Institut für Informatik
Arbeitsgruppe Software Engineering

Modell und Optimierungsansatz für Open Source-Softwareentwicklungsprozesse

Dissertation
zur Erlangung des akademischen Grades
"doctor rerum naturalium"
(Dr. rer. nat.)
in der Wissenschaftsdisziplin "Praktische Informatik"

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

von
Stefan Dietze

Potsdam, den 22.03.2004

Zusammenfassung

Gerade in den letzten Jahren erfuhr Open Source Software (OSS) eine zunehmende Verbreitung und Popularität und hat sich in verschiedenen Anwendungsdomänen etabliert. Die Prozesse, welche sich im Kontext der OSS-Entwicklung (auch: OSSD – Open Source Software-Development) evolutionär herausgebildet haben, weisen in den verschiedenen OSS-Entwicklungsprojekten z.T. ähnliche Eigenschaften und Strukturen auf und auch die involvierten Entitäten, wie z.B. Artefakte, Rollen oder Software-Werkzeuge sind weitgehend miteinander vergleichbar.

Dies motiviert den Gedanken, ein verallgemeinerbares Modell zu entwickeln, welches die generalisierbaren Entwicklungsprozesse im Kontext von OSS zu einem übertragbaren Modell abstrahiert. Auch in der Wissenschaftsdisziplin des Software Engineering (SE) wurde bereits erkannt, dass sich der OSSD-Ansatz in verschiedenen Aspekten erheblich von klassischen (proprietären) Modellen des SE unterscheidet und daher diese Methoden einer eigenen wissenschaftlichen Betrachtung bedürfen.

In verschiedenen Publikationen wurden zwar bereits einzelne Aspekte der OSS-Entwicklung analysiert und Theorien über die zugrundeliegenden Entwicklungsmethoden formuliert, aber es existiert noch keine umfassende Beschreibung der typischen Prozesse der OSSD-Methodik, die auf einer empirischen Untersuchung existierender OSS-Entwicklungsprojekte basiert. Da dies eine Voraussetzung für die weitere wissenschaftliche Auseinandersetzung mit OSSD-Prozessen darstellt, wird im Rahmen dieser Arbeit auf der Basis vergleichender Fallstudien ein deskriptives Modell der OSSD-Prozesse hergeleitet und mit Modellierungselementen der UML formalisiert beschrieben. Das Modell generalisiert die identifizierten Prozesse, Prozessentitäten und Software-Infrastrukturen der untersuchten OSSD-Projekte. Es basiert auf einem eigens entwickelten Metamodell, welches die zu analysierenden Entitäten identifiziert und die Modellierungssichten und -elemente beschreibt, die zur UML-basierten Beschreibung der Entwicklungsprozesse verwendet werden.

In einem weiteren Arbeitsschritt wird eine weiterführende Analyse des identifizierten Modells durchgeführt, um Implikationen, und Optimierungspotentiale aufzuzeigen. Diese umfassen beispielsweise die ungenügende Plan- und Terminierbarkeit von Prozessen oder die beobachtete Tendenz von OSSD-Akteuren, verschiedene Aktivitäten mit unterschiedlicher Intensität entsprechend der subjektiv wahrgenommenen Anreize auszuüben, was zur Vernachlässigung einiger Prozesse führt.

Anschließend werden Optimierungszielstellungen dargestellt, die diese Unzulänglichkeiten adressieren, und ein Optimierungsansatz zur Verbesserung des OSSD-Modells wird beschrieben. Dieser Ansatz umfasst die Erweiterung der identifizierten Rollen, die Einführung neuer oder die Erweiterung bereits identifizierter Prozesse und die Modifikation oder Erweiterung der Artefakte des generalisierten OSS-Entwicklungsmodells. Die vorgestellten Modellerweiterungen dienen vor allem einer gesteigerten Qualitätssicherung und der Kompensation von vernachlässigten Prozessen, um sowohl die entwickelte Software- als auch die Prozessqualität im OSSD-Kontext zu verbessern.

Desweiteren werden Softwarefunktionalitäten beschrieben, welche die identifizierte bestehende Software-Infrastruktur erweitern und eine gesamtheitlichere, softwaretechnische Unterstützung der OSSD-Prozesse ermöglichen sollen. Abschließend werden verschiedene Anwendungsszenarien der Methoden des OSS-Entwicklungsmodells, u.a. auch im kommerziellen SE, identifiziert und ein Implementierungsansatz basierend auf der OSS *GENESIS* vorgestellt, der zur Implementierung und Unterstützung des OSSD-Modells verwendet werden kann.

Abstract

In recent years Open Source Software (OSS) has become more widespread and its popularity has grown so that it is now established in various application domains. The processes which have emerged evolutionarily within the context of OSS development (OSSD – Open Source Software Development) display, to some extent, similar properties and structures in the various OSSD projects. The involved entities, e.g., artifacts, roles or software tools, are also widely comparable.

This leads to the idea of developing a generalizable model which abstracts the generalizable development processes within the context of OSS to a transferable model. Even the scientific discipline of Software Engineering (SE) has recognized that the OSSD approach is, in various aspects, considerably different from traditional (proprietary) models of SE, and that these methods therefore require their own scientific consideration.

Numerous publications have already analyzed individual aspects of OSSD and formulated theories about the fundamental development methods, but to date there is still no comprehensive description of the typical processes of OSSD methodology based on an empirical study of existing OSSD projects. Since this is a precondition for the further scientific examination of OSSD processes, a descriptive model of OSSD processes is obtained on the basis of comparative case studies and described in a formalized manner with UML modeling elements within the context of this dissertation. The model generalizes the identified processes, process entities and software infrastructures of the analyzed OSSD projects. It is based on a specially developed meta model which identifies the entities to be analyzed and describes the modeling viewpoints and elements which are used for the UML-based description of the development processes.

Another procedure step includes the further analysis of the identified model in order to display the implications, and the potential for optimization. For example, these encompass the insufficient planning and scheduling capability of processes or the observed tendency of OSSD actors to carry out various activities at different intensities depending on the subjectively perceived incentives which leads to some processes being neglected.

Subsequently, the optimization targets which address these inadequacies are displayed, and an optimization approach for the improvement of the OSSD model is described. The approach incorporates the expansion of the identified roles, the introduction of new or the expansion of already identified processes and the modification or expansion of artifacts of the generalized OSSD model. The presented model enhancements serve, above all, to increase quality assurance and to compensate neglected processes in order to improve developed software quality as well as process quality in the context of OSSD.

Furthermore, software functionalities are described which expand the existing identified software infrastructure and should enable an overall, software-technical support of OSSD processes. Finally, the various application scenarios of OSSD model methods - also in commercial SE - are identified and an implementation approach based on the OSS *GENESIS* is presented which can be used to implement and support the OSSD model.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASF	Apache Software Foundation
ASL	Apache Software License
BMBF	Bundesministerium für Bildung und Forschung
BSI	Bundesamt für Sicherheit in der Informationstechnologie
CASE	Computer Aided Software Engineering
CMM	Capability Maturity Model
CoXR	Code Cross Reference
CTR	Commit Then Review
CVS	Concurrent Versions System
DBMS	DataBase Management System
DMS	Document Management System
EDC	Electronic Data Capture
EGCS	Enhanced GNU Compilation System
FAQ	Frequently Asked Questions
FDA	Food and Drug Association
FSF	Free Software Foundation
FTP	File Transfer Protocol
GA	General Availability
GCC	GNU C Compiler
GCP	Good Clinical Practice
GDB	GNU Debugger
GENESIS	GENeralised Environment for ProceSs Management In Cooperative Software Engineering
GNU	GNU is Not Unix
GNU FDL	GNU Free Documentation License
GNU GPL	GNU General Public License
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Intellectual Property
IRC	Internet Relay Chat
IST	Information Society Technology Program
KBSt	Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnologie
KKS	Koordinierungszentrum für klinische Studien
LDP	Linux Documentation Project
LXR	Linux Cross Reference
MPAA	Motion Picture Association of America
MPL	Mozilla Public License
NCSA	National Center for Supercomputing Applications
OMG	Object Management Group
ODP	Open Dircetory Project
ODL	Open Directory License
OPHELIA	Open Platform and Methodologies for DeveLopment Tools Integration
OPL	Open Publication License

OS	Open Source
OSD	Open Source Definition
OSI	Open Source Initiative
OSS	Open Source Software
OSSD	Open Source Software Development
P2P	Peer-to-Peer
PDF	Portable Document Format
PGL	PhOSCo General License
PhOSCo	Pharma Open Source Community
RDBMS	Relational Database Management System
RCS	Revision Control System
RDE	Remote Data Entry
RFE	Request for Enhancement
RIAA	Recording Industry Association of America
RTC	Review Then Commit
RUP	Rational Unified Process
SE	Software Engineering
SPEM	Software Process Engineering Model
SPICE	Software Process Improvement and Capability Determination
SW	Software
TCL	Tool Command Language
TMF	Telematikplattform für Medizinische Forschungsnetze
UML	Unified Modelling Language
URL	Uniform Resource Locator
USDP	Unified Software Development Process
W3C	World Wide Web Consortium
WCMS	Web Content Management System
XML	Extensible Markup Language

1 Einleitung

„Es wäre sehr schön, wenn das Gebiet Software Engineering durch jede Dissertation um ein Epsilon schöner, größer oder sicherer würde. Dazu tragen Arbeiten bei, die die Landschaft, in der wir leben, erkunden und verändern, nicht solche, die neue Landschaften postulieren.“ [Ludewig02]

Die Open Source Bewegung hat im Laufe der Zeit verschiedene populäre Softwareprodukte hervorgebracht, wobei an dieser Stelle bloß die wahrscheinlich populärsten Vertreter, der Apache HTTPD-Server und der Linux Kernel erwähnt werden sollen. Nachdem in den frühen Phasen durch die rapide wachsende Open Source Community vor allem serverseitige oder infrastrukturelle Software hervorgebracht wurde, die eher „*more sophisticated users*“ [LeTi02] adressierte, konnten vor allem in den letzten Jahren die verschiedensten Anwendungsdomänen erschlossen werden, was dazu führte, dass inzwischen in den verschiedenen Anwendungsdomänen neben kommerziellen Produkten auch alternative, Open Source-basierte Software verfügbar ist.

