplexe Software wie Betriebsysteme oder Office-Anwendungen werden in der Regel nicht als ein Programm betrachtet)

Verbreitung in einer Kombination mit proprietärer SW	Die getrennte Verbreitung proprietärer und „Copyleft“-Software ist problemlos möglich. Im Fall einer gemeinsamen Verbreitung greifen die Bestimmungen der Lizenzen, sodass die resultierende Anwendung ebenfalls zu den Bestimmungen der „Copyleft“-Lizenz verbreitet werden muss, sofern ein abgeleitetes Werk vorliegt.
Weitere Pflichten	Zusätzliche Pflichten der Benutzer im Fall einer Verbreitung der Software bestehen in • der Mitlieferung der Lizenzbestimmungen • dem Verbot der Veränderung von Urhebervermerken • der klaren Deklarierung von vorgenommenen Änderungen
Lizenzen der Kategorie	• GNU General Public License (GPL) • GNU Lesser General Public License (LGPL)

A.2 „BSD“-artige Lizenzen

„BSD“-artige Lizenzen zeichnen sich vor allem durch die umfangreichen Freiheiten aus, die sie dem Benutzer der Software gewähren.

Tabelle 19: Eigenschaften „BSD“-artiger Lizenzen

	„BSD“-artige Lizenzen
Einfaches Nutzungsrecht	Die Benutzer der Software erhalten das Recht, die Software zu nutzen, zu vervielfältigen und unveränderte Versionen zu verbreiten.
Verbreitung im Objektcode	Bei einer Verbreitung im Objektcode müssen der Urhebervermerk, die Lizenzbestimmungen und der Haftungsausschluss in der Dokumentation oder anderen mitgelieferten Materialien enthalten sein
Verbreitung modifizierter Versionen	Die Verbreitung veränderter Versionen ist lediglich an das Verbot der Nennung des ursprünglichen Urhebers gebunden. Der Quellcode muss darüber hinaus nicht offen gelegt werden.
Verbreitung in einer Kombination mit proprietärer SW	Genau wie die Verbreitung modifizierter Versionen ist auch die Verbreitung einer Kombination von proprietärer Software und Software unter einer „BSD“-artigen Lizenz problemlos möglich
Besondere Pflichten	Zusätzliche Pflichten der Benutzer im Fall einer Verbreitung der Software bestehen in • der Mitlieferung der Lizenzbestimmungen, des Urhebervermerks und des Haftungsausschlusses • der Aufnahme eines Hinweises auf den Urheber in Werbematerialien (nur einige der Lizenzen)
Lizenzen der Kategorie	• BSD Copyright-Lizenz • Apache Software License Version 1.1 • X-Lizenz

A.3 „Mozilla“-artige Lizenzen

„Mozilla“-artige Lizenzen zeichnen sich dadurch aus, dass sie einen Kompromissweg zwischen den strikten „Copyleft“-Lizenzen und den freizügigen „BSD“-artigen Lizenzen einschlagen.

Tabelle 20: Eigenschaften „Mozilla“-artiger Lizenzen

	„Mozilla“-artige Lizenzen
Einfaches Nutzungsrecht	„Mozilla“-artige Lizenzen differenzieren bei der Vergabe der Nutzungsrechte zwischen der ursprüngliche Version einer Software und entwickelten Modifikationen. Während die Benutzer im ersten Fall analog zu den beiden zuvor genannten Lizenzkategorien ein einfaches Nutzungsrecht erhalten, so verlieren sie im Zuge der Verbreitung einer Modifikation der Originalsoftware die Rechte an eventuell betroffenen Patenten.
Verbreitung im Objektcode	Die Verbreitung der Software im Objektcode ist gestattet, der Quellcode muss allerdings entweder mitgeliefert oder auf einem üblichen Verbreitungsmedium zur Verfügung gestellt werden.
Verbreitung modifizierter Versionen	Modifizierte Versionen einer Software müssen im Fall einer Verbreitung wieder der jeweiligen „Mozilla“-artigen Lizenz unterstellt werden, sofern die vorgenommenen Änderungen bereits bestehende Dateien betreffen. Werden neue Dateien integriert, so können diese auch proprietär vertrieben werden.
Verbreitung in einer Kombination mit proprietärer SW	Die Verbreitung einer Kombination von proprietärer Software und Software unter einer „Mozilla“-artigen Lizenz ist gestattet, sofern die proprietären Bestandteile in separaten Dateien angeordnet sind und die übrigen Lizenzbedingungen erfüllt werden.
Besondere Pflichten	• Vorgenommene Veränderungen müssen klar dokumentiert werden • Rechte an betroffenen Patenten gelten nur für die Originalversion einer Software • Die Lizenzbestimmungen, der Urhebervermerk und der Haftungsausschluss müssen im Zuge einer Verbreitung der Software ebenfalls verbreitet werden
Lizenzen der Kategorie	• Mozilla Public License 1.1 • Interbase Public License Version 1.0 • ICS Open Source Public License Version 1.0 • Sun Public License Version 1.0

A.4 „Artistic“-Lizenzen

„Artistic“-Lizenzen sind gekennzeichnet durch die unterschiedlichen Rechtsfolgen, die sie entsprechend der Art der vorgenommenen Änderung an einer Software beinhalten.