Die zunehmende Bedeutung der Open Source Bewegung und die steigende Popularität und Professionalität der entwickelten Open Source Software steht aber noch einer vergleichsweise stagnierenden Erforschung, Beschreibung und Weiterentwicklung der in diesem Kontext praktizierten, Open Source-spezifischen Prozesse der Softwareentwicklung gegenüber. Daher wird im Rahmen dieser Arbeit versucht, auf empirischer Basis eine möglichst generalisierbare Beschreibung der in diesem Zusammenhang praktizierten Softwareentwicklungsmethoden zu erstellen und dadurch eine möglichst fundierte Analyse dieses deskriptiven Prozessmodells zu ermöglichen. Dies kann eine valide Basis darstellen, um eine wissenschaftliche Auseinandersetzung dieses vergleichsweise unerforschten Ansatzes zu ermöglichen und durch die identifizierten Methoden und Entitäten das gesamte Verständnis des Software Engineering (SE) zu erweitern [Cubranic01][1]. Dabei stellt der Open Source-typische, möglichst offene und unreglementierte Charakter der Entwicklungsprozesse spezifische Anforderungen, die es durch geeignete Prozesse und Infrastruktuern zu unterstützen gilt.

Den thematischen Kontext dieser Arbeit bilden somit die Methoden des SE im spezifischen Kontext der Open Source Software Entwicklung. Daher erläutern die folgenden Abschnitte dieses Kapitels neben der Zielsetzung und der Motivation, welche zur Durchführung dieses Promotionsvorhabens geführt haben, die Schwerpunktbegriffe dieser Arbeit aus der Modelltheorie, dem Software Engineering und in diesem Zusammenhang speziell dem Kontext quelloffener Software. Außerdem wird die instrumentalisierte Vorgehensweise erläutert.

1.1 Darstellung der Domäne Open Source

Dieser Abschnitt beschreibt die zentralen Charakteristika des Open Source Paradigmas, um ein Verständnis für die Zielstellung dieser Arbeit zu ermöglichen. Diesbezüglich werden allgemeingültige Merkmale von Open Source Software (OSS) und der praktizierten Entwicklungsprozesse dargestellt.

1.1.1 Open Source Software (OSS)

„The essence of Open Source software is that source code is 'free' -- that is -- open, public, non-proprietary.“ [Weber00]

[1] „[...] open-source software as a development methodology is not only here to stay, but has the potential to alter the whole approach to making software - resulting, its proponents would say, in more reliable products and faster and leaner development.“ [Cubranic01]

Der Begriff *Open Source Software* wurde von der Open Source Initiative (OSI) geschaffen, die damit eine Variante des vergleichsweise restriktiveren (vgl. [Perens99], [Stallman99], [Stallman01], [FSF03][2]) Free Software-Begriffes der Free Software Foundation definierte und eine möglichst weitreichende Akzeptanz und Verbreitung von Open Source Software sicherstellen sollte. Die Open Source Initiative (vgl. [OSI02]) hat zum Zweck der konkreten Begriffsdefinition und –abgrenzung die Open Source Definition [OSI02][3] entwickelt, die im folgenden vorgestellt wird und den Begriff der Open Source Software allgemein und auch für den weiteren Verlauf dieser Arbeit definiert.

Open Source Definition (OSD)
Die Open Source Definition (OSD) definiert verschiedene Kriterien, welche in den Lizenzbestimmungen der jeweiligen Software enthalten sein müssen, um als OSS entsprechend der OSD betrachtet werden zu können (vgl. [Jordan01]).

Die zentralen Inhalte der OSD sind zu den folgenden Kriterien zusammenfassbar sind (vgl. [OSI02][3]):

- Möglichkeit zur freien Verteilung
- Freier Zugang zum Quellcode
- Modifikationen und Derivate möglich
- Keine Einschränkung von Anwenderkreis oder Einsatzbereich

Eine Softwarelizenz entsprechend der OSD darf also niemanden in dem Recht auf die freie Weitergabe der Software und auf die Modifikation des Quellcodes beschränken, wobei der Quellcode als integraler Bestandteil der Software betrachtet werden muss. Zudem muß es die Lizenz ermöglichen, durch Modifikation des Quellcodes entstandene Derivate der Software unter den selben Lizenzbestimmungen weiterzuvertreiben.

Um eine konkrete Abgrenzung zu ermöglichen, können die verschiedenen Softwarelizenzen nach ihrer Kompatibilität zur OSD unterschieden werden, wobei eine Liste aller OSD-konformen Softwarelizenzen von der OSI zur Verfügung gestellt wird (vgl. [OSI02][4]). Im folgenden wird der Begriff Open Source Software ausschließlich für Software verwendet, die der OSD entspricht. Daher wird auf die Definition von weiteren Begriffen, die im Open Source Kontext von Bedeutung sind, wie z.B. Copyleft, Public Domain, Free Software, Freeware oder Shareware verzichtet.

1.1.2 OSS-Entwicklungsprozesse (OSSD[5])

Trotz der heterogenen Struktur der Entwicklungsprozesse und der vergleichsweise individuellen und unreglementierten Durchführung von Open Source Projekten, lassen sich verschiedene gemeinsame Charakteristika der evolutionär entstandenen Entwicklungsprozesse identifizieren, welche als Distinktionsmerkmale die OSS-Entwicklung vom traditionellen und proprietären SE abgrenzen (vgl. [FoBa02], [GaLaAr00], [Vixie99], [Koch00]). Zwar unterliegen die verschiedenen OSS-Entwicklungsprojekte keinen gemeinsamen Reglementierungen, aber es lassen sich verschiedene gemeinsame Eigenschaften der involvierten Entwicklungsprozesse identifizieren, die tendenziell auf alle OSS-Projekte anwendbar sind.

- Kollaborative Entwicklung durch dezentrale Akteure
- Heterogene Struktur der global verteilten Akteure
- Aktive Mitwirkung auf freiwilliger Basis
- Interaktion ausschließlich über internetbasierte Technologien (vgl. [FeFi02][6])

[2] URL: http://www.fsf.org/philosophy/free-doc.html; Abfrage: 22.03.2002
[3] URL: http://opensource.org/docs/definition.php; Abfrage: 22.03.2002
[4] URL: http://opensource.org/licenses/index.php; Abfrage: 22.03.2002
[5] OSSD: Open Source Software Development

- Individuelle, parallel ausgeführte Entwicklungsprozesse einzelner Akteure
- Unabhängiger *Peer Review*
- Dynamische Releaseprozesse
- Keine expliziten, formal hierarchischen Managementstrukturen

Eric S. Raymond assoziiert in [Raymond01] das traditionelle Software Engineering mit einer Kathedrale, die durch eine strikt zentralisierte Kontrolle und eine klare Managementstruktur gekennzeichnet ist. Im Gegensatz dazu entwickelt er das Modell eines Basars, der durch viele, global verteilte Anwender geprägt ist, die an stark parallelisierten Entwicklungsprozessen mitwirken und dabei auf eine zentrale Kontrolle weitgehend verzichten. Die parallelen Entwicklungsprozesse ermöglichen die simultane Bearbeitung verschiedener Bestandteile oder Aspekte der OSS [FeFi02][7]. Außerdem sind die verschiedenen Akteure im OSS-Entwicklungsprozess durch sehr heterogene Eigenschaften und Qualifikationen gekennzeichnet. Ein wichtiges Merkmal und zugleich auch signifikanter Vorteil des OSS-Entwicklungsmodells sind die unabhängigen Peer-Review-Prozesse, welche den Prozess der Software-Entwicklung im Open Source-Kontext prägen [FeFi02][8].

Die Koordination, respektive das Management der Prozesse hat nach [Nüttgens00], [Scacchi02] im OSS-Kontext lediglich korrigierenden und unterstützenden Charakter und weist nur marginale Anzeichen einer klassischen Hierarchiestruktur auf. Trotzdem existieren in allen Projekten zentrale Instanzen, welche in verschiedenen Ausprägungen die Entwicklungsprozesse ermöglichen, unterstützen und koordinieren [NoSt98][9] und diverse managementorientierte Aufgaben wahrnehemen.

Generell ist der OSS-Entwicklungsprozess durch nur sehr marginal stattfindende Planung und eine damit einhergende, vergleichsweise geringe Vorhersagbarkeit gekennzeichnet. Dies resultiert vor allem aus der nach [Nüttgens00] ausschließlich *„persönlich induzierten, volontären Zurverfügungstellung“* der Akteure, wodurch keinerlei Ressourcenplanung möglich ist. Entwickler, die in einem OSS-Projekt produktiv tätig werden, stellen ihr Wissen und ihre Fähigkeiten unentgeltlich zur Verfügung und lassen sich demzufolge von anderen Aspekten als der gewöhnlichen Leistungsvergütung motivieren, wie sie bei der Entwicklung proprietärer Software Anwendung findet. Die sozio-politische Motivation für die aktive Partizipation in OSS-Projekten umschreibt [FeFi00] wie folgt: *„Human motivations for OSS include scratching a developer's 'personal itch,' the desire for advancement through mentorship, peer reputation, the desire for 'meaningful' work, and community oriented idealism.“*

Eine der bedeutendsten Managementaufgaben ist die Veröffentlichung von Software-Releases. Dies ist durch sehr dynamische und häufige Release-Zyklen gekennzeichnet, durch die i.d.R. nur relativ wenig, zusätzliche Funktionalitäten oder Fehlerbehebungen zur Verfügung gestellt werden [FeFi02][10].

Die genannten Spezifika, welche als Distinktionsmerkmale die Entwicklung proprietärer Software von dem Prozess der Entwicklung von OSS abgrenzen, verdeutlichen die Unmöglichkeit der Anwendbarkeit klassischer Vorgehensmodelle und Methoden in diesem Kontext [Cubranic01][11].

[6] „The OSS process generally involves (or has the potential to involve) large, globally distributed communities of developers collaborating primarily through the Internet.“ [FeFi02]

[7] „These Developers tend to work in parallel, with different individuals/groups working on different aspects of the system simultaneously.“ [FeFi02]

[8] „OSS development communities often exploit the power of peer review to facilitate the debugging process, better articulate system requirements, and speed up the process of feature enhancement.“ [FeFi02]

[9] „Although all projects utilize some kind of central coordinating organization, the influence this organization on the contributions to the actual software differs.“ [NoSt98]

[10] „OSS projects are generally characterized by rapid, incremental release schedules, in which limited extra functionality is added in each release.“ [FeFi02]

[11] „[...] open-source software also faces some significant obstacles if it is to make a successful leap from the 'early adopters' of programming enthusiasts to the mainstream, obstacles that [...] cannot be surmounted by simply attempting to transplant ideas from the more traditional development models, even those that are also Internet-based.“ [Cubranic01]

1.2 Wissenschaftlicher Kontext des Vorhabens

In diesem Abschnitt werden das wissenschaftliche Umfeld und verwandte Arbeiten dargestellt, die maßgeblich für das weitere Vorhaben sind. Dabei werden primär offene Fragestellungen der Problemdomäne dieser Arbeit und für den weiteren Verlauf der Arbeit relevante Methoden und Begriffe erläutert, um ein Verständnis der darauf aufbauenden Zieldarstellungen und Vorgehensweisen zu ermöglichen.

1.2.1 Fragestellungen im Kontext von Open Source

„*What is needed [...] is research that will examine open-source software and development process, in light of their current accomplishments and respecting their specificities, and find what works, what doesn't, and how it can be further improved.“* [Cubranic01]

[Massey03] dokumentiert in diesem Zusammenhang die vorherrschende Diskrepanz zwischen den verschiedenen, domänenspezifischen Konnotationen des SE-Begriffes, da sich die Modelle der Wissenschaftsdisziplin des SE signifikant von den real existierenden Entwicklungsprozessen der (proprietären) Softwareentwicklungsindustrie, und speziell vom Verständnis der OSS-Bewegung unterscheiden. Nach [Massey03] betrifft dies sowohl die deskriptiven, als auch die präskriptiven Modelle (vgl. *1.2.2.1 Präskriptive, deskriptive und transiente Modelle*), die durch die SE-Community postuliert werden. Daher ist die Analyse real praktizierter Entwicklungsmethoden erforderlich, um ein Verständnis für die realen Anforderungen in der Softwareentwicklung zu schaffen und die Modelle des SE realen Erfordernissen annähern zu können.

Derzeit existieren noch viele Forschungslücken und offene Fragen bezüglich des OSS-Ansatzes der Software-Entwicklung, die ein gesamtheitliches, modellbasiertes Verständnis dieses Ansatzes noch nicht ermöglichen. Die verschiedenen wissenschaftlichen Arbeiten in diesem Kontext thematisieren i.d.R. lediglich verschiedene Teilaspekte, wie z.B. die ökonomische Motivation für OSS-Entwicklung (vgl. [Weber00], [LeTi02]) oder den Teilprozess der Anforderungsanalyse in diesem Zusammenhang (vgl. [Scacchi01a]), bieten aber keine umfassende Betrachtung des Entwicklungsprozesses und seiner spezifischen Merkmale in Form eines formalisierten Prozessmodells.

So sind z.T. in Anlehnung an [FeFi00] folgende Fragestellungen definierbar, welche in diesem Zusammenhang von Bedeutung sind:

- Welchem Prozessmodell unterliegt die Softwareentwicklung unter dem Paradigma von Open Source?
- Welche Erkenntnisse und Implikationen lassen sich aus dem dynamischen und kollaborativen Entwicklungsprozess ableiten?
- Wie entstehen Communities im Kontext von OS-Projekten und wie werden diese organisiert? Welche Implikationen lassen sich daraus für andere virtuelle Communities ableiten?
- Welche Merkmale unterscheiden die OSS-Entwicklung von proprietären, verteilten Softwareprojekten?
- Durch welche Software-Werkzeuge wird der OSS-Entwicklungsprozess unterstützt und wie unterscheiden sich diese von den im Kontext der traditionellen Software Entwicklung verwendeten?
- Welche Anforderungen muss eine optimale Projektinfrastruktur zur Durchführung von OSS-Projekten erfüllen? Kann eine derartige Infrastruktur auch partiell oder gesamtheitlich in der proprietären SW-Entwicklung eingesetzt werden?

Momentan existieren bereits verschiedene wissenschaftliche Arbeiten, welche jeweils verschiedene Aspekte der OSS-Entwicklung betrachten, z.B. [Raymond01], [Hertel02], [Weber00], [ArGaLa00], [Grassmuck02], [MoFiHe00], [BoHoBr99], [Cubranic01] und dabei z. T. auch ähnliche Zielsetzungen wie diese Arbeit verfolgen, wie z.B. [Scacchi01], [FeFi02] oder [Koch01]. Bisher besteht aber noch keine formalisierte und vollständige

Modellierung der Prozesse unter dem Paradigma von Open Source, die zudem auch auf einer fundierten und repräsentativen Analyse beruht [Scacchi01c][12].

Die verschiedenen Aspekte der Koordination der heterogenen und dezentralen Akteure in den virtuellen Communities der OSS-Projekte sind ebenso weitgehend unerforscht wie auch die Kriterien für eine erfolgreiche Durchführung eines Open Source Projekts, die als Basis für die Entwicklung eines Referenzmodells dienen könnten [Scacchi01c][13]. Zudem ist momentan eine Unterstützung der OSS-Entwicklung durch Software-Werkzeuge nur sehr begrenzt gegeben und es existieren keine, explizit den Anforderungen der OSS-Entwicklung angepassten Software-Infrastrukturen, welche diesen spezifischen Entwicklungsprozess gesamtheitlich abbilden und unterstützen [Cubranic01][14].

1.2.2 Modellbildung

Da die weitere Arbeit auf der strukturierten Entwicklung von Modellen nach wissenschaftlichen Methoden basiert, werden in diesem Abschnitt die zentrale Terminologie und die Methoden der Modellbildung erläutert.

1.2.2.1 Präskriptive, deskriptive und transiente Modelle

Modelle sind nach verschiedenen Kriterien klassifizierbar, wobei [Ludewig02] unter anderem zwischen deskriptiven und präskriptiven Modellen unterscheidet. Deskriptive Modelle, also Abbilder eines oder mehrerer real existierender Betrachtungsgegenstände, können dabei als Vorlage für die Erstellung eines präskriptiven Modells, also eines Vorbilds für einen realen Betrachtungsgegenstand verwendet werden. Dieser Spezialfall wird nach [Ludewig02] auch als transientes Modell bezeichnet.

Ein so erstelltes präskriptives Modell kann zur Definition eines Soll-Zustands, also als Vorgabe für eine konkrete Instanz des präskriptiven Modells instrumentalisiert werden.

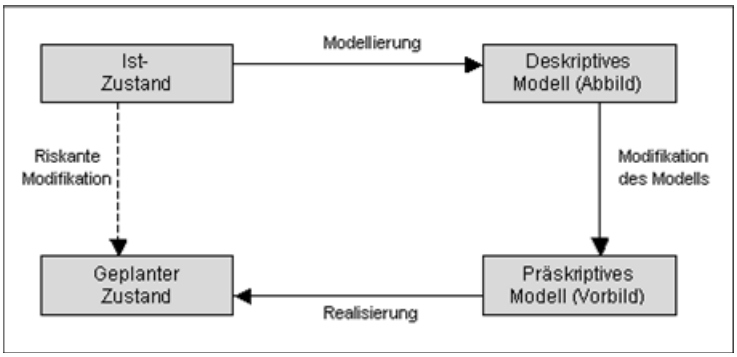

Abb. 1.1: Modellierungskreislauf nach [Ludewig02]

1.2.2.2 Metamodellierung

„In der Abstraktion, die mit der Schaffung von Modellen stets verbunden ist, liegt ihre Stärke: Das Modell gilt [...] für mehrere, unbegrenzt viele, für eine Klasse von Gegenständen.“ [Ludewig02]

[12] „While there is a growing popular literature attesting to open software [...], there are very few systematic studies - e.g., [FeFi00], [MoFiHe00], that informs how these communities produce software.” [Scacchi01c]

[13] „Similarly, little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success.“ [Scacchi01c]

[14] „What is needed are tools (and techniques, including social processes) that are flexible, complement the tools currently in use, and support the existing open-source development model almost transparently.” [Cubranic01]

Gemäß [OMG03][15] lassen sich in diesem Zusammenhang die folgenden Abstraktions- bzw. Metamodellebenen unterscheiden, deren Bedeutungsrahmen an die Erfordernisse der Arbeit geringfügig angepasst wurde:

Tab. 1.1: Metamodellebenen nach [OMG03][15]

Ebene	Beschreibung	Beispiel
Meta-Metamodell	Infrastruktur einer Metamodellierungs-architektur / Sprache zur Beschreibung von Metamodellen	Klassen, Attribute, Operationen
Metamodell	Instanz eines Meta-Metamodells / Sprache zur Spezifikation eines Modells	Rolle, Werkzeug, Artefakt
Modell	Instanz eines Metamodells / Sprache zur Beschreibung einer Informationsdomäne	Developer, Bug Tracking System, Patch
Instanz	Instanz eines Modells / Definiert eine bestimmte Informationsdomäne	Developer A, Bugzilla, Patch #2386

Der Grundgedanke der Metamodellierung beruht auf der Durchführung einer strukturierten Modellbildung, wobei ein Metamodell als präskriptives Modell des zu erstellenden Modells betrachtet werden kann bzw. ein Modell die Instanz eines Metamodells repräsentiert. Ein Metamodell beschreibt somit die Rahmenbedingungen zur Instanziierung eines Modells.