Tabelle 21: Eigenschaften von „Artistic“-Lizenzen

	„Artistic“-Lizenzen
Einfaches Nutzungsrecht	Die Benutzer der Software erhalten das Recht, die Software zu nutzen, zu vervielfältigen und unveränderte Versionen zu verbreiten.
Verbreitung im Objektcode	Die Verbreitung der Software im Objektcode ist gestattet, sofern die Nutzung der Software für den Benutzer nicht erkennbar ist. Ansonsten muss eine der folgenden vier Bedingungen erfüllt sein: 1. Überstellung der Anwendung in die Public Domain 2. Verbreitung lediglich im eigenen Unternehmen 3. Umbenennung der Anwendung und klare Beschreibung der Unterschiede 4. Einigung über besondere Lizenzgestaltungen mit dem Softwareurheber
Verbreitung modifizierter Versionen	„Artistic“-Lizenzen differenzieren zwei Arten von Modifikationen. Dies sind zum einen wenig weit reichende Modifikationen wie Fehlerbeseitigungen. In diesem Fall ergeben sich aus einer Verbreitung der Software keine weiteren Verpflichtungen. Werden jedoch Veränderungen am Gesamtpaket vorgenommen, also zusätzliche Dateien hinzugefügt oder ausgetauscht, so muss eine der vier genannten Bedingungen erfüllt werden.
Verbreitung in einer Kombination mit proprietärer SW	siehe Verbreitung im Objektcode
Besondere Pflichten	• Die Nutzung des Urhebernamens für veränderte Softwareversionen ist untersagt
Lizenzen der Kategorie	• Perl Artistic License • Clarified Artistic License

Anhang B: Umfrage zum Thema „Risiken und Vorteile eines Einsatzes von Open Source Software“

Anhang B enthält Informationen zu der im Rahmen der Arbeit durchgeführten Unternehmensumfrage zu den Risiken und Vorteilen eines Einsatzes von Open Source Software in Unternehmen.

B.1 Fragenkatalog der Unternehmensumfrage

Im Folgenden werden die 14 inhaltlichen Fragen der durchgeführten Unternehmensumfrage aufgeführt. Im Fall, dass Unterschiede zwischen den Fragen an Anwendungsunternehmen und Softwareunternehmen bestehen, so sind diese Fragen entsprechend gekennzeichnet.

1. In welcher Branche arbeitet ihr Unternehmen?
2. Wie viele Mitarbeiter beschäftigt ihr Unternehmen?
   [] 1-10
   [] 11-100
   [] 101-1000
   [] über 1000
3. Wird in Ihrem Unternehmen bei der Entwicklung eigener Software auf Open Source Software zurückgegriffen? **[Softwareunternehmen]**
3. Wird in Ihrem Unternehmen im Rahmen der täglichen Arbeitsroutinen auf Open Source Software wie beispielsweise Browser oder Office-Pakete zurückgegriffen? **[Anwendungsunternehmen]**
4. Welche Art von Open Source Komponenten setzen Sie dabei ein und wie intensiv setzen Sie diese Komponenten ein? **[Softwareunternehmen]**
   [] Komplette Open Source Anwendungen als Teile/ Infrastruktur von Anwendungen (z.B. Datenbanken, Webserver …)
   [] Open Source Anwendungen als Grundlage für Modifikationen
   [] Open Source Frameworks
   [] Open Source Programmierbibliothken
   [] Codefragmente aus Open Source Anwendungen
   [] Code erzeugt durch Open Source Codegeneratoren
   [] Testsets/ Testsuiten
   [] Bilder, Grafiken, Artwork
   [] Sonstige

4. Welche Art von Open Source Software setzen Sie dabei ein und wie intensiv setzen Sie diese Software ein? **[Anwendungsunternehmen]**

   [] Betriebssysteme
   [] Datenbanken und Datenbanktools
   [] Webserver
   [] Applikationsserver
   [] Office-Anwendungen
   [] Grafik- oder Bildbearbeitungsprogramme
   [] Webbrowser
   [] Mail-Programme
   [] HTML-Editoren

5. In welchem Maße sind Sie in Open Source Projekte involviert? **[nur Softwareunternehmen]**

   [] Wir haben keinen Kontakt zu den Projekten
   [] Wir nutzen die Foren und Mailinglisten, um Antworten auf sich uns stellende Probleme zu erhalten
   [] Wir sind in den Foren und Mailinglisten auch als Antwortgeber aktiv
   [] Von uns entdeckte Fehler werden an die Projekte weitergeleitet
   [] Unsere Entwickler lösen bei uns auftretende Probleme und stellen ihre Lösungen als Patches zur Verfügung
   [] Wir sind aktiv an der Beseitigung von Fehlern beteiligt, die bisher nicht bei uns aufgetreten sind
   [] Wir sind aktiv an der Weiterentwicklung (und nicht ausschließlich der Fehlerbeseitigung) der Software beteiligt
   [] Unsere Entwickler helfen mit administrativen Aufgaben wie die Veröffentlichung von neuen Versionen und die Wartung der Webseite
   [] Unsere Entwickler sind aktiv an der Führung des Projektes beteiligt

6. Welche rechtlichen Risiken sehen Sie in Open Source Software und wie hoch schätzen Sie diese ein?

7. Welche technischen Risiken sehen Sie in Open Source Software und wie hoch schätzen Sie diese ein?

8. Welche wirtschaftlichen Risiken sehen Sie in Open Source Software und wie hoch schätzen Sie diese ein?

9. Welche sonstigen Risiken sehen Sie in Open Source Software und wie hoch schätzen Sie diese ein?

10. Welche rechtlichen Vorteile sehen Sie in Open Source Software und wie hoch schätzen Sie diese ein?

11. Welche technischen Vorteile sehen Sie in Open Source Software und wie hoch schätzen Sie diese ein?

12. Welche wirtschaftlichen Vorteile sehen Sie in Open Source Software und wie hoch schätzen Sie diese ein?
13. Welche sonstigen Vorteile sehen Sie in Open Source Software und wie hoch schätzen Sie diese ein?
14. Welche Informationen bezüglich Struktur und Organisation würden Sie sich von Open Source Projekten wünschen, um Bedenken gegenüber dem Einsatz von Open Source Software zu beseitigen?

B.2 Einsatzgebiete von OSS in Anwendungsunternehmen

Die folgende Tabelle liefert einen Überblick über die unterschiedlichen Einsatzgebiete von Open Source Software in den befragten Anwendungsunternehmen. Zudem werden die Anzahl der Nennungen des jeweiligen Aspektes durch Unternehmen sowie die durchschnittliche Intensität des Einsatzes in separaten Spalten angegeben. Die dafür verwendete Skala besteht aus fünf Items.

1 = wird nicht eingesetzt
2 = ein Einsatz ist zukünftig geplant
3 = mittelmäßig
4 = hoch
5 = sehr hoch

Tabelle 22: Einsatzgebiete von OSS in Anwendungsunternehmen

Einsatzgebiete	Anwendungen (Beispiele)	Anz. der Unternehmen	Intensität des Einsatzes[168]	Status
Applikationsserver	JBoss AS	6	signifikant	vorgegeben
Betriebssysteme	Linux	5	eingeschränkt	vorgegeben
Content Management Systeme	Typo3 ConPresso CMS	2	signifikant	ergänzt
Datenbanken oder Datenbanktools	MySQL PostgreSQL	6	eingeschränkt	vorgegeben
Entwicklungs-umgebungen	Eclipse Netbeans	3	signifikant	ergänzt
Grafikprogramme		3	eingeschränkt	vorgegeben
HTML-Editoren		3	testweise bis	vorgegeben

[168] Die verwendete Skala erlaubt eigentlich keine Durchschnittsbildung, sodass die Angaben lediglich Tendenzen darstellen

			einschränkt	
Mail-Programme		4	eingeschränkt	vorgegeben
Office-Anwendungen	OpenOffice	4	testweise	vorgegeben
Programmiersprache	Java	1	eingeschränkt	ergänzt
Qualitätssicherung	JMeter	1	eingeschränkt	ergänzt
Remote-Verbindung	Putty	1	signifikant	ergänzt
Scriptsprache	PHP, Perl	2	signifikant	ergänzt
Sicherheit	OpenSSL OpenSSH	1	signifikant	ergänzt
Systemüberwachung	Nagios	2	signifikant	ergänzt
Systemtools (andere)	Samba	2	signifikant	ergänzt
Verzeichnisdienste	OpenLDAP	1	signifikant	ergänzt
Versionskontrolle	TortoiseCVS TortoiseSVN	1	eingeschränkt	ergänzt
Webbrowser	Mozilla Firefox	7	eingeschränkt	vorgegeben
Webserver	Apache WS	7	signifikant	vorgegeben

B.3 Einsatzgebiete von OSS in Softwareunternehmen

Analog zu den Einsatzgebieten von Open Source Software in Anwendungsunternehmen liefert Tabelle 25 einen Überblick über die Art des Einsatzes offener Software in Softwareunternehmen. Entsprechend der in der Umfrage vorgegebenen Unterteilung beschreiben die untersuchten Aspekte eine eher strukturelle als inhaltliche Differenzierung.

Tabelle 23: Art des Einsatzes von OSS in Softwareunternehmen

	Anwendungen (Beispiele)	Anz. der Unternehmen	In Planung	Intensität des Einsatzes[169]
Komplette Anwendungen als Teile/Infrastruktur von Anwendungen		11	---	signifikant
Anwendungen als		9	2	eingeschränkt

[169] Die verwendete Skala erlaubt eigentlich keine Durchschnittsbildung, sodass die Angaben lediglich Tendenzen darstellen

Grundlage für Modifikationen				
OS Frameworks	JGraph	10	---	eingeschränkt
OS Programmier-bibliotheken	Xerces	12	1	eingeschränkt
Codefragmente aus OS Anwendungen		6	1	testweise bis eingeschränkt
Code aus OS Code-Genratoren		6	2	testweise bis eingeschränkt
Testsets/Testsuiten	JUnit	3	2	signifikant
Grafiken, Bilder, Artwork		4	1	eingeschränkt

B.4 Risiken eines Einsatzes von OSS aus Unternehmenssicht

Tabelle 26 zeigt eine genaue Auflistung der Risiken, die Software- und Anwendungsunternehmen mit einem Einsatz von Open Source Software verbinden. Zudem werden die Anzahl der Nennungen des jeweiligen Aspektes in Klammern dahinter sowie die durchschnittliche Bewertung in einer separaten Spalte angegeben. Die dafür verwendete Skala besteht aus fünf Items.

1 = sehr gering
2 = gering
3 = mittelmäßig
4 = hoch
5 = sehr hoch

Tabelle 24: Gegenüberstellung der Risiken eines Einsatzes von OSS aus Unternehmenssicht

	Softwareunternehmen (Anz. der Nennungen)	***Bew.***	Anwendungsunternehmen (Anz. der Nennungen)	***Bew.***
Technische Risiken	**Allgemein**	**3,2**	**Allgemein**	**2,8**
	Mangelhafte(s) Architektur/Design (1)	3	Heterogenität in verschiedenen Teilen/Bereichen einer Software, da unterschiedliche Benutzer arbeiten, deren Kontakt oft nicht so eng organisiert ist, wie bei der Entwicklung proprietärer Software (1)	3
	Aufgeblähter Programmcode (1)	3	Qualität und Wartung sind abhängig vom Projekt (1)	3
	Sicherheitslücken/Virengefahr (4)	2,5	Sicherheitslücken (1)	3
	Mangelhafte Dokumentation (4)	4,25	Mangelhafte Dokumentation (2)	3
	Mangelhaftes Fehlerhandling (1)	3	Inkompatibilität mit bestehender Software (1)	2