Ein Metametamodell definiert in diesem Kontext die Infrastruktur, also die zu verwendenden Elemente und deren Beziehungen für die Erstellung von Metamodellen bzw. definiert den Gestaltungsrahmen zur Definition eines Metamodells.

Ein Metamodell ist somit eine Instanz eines Meta-Metamodells und definiert auf einer niedrigeren Abstraktionsebene die Infrastruktur und die Sprache zur Erstellung eines Modells [OMG03][16]. Meta-Modelle definieren u.a. die verfügbaren Arten von Modellbausteinen und deren Beziehungen und somit den Gestaltungsrahmen für ein Modell. Metamodelle sind somit konkreter und elaborierter als Meta-Metamodelle. Diese Definition des Metamodellbegriffs ist äquivalent zur Definition des ebenfalls häufig verwendeten Begriff des Modelltyps in [HoSc93][17].

Ein Modell stellt eine Instanz eines Metamodells dar und beschreibt die Sprache zur konkreten Modellierung einer konkreten Ausprägungs- oder Problem Domäne. Ein Modell dient somit der Erstellung konkreter Modellinstanzen und beschreibt die entsprechenden Rahmenbedingungen.

1.2.2.3 Sichten

Modelle betrachten eine bestimmte Informationsdomäne unter verschiedenen Gesichtspunkten, die durch die Definition von Sichten voneinander abgrenzbar sind [HoSc93][18]. Ein Modell repräsentiert eine Gesamtheit von Einzelansichten. Sichten können als allgemeine Abstraktions- und Transparenzkonzepte verstanden werden und ermöglichen die Fokussierung auf einen bestimmten Aspekt des Betrachtungsgegenstands. Dabei muß stets die

[15] UML Spezifikation Version 1.4; URL: http://www.omg.org/cgi-bin/doc?formal/01-09-67; Abfrage: 22.10.2002

[16] „The primary responsibility of the metamodel layer is to define a language for specifying models." [OMG03][15]

[17] „Ein Modelltyp beschreibt eine Klasse von Modellen, sowie Bedingungen, die festlegen, wann ein Modell ein Exemplar des Typs ist." [HoSc93]

[18] „Zu jedem Zeitpunkt der Beobachtung eines Gegenstandsbereichs sind bestimmte Merkmale des Gegenstandsbereichs für den Zweck der Modellierung relevant, und es sind Teilbereiche abgrenzbar, die miteinander wechselwirken." [HoSc93]

übergreifende Konsistenz aller Sichten eines gemeinsamen Modells gewährleistet sein. Häufig werden die Begriffe Perspektive, Aspekt oder Viewpoint als Synonym für den Begriff der Sicht verwendet.

Im Kontext des Software Engineering können die konzeptuellen von den implementationsbezogenen Sichten abgegrenzt werden. Für die Prozessmodellierung sind dabei primär die konzeptuellen Sichten von Bedeutung, welche keinerlei implementationsbezogene Aspekte berücksichtigen (vgl. [Wortmann01], [Fowler00]).

1.2.2.4 Prozessmodelle im Kontext des Software Engineering

„Ein Prozessmodell ist eine Beschreibung einer koordinierten Vorgehensweise bei der Abwicklung eines Vorhabens. Es definiert sowohl den Input, der zur Abwicklung einer Aktivität notwendig ist, als auch den Output, der als Ergebnis der Aktivität produziert wird. Dabei wird eine feste Zuordnung von Workern vorgenommen, die die jeweilige Aktivität ausüben." [Versteegen00]

Diese konkrete Definition des Prozessmodellbegriffs beschreibt die grundlegenden Prinzipien eines klassischen Prozessmodells, die auf verbreitete Modelle des klassischen SE zutreffen und sich auch auf aktuellere Methoden wie den Unified Software Development Process (USDP, vgl. [JaBoRu99]) anwenden lassen. Die Evolution der Vorgehensmodelle führte vom klassischen Wasserfallmodell über iterative Ansätze, wie das Spiral- oder das komplexe V-Modell, zu objektorientierten Ansätzen, die bereits implementationsbezogene Aspekte in den Entwicklungsprozess integrieren und sich von der Funktionsorientierung abwenden [GaGr00][19]. Allen Modellen ist dabei die Definition von Projektphasen, Rollen, Aktivitäten und Artefakten, welche die In- bzw. Outputobjekte der einzelnen Phasen darstellen, gemein, wie auch in der Prozessdefinition von [JaBoRu99][20] zum Ausdruck kommt. Vorgehensmodelle repräsentieren im Kontext der Systementwicklung i.d.R. präskriptive Modelle.

Der Begriff des Prozessmodells wird im folgenden z.T. synonym für den Vorgehensmodellbegriff verwendet, obwohl das Vorgehensmodell im Software Engineering häufig als präskriptive Ausprägung eines Prozessmodells konnotiert ist. Der Nutzen der Verwendung von präskriptiven Vorgehensmodellen liegt in der gesteigerten Softwarequalität, höherer Planungssicherheit und effizienteren Abläufen, wie sich aus den Erfahrungen im traditionellen Software Engineering ableiten läßt. Zudem bildet die Etablierung und Realisierung präskriptiver Prozessmodelle die Grundlage für die Verwendung von Qualitätsmodellen wie SPICE oder dem Capability Maturity Model (CMM).

Phasen im Software Engineering
Der wesentliche Unterschied aller Prozessmodelle liegt in den jeweiligen zeitlichen Abhängigkeiten zwischen den einzelnen Phasen. Zur Vereinfachung werden die in [Vixie99], [Kruchten00] beschriebenen Phasen des SE im Rahmen der Arbeit zu den folgenden vier Phasen zusammengefasst, welche im folgenden als die Phasen der Systementwicklung verwendet werden:

- Requirements (Anforderungsanalyse)
- Design (Entwurf)
- Implementation
- Deployment (Einsatz und Anwendungstest)

1.3 Zielsetzung und Motivation

In diesem Abschnitt wird die konkrete Zielsetzung definiert, welche mit der Erstellung dieser Dissertation verfolgt wird und die zugrundeliegende Motivation erläutert.

[19] „Bekannte Vertreter dieser Klasse von Vorgehensmodellen sind z.B. der Semantic Object Modeling Approach [...] der Objectory Software Development Process bzw. Unified Software Development Process [...] und der OPEN Process [...]." [GaGr00]

[20] „A process defines who is doing what when and how to reach a certain goal." [JaBoRu99]

1.3.1 Zielstellung

„*The road awaiting for software engineering researchers is to gain better understanding of the dynamics and requirements of open-source development, which is necessary if the tools appropriate to open-source specificities are going to be developed."* [Cubranic01]

Um die real praktizierten Entwicklungsprozesse im OSS-Kontext adäquat unterstützen und optimieren zu können, ist es notwendig, eine fundierte Analyse, Identifikation und Beschreibung dieser Prozesse bereitzustellen [Scacchi98][21]. Dies erscheint umso mehr von Bedeutung, als dass die klassischen Modelle und Methoden des SE auf typische OSS-Projekte nicht anwendbar zu sein scheinen [Massey03][22].

Deskriptives Prozessmodell
Für die Entwicklung eines präskriptiven Referenzmodells für SE im Kontext von Open Source und die Implementierung unterstützender Software-Infrastrukturen kann bisher nicht auf eine fundierte und formalisierte Spezifikation eines beschreibenden Prozessmodells der OSS-Entwicklung zurückgegriffen werden, welche die relevanten Aspekte und Entitäten berücksichtigt. Mit der Arbeit an dieser Dissertation wird daher die primäre Zielsetzung verfolgt, ein deskriptives Prozessmodell der evolutionär herausgebildeten Systementwicklungsprozesse unter dem Paradigma von Open Source zu identifizieren und formalisiert zu beschreiben. Wie auch [Ludewig02] dokumentiert, sind präskriptive Modelle häufig nicht praxisrelevant, basieren z.T. auf fehlerhaften Annahmen und entbehren oft einer fundierten, deskriptiven Grundlage, welche durch dieses beschreibende Prozessmodell geschaffen werden soll.

Modellerweiterung (Präskriptives Modell) und Softwareunterstützung
Ein beschreibendes, möglichst fundiertes Prozessmodell ermöglicht es, die Strukturen, Prozesse und Entitäten des OSSD-Prozessmodells zu analysieren und inhärente Schwachstellen zu identifizieren. Anschließend können die verschiedenen Aspekte der identifizierten und evolutionär herausgebildeten Prozesse erweitert und ergänzt werden, um das deskriptive Prozessmodell dadurch zu verbessern und auch eine präskriptive Verwendung zu ermöglichen, was im Modellierungskreislauf in *1.2.2.1 Präskriptive, deskriptive und transiente Modelle* bereits dargestellt wurde. Diese optimierten Prozesse können somit als Referenzmodell zur Realisierung bzw. Instanziierung von Open Source Projekten verwendet werden [Scacchi01][23].