	Geringe Versionskontrolle (1)	4		
Wirtschaft-liche Risiken	**Allgemein**	**3,2**	**Allgemein**	**2,7**
	Einstellung oder mangelnde Weiterentwicklung/ Pflege des Projektes (5)	3,6	Einstellung des Projektes (1)	4
	kein/mangelhafter Support (2)	4,5	kein Support (2)	2,5
	keine klare Roadmap (2)	5	keine klare Roadmap (1)	4
	keine Garantie der Lösung (1)	4	Teilweise sehr hohe Anpassungskosten (TCO) (1)	3
	Hoher Aufwand, um geeignete Bibliotheken zu finden (1)	2	Supportkosten durch erhöhten Einsatz interner Mitarbeiter (1)	3
	Kosten durch Updates (1)	1	keine Wartungsverträge direkt mit dem Hersteller möglich (1)	4
	Hohe Total Cost of Ownership (1)	5	Offenlegung des eigenen Quellcodes bei Nutzung der LGPL (1)	1
	Probleme in der kommerziellen Vermarktung der eigenen Software bei der Integration von OS-Bestandteilen (1)	4		
	Konkurrenz für Lizenzprodukte (2)	3,5		
Rechtliche Risiken	**Allgemein**	**2,7**	**Allgemein**	**2,7**
	Legalisierung von Softwarepatenten in der EU (2)	3,5	Patentrecht (1)	3
	Andere patentrechtliche Fragen (1)	3	Rechte Dritter (1)	3
	Haftungsrisiko (2)	4	Gewährleistung und Haftung (1)	4
	Einsatz der GPL für eine kommerzielle Nutzung (2)	3,5	Verschiedene OS-Lizenzen, die man kennen muss (2)	2,5
	Einsatz der LGPL für eine kommerzielle Nutzung (2)	3,5	Vertragsverhältnis (1)	3
	Einsatz der BSD-Lizenz für eine kommerzielle Nutzung (1)	1		
	Urheberrechtsansprüche (1)	1		
	Inkompatibilität der Lizenzen (1)	4		
Sonstige Risiken	**Allgemein**	**2,75**	**Allgemein**	**2,5**
	Innovationsdruck (1)	4	Begrenzte Anwenderfreundlichkeit durch technik-getriebene Entwicklungen (1)	3

B.5 Vorteile eines Einsatzes von OSS aus Unternehmenssicht

Entsprechend der ermittelten Risiken eines Einsatzes von Open Source Software in Unternehmen gibt Tabelle 27 einen Überblick über die Vorteile, die Unternehmen mit einem solchen Einsatz verbinden. Die verwendete Bewertungsskala bleibt dabei identisch.

1 = sehr gering
2 = gering
3 = mittelmäßig
4 = hoch
5 = sehr hoch

Tabelle 25: Gegenüberstellung der Vorteile eines Einsatzes von OSS aus Unternehmenssicht

	Softwareunternehmen (Anz. der Nennungen)	***Bew.***	Anwendungsunternehmen (Anz. der Nennungen)	***Bew.***
Technische Vorteile	**Allgemein**	**3,9**	**Allgemein**	**3,8**
	Standardisierung (1)	4	Standardisierung (1)	5
	Offenheit (2)	4,5	Offenheit (4)	4
	Qualität der Software (3)	4,33	Anpassbarkeit (3)	4
	Hohe Testabdeckung (1)	4	Sicherheit (1)	5
	Anpassbarkeit (3)	4,66	Schnelles Bugfixing (2)	4,5
	Sicherheit durch • Nachvollziehbarkeit der Algorithmen (1) • Schnellere Behebung von Fehlern und Sicherheitslücken (1)	 5 4	Breites Angebot an Lösungen (1)	4
	Wiederverwendung (2)	4,5		
	Transparenz (2)	3,5		
	Aufbau auf vorhandenem Wissen (1)	5		
	Zugang zu Schnittstellen (1)	4		
	Bessere Wartung (1)	4		
	Große Entwicklergemeinde (2)	3,5		
Wirtschaftliche Vorteile	**Allgemein**	**3,8**	**Allgemein**	**4,1**
	Keine Lizenzgebühren (4)	3,75	Keine Lizenzgebühren (3)	4
	Abgabe der Pflege und Weiterentwicklung (1)	4	Geringere Kosten (1)	5
	Geringerer Eigenentwicklungsaufwand (1)	5	Bessere Planung von Releasezyklen (vermindert Abhängigkeit von Herstellervorgaben) (1)	3
	Hohe Entwicklungsgeschwindigkeit (2)	4,5		

	Ideenbefruchtung (1)	5	Investitionsschutz (1)	4
	Investitionsschutz (2)	4		
	Flexible Nutzung von Bibliotheken/Frameworks (1)	4		
	Kostenersparnis (4)	4,25		
	Keine Knebelverträge durch Lizenzen (1)	5		
Rechtliche Vorteile	**Allgemein**	**3,1**	**Allgemein**	**2,8**
	In Streitfällen spricht der Einsatz von OSS für ein hohes Qualitätsbewusstsein (1)	4	Eine Handvoll Open Source Lizenzen sind rechtlich einwandfrei und werden von der Open Software Foundation überwacht (1)	4
	Vermeidung weiterer Softwarepatente (1)	2	Lizenzbedingungen meist günstig (1)	2
	Keine eigene Haftung bei der Herstellung von OSS => Verschiebung Richtung Support (2)	4		
	Freie Nutzung des Quellcodes (1)	2		
	Flexible Nutzung von Bibliotheken/Frameworks (1)	4		
	Einsatz der GPL (1)	4		
Sonstige Vorteile	**Allgemein**	**3,4**	**Allgemein**	**2,75**
	Freude am Austausch mit Anderen (1)	4	Idealismus im Bereich der Softwareentwicklung (1)	3
	Dynamik der Entwicklungen (1)	4	Einfacheres Anwerben von qualifizierten Mitarbeitern, da OSS Produkte häufig zum Einsatz kommen (1)	4
	Bessere Fähigkeiten der Mitarbeiter (1)	4		
	Open Source Software ist oft sehr gut dokumentiert und für gängige Probleme findet sich schnell und kostenlos eine Lösung (Newsgroups, Mailinglisten, Webforen)			

Anhang C: GQM-Modell nach Cruz *et al.*

Nachfolgend ist die Zerlegung des Untersuchungsgegenstandes „Open Source Projekt“ in ein GQM-Modell nach Cruz *et al.* [Cruz2006] tabellarisch skizziert.

C.1 Anforderungen der Unternehmen

Tabelle 28 zeigt dazu zunächst die von Cruz *et al.* ermittelten Anforderungen von Unternehmen, die als mögliche Zielfaktoren in eine GQM-Zerlegung eingehen.

Tabelle 26: Anforderungen von Unternehmen an Open Source Software nach Cruz *et al.*

	Anforderungen der Unternehmen
Funktionale Anforderungen	Die wichtigen Funktionalitäten sind abgedeckt Eine klare Entwicklungsrichtung ist erkennbar
Technische Anforderungen	Die Zielplattform wird unterstützt Die Software ist zuverlässig Die Software ist leicht pflegbar
Organisatorische Anforderungen	Eine Projektcommunity existiert Die Software wird an sich ändernde externe Bedingungen angepasst Es ist ausreichend Support für die Software verfügbar Die Projektcommunity und ihre Software existieren langfristig Der Entwicklungsprozess im Projekt ist kompatibel mit dem Unternehmensprozess
Rechtliche Anforderungen	Kein Copyleft für Kombinationen aus OS Software und Unternehmenssoftware Keine Haftung für OS Software Keine Patentrechtsverletzungen
Wirtschaftliche Anforderungen	Die Weiterentwicklung der Software und entsprechender Support sind nachhaltig gesichert Eine Migration auf OS Software ist langfristig rentabel (Investitionsschutz) Der Einsatz von OS Software erhöht die Produktivität Die Software ist flexibel anpassbar an individuelle Bedürfnisse Die Software ist schnell verfügbar Die Nutzung von OS Software erhöht das Know-how der Unternehmensmitarbeiter Die Nutzung von OS Software reduziert die Unternehmenskosten Die Nutzung von OS Software ermöglicht eine Teilung der Entwicklungskosten
Politische Anforderungen	Die Weiterentwicklung der Software im OS Projekt ist beeinflussbar Der Einsatz von OS Software reduziert Herstellerabhängigkeiten Sicherheit der Software durch Offenlegung des Quellcodes kontrollierbar Der Einsatz von OS Software oder die Teilnahme in den Projekten bringt positive Marketingeffekte mit sich

C.2 Projektkriterien und abgeleitete Fragestellungen

Tabelle 29 gibt einen Überblick über die von Cruz *et al.* aufgestellten Kriterien von Open Source Projekten, die mit den zuvor beschriebenen Unternehmensanforderungen in Verbindung stehen. Zudem werden die Kriterien in spezifische Fragestellungen für eine Einordnung in ein GQM-Modell umgewandelt sowie die von Cruz *et al.* vorgeschlagenen Metriken genannt.

Tabelle 27: Projektkriterien und Fragestellungen nach Cruz *et al.*

	Projektkriterien	***Fragestellungen***
Funktionale Kriterien	Funktionsumfang	Welche Funktionalitäten realisiert die Software? [Metrik]: *Liste der Features auf der Webseite*
Technische Kriterien	Anzahl der Fehler im Programm	Wie viele Fehler enthält die Software? [Metrik]: *Anzahl der Fehler im Bug-Tracker*
	Anzahl der offenen Feature Requests	Wie viele Features der Software sind offen, d.h. noch nicht implementiert? [Metrik]: *Anzahl der Feature Request im Bug-Tracker*
	Code Metriken	Wie viele Dateien umfasst die Software? Wie viele Klassen umfasst die Software? Wie lang sind die Funktionen durchschnittlich? Wie viele Kommentare enthalten die Funktionen? [Metrik]: *Ergebnisse einer skriptbasierten Analyse des Quellcodes*
	Häufigkeit der Programmänderungen	Wie häufig/regelmäßig erfolgen Änderungen an der Software? [Metrik]: *Ergebnisse einer skriptbasierten Analyse der Log-Dateien der Quellcodeverzeichnisse*
	Abhängigkeiten von anderer Software	Von wie vielen anderen Softwarepaketen hängt das Funktionieren der Software ab? [Metrik]: *Angaben in der „ReadMe-Datei“ des Projektes*

Organisatorische Kriterien	Anzahl der Entwickler	Wie viele Entwickler umfasst das Projekt? [Metrik]: *Angaben auf der Webseite*
	Anzahl der Tester	Wie viele Tester umfasst das Projekt? [Metrik]: *Anzahl der Personen im Bug-Tracker, die Fehler melden und nicht zu den Entwicklern gehören*
	Anzahl der User	Wie viele User umfasst das Projekt? [Metrik]: *Anzahl der Downloads* *Anzahl der Teilnehmer in der Benutzermailingliste*
	Entwicklungsprozessmuster	Welchen Regeln folgt der Softwareentwicklungsprozess im Projekt? [Metrik]: *Angaben auf der Webseite*
	Fähigkeiten der Projektentwickler	Welche Fähigkeiten besitzen die Projektentwickler? [Metrik]: *Verfolgen der Diskussionen auf der Entwicklermailingliste*
Rechtliche Kriterien	Copyleft-Effekt der Lizenz	Enthält die OS-Lizenz ein Copyleft? [Metrik]: *Lizenzbestimmungen werden in der Regel gemeinsam mit der Software verbreitet*
	Haftung bei Weiterverkauf	Ist ein Unternehmen im Falle des Weiterverkaufs der Software für diese haftbar? Welches Ausmaß hat die Haftung? [Metrik]: *Beantwortung von drei Fragen:* • *Enthält die Lizenz einen kompletten Haftungsausschluss* • *Ist ein kompletter Haftungsausschluss im jeweiligen Land überhaupt möglich* • *Ist ein kompletter Haftungsausschluss für das evtl. im Hintergrund der Entwicklung stehende Unternehmen überhaupt möglich*

Wirtschaftliche Kriterien	Verfügbarkeit von Entwicklungsressourcen	In welchem Umfang investieren Unternehmen in Form von Infrastruktur, finanziellen Mitteln oder Entwicklungsressourcen in das Projekt? [Metrik]: *Angaben auf der Webseite*
	Migrationsaufwand	Welchen Aufwand bringt eine Migration auf die Software mit sich? [Metrik]: *Durchführung einer Kosten-Nutzen-Analyse*
	Monatliche Kosten	Mit welchen monatlichen Kosten ist zu rechnen? [Metrik]: *Durchführung einer Analyse zu den Laufzeit- und Pflegekosten der Software*
Politische Kriterien	Involvierung bekannter Unternehmen	Welche bekannten Unternehmen sind in das Projekt involviert? [Metrik]: *Angaben auf der Webseite*
	Abhängigkeiten von anderer OS-Software	Welche Abhängigkeiten existieren zu anderer OS-Software? [Metrik]: *Angaben auf der Webseite*
	Anzahl der Sprachen der Dokumentation	Existiert Dokumentation zur Software? In wie vielen Sprachen existiert die Dokumentation? [Metrik]: *Angaben auf der Webseite*
	Klima in den Diskussionsforen	Herrscht ein konstruktives, fortschrittsorientiertes Klima in den Foren? Dominieren hilfreiche Ratschläge oder destruktive Antworten die Diskussionen? [Metrik]: *Beobachtung der Diskussionen auf den Mailinglisten*
	Erwartete Änderungen im Umfeld des OS-Projektes	Stehen Unternehmen im Umfeld des Projektes kurz vor einem Verkauf, einer Fusion oder der Insolvenz? Stehen Unternehmen im Umfeld des Projektes vor einem Wechsel im Bereich des Managements? [Metrik]: *Angaben auf der Webseite*