Da bisher keine gesamtheitliche Unterstützung aller Entwicklungsprozesse im OSS-Kontext durch eine adäquate Software-Infrastruktur existiert, bildet dieses Referenzmodell zudem die Grundlage, um die Anforderungen an eine möglichst integrierte Software-Infrastruktur zu identifizieren und implementationsunabhängig zu beschreiben. Eine derartige Infrastruktur sollte die spezifischen Anforderungen der OSS-Entwicklung unterstützen und die erforderlichen Funktionalitäten in einer gemeinsamen Plattform zur Verfügung stellen.

Ein derartig verbessertes und softwaregestütztes Prozessmodell kann die Basis für eine effizientere Realisierung von Softwareentwicklungsprojekten unter dem Paradigma von Open Source bilden und dadurch sowohl den Entwicklungsprozess, als auch die in dessen Rahmen entwickelte Software verbessern helfen. Obwohl die Formalisierung und Reglementierung des OSS-Entwicklungsansatzes der Philosophie dieses Ansatzes z.T. widerspricht, wurden die Notwendigkeit derartiger präskriptiver Modelle und die Potentiale zu deren Realisierbarkeit im OSSD-Kontext bereits dokumentiert (vgl. [CaLeCh02]). Zudem wurde bereits in verschiedenen Arbeiten (vgl. [ArGaLa00], [TrGoLeHo00], [Godfrey00], [GrTa99], [GoLe00], [BoHoBr99]) der

[21] „As such, if our goal is to (re-) design, formally model, simulate and enact new to-be software processes that use advanced technologies, then we need a systematic basis for understanding as-is software processes […]." [Scacchi98]

[22] „[...] the prescriptive and descriptive process models of software development apostolized by the SE community are widely viewed by the OSS community as inappropriate for their work." [Massey03]

[23] „Such a meta-model can be used to construct a predictive, testable, and incrementally refined theory of open software development processes within or across communities or projects." [Scacchi01]

Bedarf für die Gewährleistung einer gewissen Qualitätssicherung auch im Rahmen des OSSD-Modells identifiziert, indem die OSS-Qualität und –Architektur in Abhängigkeit von den zugrundliegenden OSS-Entwicklungsprozessen analysiert wurde.

1.3.2 Motivation und Nutzen

Diese Arbeit identifiziert und ein allgemeines Modell der Softwareentwicklung unter dem Paradigma von Open Source und beschreibt dieses basierend auf formalisierten und allgemeinverständlichen Darstellungsmethoden. Dadurch wird ein Verständnis der OSS-typischen, real praktizierten Softwareentwicklungsprozesse ermöglicht und ein Beitrag zur wissenschaftlichen Auseinandersetzung mit diesem spezifischen Modell der Softwareentwicklung geleistet, welches anhand real praktizierter Softwareentwicklungsprozesse hergeleitet und formalisert wurde. Dies stellt zudem die Basis für die weitere, fundierte Analyse dieser Prozesse dar und unterstützt die Erstellung eines präskriptiven Referenzmodells für die Entwicklung von OSS. Durch die fundierte Analyse und gesamtheitliche Modellierung der relevanten Aspekte der Software-Entwicklung im Kontext von OSS kann eine adäquate Optimierung der Prozesse durchgeführt und eine geeignete, möglichst gesamtheitliche softwaretechnische Unterstützung der Entwicklungsprozesse entwickelt werden. Die Prozessoptimierung und -automatisierung durch eine verbesserte Software-Infrastruktur kann zudem einen Beitrag zur Steigerung sowohl der Prozess- als auch der Softwarequalität leisten.

Die Ergebnisse dieser Arbeit können somit als Basis für die Etablierung neuer Softwareprojekte verwendet werden, welche zumindest teilweise nach Prinzipien der OSS-Entwicklung durchgeführt werden sollen. Dadurch könnte der OSSD-typische, langwierige Evolutionsprozess der Infrastrukturen und Prozesse eines Projekts und der Prozess der schrittweisen Annäherung an *optimale* Prozesse und Infrastrukturen erheblich verkürzt werden.

Da i.d.R. auch proprietäre Softwareentwicklungsprojekte in verteilten, dezentral organisierten sozialen Systemen durchgeführt werden, kann das OSS-Entwicklungsmodell auch als spezielles Paradigma der verteilten Software-Entwicklung betrachtet werden. Daher können die Erkenntnisse aus dem Bereich der OSS-Entwicklung u.U. auch Einfluss auf traditionelle (proprietäre) Softwareentwicklungsmethoden und das gesamte Verständnis der Software-Entwicklung ausüben [Cubranic01][24]. Somit kann die fundierte Betrachtung der in der allgemeinen Systementwicklung relevanten Prozesse und Prozess-Entitäten in dem speziellen Kontext des OSSD dazu verwendet werden, die Potentiale zur Berücksichtigung bestimmter Elemente des OSSD auch in der kommerziellen Softwareentwicklung zu evaluieren (vgl. [NoSt98]) bzw. in die durch die Wissenschaftsdisziplin des Software Engineering postulierten Modelle zu übernehmen.

1.3.3 Einschränkungen der Zielstellung

„Systematisierung und Formalisierung finden ihre Grenze an den nichtformalisierbaren Denk- und Assoziationsvorgängen während der kreativen Tätigkeit der Menschen und an den nichtsystematisierbaren und nichtformalisierbaren, im wesentlichen durch psychologische und soziologische Kategorien bestimmten Interaktionen zwischen den an der Softwareentwicklung beteiligten Menschen." [Weber92]

Die spezifischen Merkmale der OSS-Entwicklung provozieren Zweifel an der uneingeschränkten Realisierbarkeit eines präskriptiven Prozessmodells und einer gesamtheitlichen, allgemein einsetzbaren Projektinfrastruktur.

Aufgrund der Abwesenheit einer hierarchisch übergeordneten Instanz mit den Aufgaben der Planung, Überwachung und Steuerung des Entwicklungsprozesses im Sinne eines effizienten Projektmanagements, ist die

[24] „[...] open-source software as a development methodology is not only here to stay, but has the potential to alter the whole approach to making software - resulting, its proponents would say, in more reliable products and faster and leaner development." [Cubranic01]

Etablierung eines allgemeinen Vorgehensmodells nur in begrenztem Maße möglich. [Weber92][25] dokumentiert, dass im gesamten Bereich der Software-Entwicklung präskriptive Prozess- bzw. Vorgehensmodelle nur eingeschränkt und nie erschöpfend etablierbar sind. Zudem steht die Entwicklung eines derartigen Modells im Kontext des OSSD unter anderen Vorzeichen als die vergleichbarer Rahmenwerke des klassischen SE, da tendenziell sehr heterogene und kaum planbare Strukturen, sowohl bezüglich der Aufbau- als auch der Ablauforganisation, unterstützt werden müssen. Anspruch dieses präskriptiven Prozessmodells kann es somit nur sein, eine weitgehend selbstregulierende und autarke Arbeitsweise zu ermöglichen und nötige Vorgaben für die bestmögliche Unterstützung und Automatisierung der verteilten Entwicklungsprozesse zu definieren und damit eine Standardisierung dieses bisher sehr individuell umgesetzten Ansatzes zu ermöglichen, soweit dies sinnvoll und mit den Implikationen der OSS-Entwicklung vereinbar ist.

Daher können die Modifikationen, welche zur Optimierung der Prozesse durchgeführt werden, nur in engen Grenzen stattfinden, um damit die freie Evolution sowohl des Entwicklungsprozesses als auch der OSS nicht einzuschränken und mit den typischen Charakteristika des OSSD nicht zu konfligieren. Dabei sollten die identifizierten Entwicklungsprozesse also lediglich unterstützt und nicht determinierend reglementiert werden, indem unterstützende Methoden, koordinierende Prozesse, qualitätssichernde Elemente oder relevante Softwarefunktionalitäten definiert werden, die durch eine gesamtheitlichere Software-Infrastruktur im Rahmen der OSS-Entwicklung unterstützt werden können [Scacchi01][26].

Eine allgemein einsetzbare Software-Infrastruktur scheint zudem nur bedingt realisierbar, da OSSD-Prozesse in verschiedenen Projekten z.T. individuell verschieden praktiziert werden. Daher verfolgt die Analyse dieser Arbeit lediglich die Zielstellung, eine möglichst flexible und an die verschiedenen Anforderungen adaptierbare Software-Infrastruktur zu konzeptionieren, welche individuell konfiguriert werden kann und in die verschiedenen Szenarien von Open Source Projekten integrierbar ist.

1.4 Methodik

„[…] if our goal is to develop and refine testable and predictive theories of software development, use, or evolution, then we need to empirically examine, describe and analyze how software processes occur, or will occur, in different organizational settings. These are the goals of comparative case analysis for understanding software processes […]" [Scacchi98]

Zur Erreichung der ersten Zielstellung der Arbeit, respektive der Erstellung eines deskriptiven Prozessmodells der Systementwicklung im Open Source Kontext, ist die Durchführung von vergleichenden Fallstudien eine geeignete Methode [Scacchi98][27]. Projektbezogene, empirische Studien können zur Identifikation von generalisierbaren Strukturen und Prozessen verwendet werden, die anschließend die Basis der formalisierten Beschreibung des verallgemeinerten, deskriptiven Prozessmodells darstellen. Vergleichende Fallstudien repräsentieren nach [Scacchi98][28] ein geeignetes Instrument, um generalisierbare und beweisbare Aussagen über real existierende Strukturen zu treffen.