Anhang D: Datenerhebung zum Thema „Zukunftssicherheit von Open Source Projekten“

Anhang D enthält zusätzliche Informationen zur durchgeführten Datenerhebung zum Thema „Zukunftssicherheit von Open Source Projekten. Dies beinhaltet den Fragenkatalog zur Umfrage an die Projektadministratoren sowie eine tabellarische Übersicht über die Ergebnisse der CVS/SVN-Analysen.

D.1 Fragenkatalog der Projektumfrage

Die folgenden Fragen wurden den Projektadministratoren der sechs untersuchten Open Source WCM-Systeme geschickt. Abgesehen von den zwei Antworten des Magnolia-Projektes beantwortete in den fünf anderen Fällen immer lediglich eine Administrator den zugesandten Fragenkatalog.

1. Does the …-Project have an income and which sources of income do you have?
   - [] sell of software support
   - [] dual licensing
   - [] advertising
   - [] merchandising
   - [] donations
   - [] the project has no income
   - [] others:
2. How many paid developers do you have in the …-Project?
   - [] 0
   - [] 1
   - [] 2
   - [] 3-5
   - [] 6-10
   - [] >10
3. How many companies do you know which provide support for your software?
4. How long do you think it takes for a new developer to get the knowledge about the software to become a core developer?
   - [] <1 year
   - [] 1-2 years
   - [] 3-5 years
   - [] 6-10 years
   - [] >10 years

5. How do you help new developers to get this knowledge?

   [] FAQ
   [] API specification
   [] process specification
   [] requirement description
   [] coding guide
   [] committer guide
   [] new member guide
   [] task list
   [] bugtracking system
   [] answering technical questions on the developer mailing list
   [] answering questions on the developer mailing list concerning the development direction
   [] answering questions on the developer mailing list concerning new contributions
   [] giving feedback about contributions of new developers
   [] tutorials
   [] others:

6. How much time do you spend per week providing support for new developers?

   [] 0 hours
   [] 1 hour
   [] 2 hours
   [] 3-5 hours
   [] 6-10 hours
   [] >10 hours

7. Are there any core values in the …-Project, which are shared by the community?

   7a. If yes, which?

   7b. If no, why not?

8. What is the main purpose of being for the …-Project?

9. Are there any long-term project goals in the …-Project?

   9a. If yes, which?

   9b. If no, why not?

10. Which of the following problems do you think could harm the future development of the …-Project?

    [] technical problems
    [] attracting new developers
    [] training of new developers
    [] skills of new developers
    [] project leadership
    [] only few core developers
    [] only few developers
    [] only few users
    [] different goals for the future development among the developers

[] project fork
[] decreasing developer motivation
[] financing of the development effort
[] the problem solved by the software is not up to date any more
[] others:

11. Which of the following statements do you think apply to the …-Project?
In 5 years...

[] ...the project will have a much larger community
[] ...the project will have a little larger community
[] ...the project community will have the same size
[] ...the project will have a little smaller community
[] ...the project will have a much smaller community
[] ...the project won't exist any more
[] I don't know what will be in five years

12. Which of the following statements do you think apply to the …-Project?
In 5 years...

[] ...the project software will be mature
[] ...the project software will be on a higher level
[] ...the project software will be on a similiar level
[] ...the project software will be on a lower level
[] ...the project software will be unmaintained
[] I don't know what will be in five years

13. What do you think are the key aspects that will lead to the success of the …-Project?

D.2 Ergebnisse der CVS/SVN-Analysen

Die folgende Tabelle ist eine Zusammenfassung der Resultate der CVS- bzw. SVN-Analysen von Drupal, Magnolia, Mambo, OpenCms, TGS CMS und Typo3.

Tabelle 28: Ergebnisse der CVS/SVN-Analysen von Drupal, Magnolia, Mambo, OpenCms, TGS CMS und Typo3

	Drupal	*Magnolia*	*Mambo*	*OpenCms*	*TGS CMS*	*Typo3*
Dev.	8	16	15	25	3	25
Dev.3	3	6	9	6	2	17
Dev.6	4	7	9	8	3	17
Dev.12	5	12	12	11	3	19
LOC	59.131	218.706	261.516	678.326	389.242	99.165
LOC3	1.611	1.226	18.652	7.341	6.767	54.307
LOC6	6.631	26.981	57.198	16.745	19.605	99.814
LOC12	18.907	131.672	196.432	22.542	50.853	99.814
Com.3	215	114	161	26	63	193
Com.6	715	403	238	126	126	279
Com.12	1.425	1.638	501	241	126	432
LOC3(ch)	7.841	4.695	84.050	9.331	58.674	11.131
LOC6(ch)	17.058	43.453	137.166	38.023	110.353	27.235
LOC12(ch)	42.691	220.334	313.192	50.612	110.353	89.494

Dev. = Anzahl der Entwickler im CVS oder SVN
Dev.x = Anzahl der in den letzten x Monaten aktiven Entwickler im CVS oder SVN
LOC = Gesamtanzahl der Quellcodezeilen der Software
LOCx = Anzahl der in den letzten x Monate hinzugefügten Quellcodezeilen
Com.x = Anzahl der in den letzten x Monaten durchgeführten Commits
LOCx(ch) = Anzahl der in den letzten x Monaten geänderten Quellcodezeilen

Glossar

Evaluation: Eine Evaluation bezeichnet die systematische Untersuchung des Wertes oder des Nutzens eines Gegenstandes. Solche Gegenstände können dabei z.B. Programme, Projekte, Produkte, Maßnahmen, Leistungen oder Technologien sein. Eine Evaluation sollte immer nachvollziehbar sein und auf empirisch gewonnenen Daten beruhen.

Goal-Question-Metric-Verfahren: Das Goal-Question-Metric-Verfahren ist ein iteratives Top-Down-Verfahren zur Analyse von z.B. Prozessen, Gegenständen oder Strukturen. Das Verfahren setzt sich dabei aus drei Analyseebenen zusammen, die aus speziellen Zielen zum Untersuchungsgegenstand, Fragestellungen zur Operationalisierung der Ziele sowie Metriken zur Messung der Fragestellungen bestehen.

Kernideologie: Das Konzept der Kernideologie in visionären Unternehmen nach Collins und Porras umfasst zwei Aspekte. Dies ist zum einen eine kleine Menge von Kernwerten, die zeitlich überdauernde Prinzipien eines Unternehmens repräsentiert. Zum anderen beinhaltet eine Kernideologie eine Daseinsberechtigung, die über finanzielle Interessen hinausgeht.

Open Source Projekt: Der Begriff „Open Source Projekt“ bezeichnet im Rahmen der Arbeit die Gesamtheit der Strukturen, Prozesse und beteiligten Personen im Zusammenhang mit der Entwicklung von Open Source Software sowie die Software selbst. Die Gruppe der an der Entwicklung aktiv oder passiv beteiligten Personen wird als „Community“ des Open Source Projektes bezeichnet.

Open Source Software: Open Source Software ist Software, deren Lizenz dem Nutzer drei grundlegende Freiheiten garantiert – die uneingeschränkte Nutzung, das Kopieren sowie die Weiterverbreitung. Letzteres kann in veränderter oder unveränderter Form erfolgen. Für eine Verbreitung der Software dürfen dabei keine Lizenzgebühren erhoben werden. Im Rahmen der Arbeit wird der Begriff „Open Source Software“ verwendet, um Software zu kennzeichnen, die unter einer der von der OSI oder FSF als „frei“ eingestuften Lizenz verbreitet wird.

Web Content Management System: Ein Web Content Management System ist ein rechnergestütztes System zur Verwaltung, Bearbeitung und Präsentation von Inhalten. Im Gegensatz zu Enterprise Content Management Systemen beschränken sich Web Content Management Systeme dabei im Wesentlichen auf webbasierte Formate wie HTML, XML oder GIF.

Zielfaktor: Mit dem Begriff „Zielfaktor“ werden im Rahmen der Arbeit Ziele innerhalb des Goal-Question-Metric-Verfahrens bezeichnet. Diese besitzen vier Dimensionen, die in dem Untersuchungsgegenstand selbst, dem betrachteten Aspekt, der Perspektive des Betrachters und der Absicht der Untersuchung bestehen.

Zukunftssicherheit: Zukunftssicherheit von Open Source Projekten bezeichnet im Rahmen der Arbeit die Wahrscheinlichkeit, mit der ein Open Source Projekt langfristig besteht und seine Software weiterentwickelt wird.

Literaturverzeichnis

[Balzert1998] Balzert, H.: Lehrbuch der Software-Technik: Software-Management. *Spektrum Akademischer Verlag GmbH,* **1998**

[Balzert2000] Balzert, H.: Lehrbuch der Software-Technik: Software-Entwicklung. *Spektrum Akademischer Verlag GmbH,* **2000**

[Basili1994] Basili, Victor R.; Caldiera, Gianluigi und Rombach, H. Dieter: The Goal Question Metric Approach. *http:// wwwagse.informatik.uni-kl.de/pubs/repository/basili94b/encyclo.gqm.pdf*, **1994**

[BCG2002] The Boston Consulting Group: Hacker survey. *http://www.osdl.com/bcg/*, **2002**

[Bitzer2004] Bitzer, J.; Schrettl, W. und Schröder, P.J.: Intrinsic Motivation in Open Source Software Development. *http://opensource.mit.edu/papers/bitzerschrettlschroder.pdf,* **2004**

[Bonaccorsi2005] Bonaccorsi, A. und Rossi, C.: Intrinsic motivations and profit-oriented firms in Open Source software. Do firms practise what they preach? *http://opensource.mit.edu/papers/rossi_motivations.pdf,* **2005**

[Brooks2000] Brooks, F.P., Jr.: The Mythical Man-Month: Essays on Software Engineering. *Addison-Wesley Publishing Company, 2nd edition*, **2000**

[Collins2002] Collins, J. und Porras, J. Built to Last: Successful Habbits of Visionary Companies. *HarperCollins Publishers,* **2002**

[Connell2000] Connell, C. Open Source Projects Manage Themselves? Dream On. *http://www.chc-3.com/pub/manage_themselves.htm,* **2000**

[Cox1998] Cox, A.: Cathedrals, Bazaars and the Town Council. *http://slashdot.org/article.pl?sid=98/10/13/1423253,* **1998**

[Clement2002] Clement-Fontaine, M.; Dalle, J. und Jullien, N.: New Economic Models, New Software Industry Economy. *French National Network for Software Technologies (RNTL),* **2002**

[Crowsten2005] Crowsten, K. und Howison, J.: The social structure of Free and Open Source software. *http://www.firstmonday.org/issues/issue10_2/crowston/*, **2005**

[Cruz2006] Cruz, D.; Wieland, T. und Ziegler, A.: Evaluation Criteria for Free/Open Source Software Products Based on Project Analysis. *Software Process Improvement and Practice*, **November 2006**, 107-122

[DeGEval2001] Deutsche Gesellschaft für Evaluation: Standards für Evaluation. *http://www.degeval.de/calimero/tools/proxy.php?id=227*, **2001**