[25] „Man sollte nicht erwarten, dass mehr als das Formalisieren von Fragmenten des Software-Entwicklungsprozesses möglich wäre, und sollte Softwareprozessmodellen, die vorgeben, den gesamten Prozess der Softwareentwicklung zu modellieren zu gestatten, mit äußerster Skepsis begegnen." [Weber92]

[26] „A process meta-model can also be used to configure, generate, or instantiate Web-based process modeling, prototyping, and enactment environments that enable modeled processes to be globally deployed and computationally supported." [Scacchi01]

[27] „[…] comparative analysis helps illuminate the behavioral and interactional dynamics of as-is software processes occurring in complex organizational settings." [Scacchi98]

[28] „The overall purpose of comparative case analysis is to discover and highlight second- or higher-order phenomena or patterns that transcend the analysis of an individual case. Comparative case analysis provides a strategy that enables the development of more generalizable results and testable theories than individual or disjoint case studies alone can provide." [Scacchi98]

In diesem Zusammenhang wurde daher eine konkrete Vorgehensweise definiert, welche in den folgenden Abschnitten noch explizit erläutert wird und sich z.T. an der in [Scacchi98] definierten Vorgehensweise zur Identifikation von Prozessmodellen im Systementwicklungskontext anlehnt:

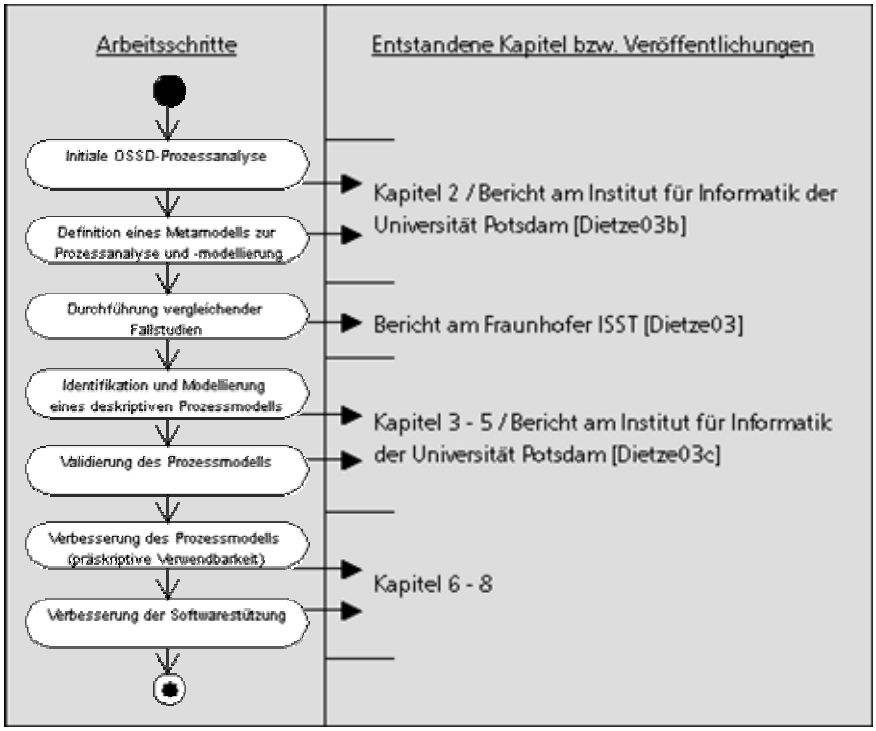

Abb. 1.2: Verwendete Vorgehensweise

Abb. 1.2 visualisiert sowohl die angewendete Vorgehensweise und die in den einzelnen Arbeitsschritten entstandenen Kapitel, als auch die bei der Erstellung dieser Arbeit entstandenen, begleitenden Veröffentlichungen, welche einige Ergebnisse der Arbeit detaillierter und erschöpfender darstellen, als es im Rahmen dieser Dissertation möglich ist. Die folgenden Abschnitte beschreiben die wichtigsten Arbeitsschritte konkret.

1.4.1 Initiale OSS-Prozessanalyse

Die Erstellung eines OSS-spezifischen Metamodells erfordert ein erstes Studium der verschiedenen OSS- und OSSD-Charakteristika anhand möglichst repräsentativer Projekte bzw. bereits existierender wissenschaftlicher Arbeiten in diesem thematischen Umfeld, um die Erstellung eines probaten Metamodells zu ermöglichen, welches die verschiedenen Distinktionsmerkmale zum proprietären Software Engineering auf der Metaebene abstrahiert.

1.4.2 Definition eines Metamodells

Um eine möglichst strukturierte Analyse einzelner Projekte im Rahmen einer vergleichenden Fallstudie zu ermöglichen, wird ein Metamodell definiert, welches die zu analysierenden Aspekte definiert und bereits auf einer Metaebene strukturiert. Die formalisierte Beschreibung eines einheitlichen Metamodells ermöglicht die strukturierte und vergleichbare Identifikation und Beschreibung der in den einzelnen Fallstudien identifizierten Aspekte, was eine wichtige Voraussetzung für die Generalisierung verallgemeinerbarer Aspekte darstellt (vgl.

[Scacchi98]). Dieses Metamodell wird analog zum Metamodell der UML [OMG03][29] basierend auf Klassendiagrammen modelliert und beschreibt die zu analysierenden Aspekte bzw. weist diesen bereits verschiedene spezialisierte Metaklassen zur Strukturierung zu. Außerdem werden in diesem Metamodell bereits verschiedene Sichten und Modellierungselemente für die formalisierte Darstellung des generalisierten Prozessmodells spezifiziert, um dadurch den Gestaltungsrahmen für die formalisierte Beschreibung der identifizierten Prozesse und Prozessentitäten bereitzustellen.

1.4.3 Durchführung vergleichender Fallstudien

„Case studies are well suited to capture and describe how software processes occur in real-world settings, what kinds of problems emerge, how they are addressed, and how software engineering tools, techniques, or concepts are employed." [Scacchi98]

Projektauswahl
Die Basis für die Modellierung des Prozessmodells im Open Source Kontext bilden vergleichende Fallstudien, die anhand konkreter OSS-Entwicklungsprojekte durchgeführt werden und die Basis zur Identifikation von generalisierbaren Aussagen über real praktizierte Softwareentwicklungsprozesse bilden [Scacchi98][30]. Dies setzt die Auswahl von mehreren Projekten voraus, welche als möglichst repräsentativ für die gesamte Klasse der OSS-Projekte betrachtet werden. Hierzu wurden die folgenden Auswahlkriterien definiert:

- Lizenzmodell kompatibel zur OSD
- Fortgeschrittener Projektstatus
- Möglichst fortgeschrittene Prozessevolution
- Weitreichende Akzeptanz der entwickelten OSS
- OSS aus verschiedenen Anwendungsdomänen

Die Wahl der Softwarelizenz determiniert in hohem Maße die Rahmenbedingungen für die Evolution der Entwicklungsprozesse. Nur die Verwendung eines Lizenzmodells, welches mit der OSD konform ist, gewährleistet, dass die entwickelte Software allen Kriterien für OSS entspricht und die darauf basierenden Entwicklungsprozesse den OSS-typischen Prämissen unterworfen sind.

Ein möglichst fortgeschrittener Projektstatus stellt sicher, dass das Projekt bereits im Rahmen einer möglichst freien Evolution die typischen Prozesse und Strukturen eines OSS-Projekts herausgebildet hat. Diese sollten möglichst weitreichend dokumentiert sein, um die Recherche und Beschreibung der Prozesse und aller involvierten Aspekte zu ermöglichen. Weiterhin sollten in der ausgewählten Projektgruppe möglichst verschiedene Softwaretypen aus unterschiedlichen Anwendungsdomänen repräsentiert sein. Aufgrund dieser Kriterien wurden Entwicklungsprojekte ausgewählt, welche die Entwicklung der folgenden OSS zum Ziel haben:

- Linux Kernel[31]
- Apache HTTPD[32] und Apache Jakarta[33]
- Mozilla[34]

[29] OMG Unified Modeling Language Specification - Version 1.4; URL: http://www.omg.org/cgi-bin/apps/doc?formal/01-09-67.pdf; Abfrage: 23.07.2002

[30] „Qualitative field studies are well suited for capturing and analyzing the structure, flow, and other behavioral dynamics that characterize how a software process is performed. […] Thus, comparative case analysis of as-is software processes can lead to a distinct class of results and insights compared to traditional individual case studies or quantitative studies." [Scacchi98]

[31] URL: http://www.kernel.org

[32] URL: http://httpd.apache.org

[33] URL: http://jakarta.apache.org

[34] URL: http://www.mozilla.org

Einzelne Apache Jakarta Teilprojekte und auch das Linux Kernel Projekt repräsentieren vergleichsweise kleine Entwicklungsprojekte, wobei die Linux Kernel Entwicklung strikt von der Entwicklung sonstiger Linux- und Betriebssystem-Software abgegrenzt werden muss, die nicht in den Fokus dieser Arbeit reicht. Mit dem Apache HTTPD- und dem Mozilla-Projekt sind auch Projekte involviert, welche über eine sehr große Community verfügen und daher als Repräsentanten für etablierte und evolutionär weit entwickelte OSS-Projekte betrachtet werden können. Zudem sollten auch Projekte in die Analyse einbezogen werden, welche ursprünglich in einem kommerziellen Kontext initiiert wurden, da dies eine häufig auftretende Evolution von OSS-Projekten darstellt (vgl. [Koch00], [Pavlicek00], [Rosenberg00], [Sandred01]). Dieser Anforderung wurde durch die Auswahl des Mozilla-Projekts Rechnung getragen.

Projektanalyse
Die Analyse dieser Projekte basiert auf einer Symbiose aus dem empirischen Studium der Projekte und der Auswertung von bereits existierenden Fallstudien, welche spezifische Teilaspekte des Projekts betrachten.