[Ebert2004] Ebert, C. und Ruffin, M.; Sauerburger, H. *(ed.)*: Produkte entwickeln mit Open-Source-Software - Risiken und Erfahrungen. *dpunkt.verlag,* **2004**, 27-40

[Fugetta2003] Fuggetta, A.: Open source software – an evaluation. *The Journal of Systems and Software,* **2003***, 66*, 77–90

[Gartner2005] Gartner Inc.: MarketScope for Web Content Management 2005. *http://www.stellent.com/stellent3/groups/mkt/documents/nativepage/p88009826.pdf*

[Gartner2007] Gartner Inc.: MarketScope for Web Content Management 2007. *http://mediaproducts.gartner.com/reprints/vignette/article4/article4.html,* **2007**

[Ghosh2002] Ghosh, R.A.; Glott, R.; Krieger, B. und Robles, G.: Free/Libre and open source software survey and study, part 4: Survey of developers. *Technical Report Deliverable D18: Final Report, University of Maastricht,* **2002**

[Gorling2003] Gorling, S.: A critical approach to Open Source software. *http://opensource.mit.edu/papers/gorling.pdf,* **2003**

[Halloran2002] Halloran, T.J. und Scherlis, W.L.: High Quality and Open Source Software Practices. *2nd Workshop on Open Source Software Engineering.* ICSE, **2002**

[Hars2001] Hars, A. und Ou, S.: Working for Free? – Motivations of Participating in Open Source Projects. *Proceedings of the 34th Hawaii International Conference on System Sciences,* **2001**

[Healy2003] Healy, K. und Schussmann, A.: The Ecology of Open-Source Software Development. *http://opensource.mit.edu/papers/healyschussman.pdf,* **2003**

[Jaeger2002] Jaeger, T. und Metzger, A.: Open Source Software – Rechtliche Rahmenbedingungen der Freien Software. *Verlag C. H. Beck,* **2002**

[Jensen2007] Jensen, C. und Scacchi, W.: Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. *http://opensource.mit.edu/papers/Jensen-Scacchi-ICSE-2007.pdf,* **2007**

[Kalwey2003] Kalwey, N.; Kooths, S. und Langenfurth, M.: Open-Source Software: An Economic Assessment. *Muenster Institute for Computational Economics (MICE),* **2003**

[Krishnam2002] Krishnamurthy, S.: Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. *http://www.firstmonday.org/issues/issue7_6/krishnamurthy/index.html*, **2002**

[Lakhani2005] Lakhani, K.R. und Wolf, R.G.: Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. *Perspectives on Free and Open Source Software,* **2005**

[Lenz2007] Lenz, U.: IT-Systeme: Ausfallsicherheit im Kostenvergleich. *tecCHANNEL, http://www.tecchannel.de/index.cfm?webcode=458076,* **2007**

[Marquardt2004] Marquardt, C.; Sauerburger, H. *(ed.)*: Open-Source-Software. Open-Source-Software-Strategie der deutschen Verwaltung. *dpunkt.verlag,* **2004**, 58-61

[Michlmayr2004] Michlmayr, M. und Senyard, A.: How to Have a Successful Free Software Project. *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, **2004**

[Michlmayr2005a] Michlmayr, M.: Software Process Maturity and the Success of Free Software Projects. *http://www.cyrius.com/pubications/michlmayr-prcess_maturity_success.pdf,* **2005**

[Michlamayr2005b] Michlmayr, M.; Hunt, F. und Probert, D. Scotto, M. und Succi, G. *(ed.)*: Quality Practices and Problems in Free Software Projects *Proceedings of the First International Conference on Open Source Systems, Genova,* **2005**, 24-28

[Michlmayr2005c] Michlmayr, M.; Robles, G. und Gonzalez-Barahona, J.M. Scotto, M. und Succi, G. *(ed.)*: Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian. *Proceedings of the First International Conference on Open Source Systems, Genova,* **2005**, 100-107

[Mitre2001] Mitre Corporation: A Business Case Study of Open Source Software. *http://www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/kenwood_software.pdf*

[Mockus2002] Mockus, A.; Fielding, R.T. und Herbsleb, J.: Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology,* **2003**, 309-346

[O'Mahony2005] O'Mahony, S. und West, J.: Contrasting Community Building in Sponsored and Community Founded Open Source Projects. *Proceedings of the 38th Annual Hawai'i International Conference on System Sciences, Waikoloa, Hawaii,* **2005**

[OSI2007] Open Source Initiative: Open Source Case for Business. *http://www.opensource.org/advocacy/case_for_business.php,* **2007**

[QASystems2003] QA Systems: Java IDE Market Share Survey. *http://www.qa-systems.com/products/qstudioforjava/ide_marketshare.html,* **2003**

[Raymond2001] Raymond, E.S.: The Cathedral and the Bazaar. O'Reilly, T. *(ed.), O'Reilly & Associates*, **2001**

[Reiter2004] Reiter, B.E.; Sauerburger, H. *(ed.)* Wandel der IT: Mehr als 20 Jahre freie Software *dpunkt.verlag,* **2004** , 83-91

[Rothfuss2002] Rothfuss, G.J.: A Framework for Open Source Projects. *Department of Information Technology, University of Zurich,* **2002**

[Rullani2006] Rullani, F.: Dragging developers towards the core - How the Free/Libre/Open Source Software community enhances developers' contribution. *http://opensource.mit.edu/papers/rullani_comsocenv_v5.pdf,* **2006**

[Singh2006] Singh, P.V.; Youn, N. & Tan, Y.: Developer Learning Dynamics in Open Source Software Projects: A Hidden Markov Model Analysis. *http://opensource.mit.edu/papers/singh-youn-tan.pdf,* **2006**

[Stuermer2005] Stürmer, M.: Open Source Community Building. *Faculty of Economics and Social Science of the University of Bern*, **2005**

[Watson2005] Watson, A.: Reputation in Open Source Software. **2005**

[Wheeler2005] Wheeler, D.A.: Why Open Source Software? *http://www.dwheeler.com/oss_fs_why.html*, **2005**

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

_____ _____

Ort, Datum Unterschrift