Das empirische Studium umfasst dabei die Recherche in den projektinternen Informationsquellen und Ressourcen und auch die aktive Partizipation am entsprechenden Projekt. Wie auch [Koch01] bemerkt, ermöglichen die Offenheit der Open Source Communities und der freie Zugang zu jeder Entwicklergemeinde die Partizipation und das umfassende Studium von OSS-Projekten. Alle Koordinations- und Kooperationswerkzeuge und –mechanismen sind frei zugänglich und auch die zentralen Ressourcen sind für jeden interessierten Akteur nutzbar und einsichtig. Diese empirisch-analytischen Aktivitäten werden durch die Auswertung von sekundärer Literatur, verwandten, wissenschaftlichen Arbeiten und existierenden Fallstudien ergänzt.

Dadurch entstehen primär qualitative aber z.T. auch quantitative Informationen über die in den Projekten praktizierten Prozesse, welche zur Konstruktion einer fundierten Projektbeschreibung beitragen, die sich an den Vorgaben des Metamodells orientiert und alle im Metamodell definierten Aspekte strukturiert darstellt. Den Schwerpunkt dieser Prozessdarstellung bilden somit qualitative Informationen über die praktizierten Entwicklungsprozesse, die zur Ableitung von generalisierbaren Prozessen verwendet werden.

Die Struktur und das Format dieser Dokumente wird dabei durch die verschiedenen Klassen des Metamodells geprägt, welche den gesamten Entwicklungsprozess bereits in verschiedene Teilprozesse und zu betrachtende Aspekte partitionieren. Die so entstandenen Projektanalysen und Prozessdokumentationen in [Dietze03] dienen als Ausgangsbasis für die Generalisierung und Modellierung eines allgemeinen Prozessmodells.

1.4.4 Identifikation und Modellierung eines deskriptiven Prozessmodells

„Comparative case studies are […] important in that they can serve as foundation for the formalization of our findings and process models as a process meta-model." [Scacchi01]

Generalisierung bzw. Abstraktion
Basierend auf den erstellten Projektspezifikationen werden im Rahmen dieses Arbeitsschrittes die über alle Projekte hinweg generalisierbaren Prozesse und Prozessentitäten abstrahiert. Sofern ein Aspekt (z.B. Teilprozess, Rolle, Artefakt) nicht in allen Projekten explizit identifiziert wurde, aber dessen implizite Existenz aufgrund einer sinnvollen Annahme vermutet werden muss, wird dies als Prämisse unterstellt und der so generalisierbare Aspekt trotzdem in die Modellierung einbezogen. Die Notwendigkeit zur Definition geeigneter Prämissen resultiert aus den Einschränkungen und Unzulänglichkeiten, die sich aus der fallstudienbasierten Vorgehensweise ergeben, da Fallstudien nie eine erschöpfende sondern lediglich eine eingeschränkte Analyse des jeweiligen Betrachtungsgegenstandes zulassen, welche zudem auch mit verschiedenen Unsicherheiten behaftet ist.

Modellierung
Anschließend können diese Prozesse und die involvierten Entitäten auf einer möglichst granularen aber noch verallgemeinerbaren Abstraktionsebene in den verschiedenen Sichten des Metamodells und basierend auf dessen Vorgaben modelliert werden.

1.4.5 Validierung des Prozessmodells

Zur Verifikation der Korrektheit und Praxisrelevanz wird das generalisierte Prozessmodell (Modellebene) anschließend anhand einer konkreten Ausprägung dieses Modells (Instanzenebene), also anhand eines spezifischen OSS-Entwicklungsprojekts validiert und die verschiedenen Ausprägungen der Prozesse und aller involvierter Aspekte in diesem Kontext betrachtet. Dies kann zur Veri- oder Falsifizierung des erstellten Prozessmodells führen.

1.4.6 Verbesserung des Prozessmodells

Die Verbesserung der Prozesse bzw. die Definition von ergänzenden Prozessen, welche die kollaborativen Entwicklungsaktivitäten unterstützen, soll die Entwicklung eines präskriptiven OSSD-Modells ermöglichen und stellt eine wichtige Bedingung dar, um eine verbesserte Softwareunterstützung zu entwickeln.

Identifikation von Optimierungsansätzen des Prozessmodells
Das formalisierte und beschreibende Prozessmodell kann anschließend als Basis für die Herleitung von verallgemeinerbaren Implikationen, Erkenntnissen oder Schwächen des OSS-Modells verwendet werden, die anhand der generalisierten Strukturen und Prozesse identifiziert werden können. Dies bildet die Basis zur Definition von Zielstellungen, die mit der Verbesserung der evolutionär herausgebildeten OSSD-Prozesse verfolgt werden sollten. Außerdem können diese Erkenntnisse der Identifikation von Anforderungen an eine unterstützende Software-Infrastruktur dienen.

Prozessmodellverbesserung
In einem weiteren Arbeitsschritt werden die formalisiert spezifizierten Prozesse des deskriptiven Prozessmodells um verschiedene unterstützende Elemente erweitert, welche etwaige identifizierte Schwachstellen kompensieren. Dabei werden die identifizierten Entwicklungsprozesse, die sich in einem evolutionären Prozess herausgebildet haben, möglichst nicht reglementiert, sondern nur durch weitere Elemente unterstützt. Dies könnte z.B. die explizite Definition von lediglich implizit identifizierten Rollen oder Prozessen, die Integration qualitätssichernder Elemente, die Definition von dedizierten Artefakten oder Rollen oder die Modellierung unterstützender Prozesse umfassen.

1.4.7 Verbesserung der Softwarestützung

Das derart erweiterte Prozessmodell dient als Basis für die fundierte Herleitung und implementationsunabhängige Beschreibung von Softwarefunktionalitäten, welche eine adäquatere Unterstützung bzw. Automatisierung des kollaborativen Prozessmodells ermöglichen. Diese hergeleiteten Funktionalitäten können anschließend durch die softwaretechnische Realisierung abgebildet werden, wobei möglichst die Verwendung einer bereits existierenden Software zur Implementation der OSSD-Prozesse und Prozessentitäten angestrebt wird.

1.4.8 Implikationen und Einschränkungen der Vorgehensweise

In diesem Abschnitt werden die gewählte Methode diskutiert und Einschränkungen thematisiert, die sich aus dem gewählten Ansatz ergeben.

Die Granularität und die Detaillierungstiefe, die bei generalisierten Prozessdarstellungen basierend auf vergleichenden Fallstudien erreicht werden, kann nie einen bestimmten Detaillierungsgrad überschreiten und wird sich daher immer auf einem vergleichsweise hohen Abstraktionsniveau bewegen [Scacchi98][35]. Daher sind sehr detaillierte Prozessmodelle in diesem Kontext wahrscheinlich nur eingeschränkt möglich. Die so erstellten

[35] „[...] processes whose granularity scales down to individual behavior or up to large team work structure have not been subjected to comparative case analysis.“ [Scacchi98]

Prozessspezifikationen können nach [Scacchi98][36] primär als Managementwerkzeug oder als Ausgangspunkt für detailliertere Forschungsaktivitäten dienen und können nicht immer den Granularitätsgrad erreichen, der zur Erstellung von unterstützenden Software-Werkzeugen notwendig ist.

Anhand der ausschließlichen Durchführung von vergleichenden Fallstudien kann ein Softwareentwicklungsprozess daher nicht erschöpfend und in all seinen Facetten analysiert und modelliert werden, da dabei lediglich Daten von begrenzter Qualität und Quantität und auf einer limitierten Detaillierungstiefe erhoben werden. Daher stellen sie lediglich eine Ergänzung zu weiteren wissenschaftlichen Methoden dar [Scacchi98][37]. Studien eines expliziten Systementwicklungsprozesses betrachten stets nur bestimmte Aspekte des Prozesses und haben immer eine konkrete Ausprägung eines Entwicklungsprozesses zum Gegenstand. Daher sind Generalisierungen und Abstraktionen hin zu einem allgemeinen Prozessmodell stets von Unsicherheiten begleitet. Zudem erfordert die verwendete Analysemethode, welche nur einen begrenzten Ausschnitt einzelner Aspekte analysieren kann, die Definition geeigneter Prämissen für Aspekte, deren Existenz aufgrund adäquater Annahmen unterstellt werden muß.

Dieser Unsicherheitsfaktor wird durch die Tatsache verstärkt, dass Studien im Kontext von Systementwicklungsprozessen unter dem Paradigma von Open Source in der Regel nur qualitative Daten erheben, was zwar die formalisierte Modellierung der Prozesse nicht aber die Ableitung von quantitativen Aussagen über deren Durchführung ermöglicht. Wie auch [Koch01] feststellt, bedingt das Fehlen eines zentralisierten Projekt-Managements bzw. -Controllings im Rahmen der OSS-Entwicklung, dass die in proprietären Kontexten üblicherweise erhobenen, quantitativen Informationen über die Entwicklungsprozesse nicht zur Auswertung zur Verfügung stehen.

Da der Nutzen dieser Arbeit vor allem aus der empirischen Ermittlung von Informationen und deren strukturierter und formalisierter Darstellung besteht, ist zudem davon auszugehen, dass im Verlauf der Arbeit eine Vielzahl relevanter Informationen entstehen, deren Berücksichtigung in der Arbeit zu einem vergleichsweise großen Umfang führen. Dies ist somit durch die Zielstellung der Arbeit und der beschriebenen Vorgehensweise gerechtfertigt. Soweit es sinnvoll und möglich erscheint, wird im folgenden versucht, Informationen, deren Kenntnis für das weitere Verständnis zwar hilfreich, aber nicht zwingend notwendig ist, in den Anhang der Arbeit oder externe Publikationen auszulagern, um die Komplexität zu verringern und die Lesbarkeit der Arbeit zu verbessern.

[36] „[...] it cannot guarantee the requisite process granularity to inform the design of new software process support technologies.“ [Scacchi98]

[37] „Comparative case analysis is not a panacea. It is not a substitute for other research modalities. Comparative analysis is a secondary assessment of extant studies and empirical findings, not a primary study seeking to identify first-order phenomena that heretofore have not been addressed or discovered.” [Scacchi98]

2 Metamodell zur Prozessanalyse und –modellierung

Dieses Kapitel beschreibt das Metamodell, welches im Rahmen dieser Arbeit sowohl zur Analyse als auch zur Modellierung der Softwareentwicklungsprozesse verwendet wird und auf der UML-Spezifikation 1.3 (vgl. [OMG03][38]) basiert. Dabei werden nur zentrale Aspekte des Metamodells detailliert dargestellt, während die vollständige Beschreibung dieses Metamodells in [Dietze03b] erfolgt.

2.1 Zielsetzung und Methode

In diesem Abschnitt wird die Zielsetzung dargestellt, die mit der Modellierung dieses Metamodells verfolgt wird und die verwendete Vorgehensweise zu dessen Erstellung erläutert.

2.1.1 Zielsetzung

Dieses Metamodell soll als Basis für die strukturierte Durchführung der vergleichenden Fallstudien dienen und die Vergleichbarkeit der Ergebnisse aller Fallstudien gewährleisten, um die Identifikation eines allgemeinen und generalisierbaren Prozessmodells zu ermöglichen. Zudem sollen in diesem Metamodell die verschiedenen Vorgaben zur formalisierten Modellierung des identifizierten Prozessmodells und alle notwendigen Modellierungselemente definiert werden.

Hierzu soll ein allgemeines Metamodell definiert werden, welches alle zu analysierenden und zu modellierenden Aspekte (Entitäten) und deren Beziehungen beschreibt. Anschließend wird die Vorgehensweise zur Dekomposition des gesamten Prozesses in einzelne Prozesse, Teilprozesse und Aktivitäten dargestellt und verschiedene Sichten auf die zu modellierenden Prozesse definiert, welche sich ebenfalls zum Teil am Metamodell der UML orientieren. Darauf aufbauend sollen für jede Sicht Modellierungselemente (Modelltypen, Modellelemente) selektiert werden, welcher zur Modellierung der jeweiligen Sicht verwendet werden sollen. Für jede Sicht wird dabei individuell festgelegt, welche der eingangs definierten Entitäten innerhalb dieser spezifischen Sicht auf den Entwicklungsprozess modelliert werden sollen und welche Modellelemente der UML-Spezifikation zu deren Modellierung vorgesehen werden.

Neben der Metamodellierung der relevanten Aspekte und deren Beziehungen in OSS-Entwicklungsprozessen soll durch das Metamodell also beschrieben werden, wie diese, im Rahmen der Prozessanalyse identifizierten und semantisch beschriebenen Strukturen, ohne bedeutende Informationsverluste in standardisierte Elemente der UML transformiert werden können. Dieses Metamodell dient somit als Ausgangspunkt sowohl für die konsistente und strukturierte Prozessanalyse im Rahmen einer vergleichenden Fallstudie, als auch für die spätere Modellierung der so identifizierten und generalisierten Strukturen.

2.1.2 Methode und Notation

Analog zum Metamodell der UML wird auch dieses Modell basierend auf Bestandteilen der UML-Spezifikation entwickelt und in Klassendiagrammen der UML beschrieben. Von besonderer Bedeutung ist hierbei die Tatsache, dass anhand der folgenden Metamodelle sowohl die zu analysierenden Aspekte der zu betrachtenden Prozesse modelliert, als auch deren Repräsentationsmöglichkeiten durch ausgewählte Modellelemente der UML definiert werden. Zur probaten und erschöpfenden Modellierung aller Aspekte der darzustellenden Prozesse ist es

[38] URL: http://cgi.omg.org/cgi-bin/doc?formal/00-03-01; Abfrage: 12.03.2002

notwendig, die formalen Modellierungsmethoden des SE, respektive die Modelltypen und Sichten der UML geringfügig zu modifizieren [Scacchi98][39]. Daher werden in diesem Metamodell verschiedene Sichten aus dem Metamodell der UML adaptiert und durch weitere, neu definierte ergänzt.

Für jede Sicht des Metamodells wird ein Modell- bzw. Diagrammtyp der UML selektiert, die für die Abbildung der spezifischen Sicht verwendet wird. Weiterhin werden alle relevanten Entitäten der identifizierten Prozessstrukturen spezifiziert, welche durch bestimmte Modellelemente des selektierten UML-Modelltyps abgebildet werden sollen. Anschließend wird eine Teilmenge der in der UML für diesen Modelltyp vorgesehenen Modellelemente ausgewählt, welche die Darstellung aller ausgewählten Aspekte ermöglicht. Diese Einschränkungen und Modifikationen des UML-Metamodells sind notwendig, da das Metamodell der UML nicht explizit und ausschließlich zur Modellierung von Prozessen entwickelt wurde und daher dieser Anforderung nicht adäquat gerecht werden kann. Dies dient u.a. auch der Komplexitätsreduzierung.

Notation
Alle Metaklassen, welche im Rahmen dieses Metamodells auf Modellelemente der UML-Spezifikation verweisen, werden in den folgenden Modellen durch das Präfix *UML_* gekennzeichnet, um eine bessere Differenzierung zwischen den Metaklassen der identifizierten Entitäten und den Metaklassen der Modellelemente der UML zu ermöglichen. Bezüglich der Notation ist weiterhin zu beachten, dass eine nicht anderweitig bezeichnete Kardinalität stets eine 1/1-Kardinalität bezeichnet, da die explizite Darstellung in diesem Fall durch das verwendete CASE-Tool ArgoUML[40] unterdrückt wird. ArgoUML repräsentiert eine UML-Modellierungssoftware, die ebenfalls als OSS in einem OSS-Entwicklungsprojekt nach den Prinzipien des Open Source-Ansatzes entwickelt wird.

2.1.3 SPEM der Object Management Group (OMG)

Inzwischen existiert das sogenannte *Software Process Engineering Metamodel* (SPEM) der OMG, dessen Zielstellungen starke Kongruenzen zu denen des Metamodells dieser Arbeit aufweisen [OMG03][41]. Analog zum Metamodell, welches in diesem Dokument beschrieben wurde, definiert das SPEM eine Teilmenge der UML-Modellelemente, welche zur adäquaten Beschreibung von Softwareentwicklungsprozessen verwendet werden können [OMG03][42].
Da aber zum Zeitpunkt der Arbeit an diesem Metamodell die SPEM-Spezifikation noch nicht durch die OMG veröffentlicht war, konnte dieses Metamodell leider noch nicht berücksichtigt werden. Desweiteren wurden einzelne Aspekte in der SPEM-Spezifikation nicht oder nur geringfügig berücksichtigt, was z.B. auf die in die Entwicklungsprozesse involvierten Softwaretools zutrifft, so dass eine Erweiterung dieses Ansatzes sinnvoll erscheint. Zudem wurde durch die OMG noch keinerlei Fokussierung auf spezifische Domänen vorgenommen und noch keine Kategorisierung der Prozesse bzw. Prozessentitäten in die Spezifikation aufgenommen (vgl. [OMG03][42]), was aber als sinnvoll für die strukturierte Analyse und Beschreibung von Softwareentwicklungsprozessen betrachtet wird und daher einen wichtigen Bestandteil dieses Metamodells darstellt.

[39] „Formal representation of software process data and relations does not readily conform to procedural process programming or conventional object-oriented modeling paradigms." [Scacchi98]

[40] URL: http://argouml.tigris.org/

[41] „This metamodel is used to describe a concrete software development process or a family of related software development processes." [OMG03]; SPEM-Spezifikation 1.0; URL: http://www.omg.org/cgi-bin/apps/doc?formal/02-11-14.pdf; Abfrage: 07.10.2003

[42] „In this specification, we are limiting ourselves to defining the minimal set of process modeling elements necessary to describe any software development process, without adding specific models or constraints for any specific area or discipline, [...]. The SPEM metamodel is [...] formally defined as an extension of a subset of UML called SPEM-Foundation." [OMG03]; SPEM-Spezifikation 1.0; URL: http://www.omg.org/cgi-bin/apps/doc?formal/02-11-14.pdf; Abfrage: 07.10.2003

2.1.4 Abgrenzung zum klassischen Systementwicklungskontext

Das im folgenden dargestellte Metamodell ist mit einigen Einschränkungen auch auf die Modellierung von klassischen Entwicklungsprozessen anwendbar, wobei es aber auf die Spezifikation von Systementwicklungsprozessen im Open Source Kontext hin optimiert wurde. Es werden bereits verschiedene OSSD-spezifische Kategorien aufgrund einer vorausgegangenen Prozessanalyse gebildet und in Form von Metaklassen in das Modell integriert. Diese Kategorien werden z.B. als Spezialisierungen einzelner Metaklassen definiert und dienen der Einteilung dieser Aspekte bzw. der besseren Strukturierung der auf dem Metamodell basierenden Modelle. Diese verschiedenen Metaklassen unterscheiden sich bezüglich ihrer Abgrenzungskriterien z.T. von den Kategorien, die in einem Metamodell im klassischen Systementwicklungskontext sinnvollerweise gebildet werden.

Weiterhin werden verschiedene Sichten modelliert, welche die identifizierten Prozesse möglichst erschöpfend in all ihren Facetten abbilden. Auch die Modellierungsmethoden dieser Sichten wurden bereits unter dem Gesichtspunkt definiert, diese anschließend in dem spezifischen Kontext der OSS-Entwicklung einzusetzen.
So stellt z.B. die Verwendung der Anwendungsfallsicht gerade im OS-Kontext eine adäquate Methode zur Modellierung und Strukturierung einzelner Prozesse und Teilprozesse dar. Dies resultiert vor allem aus dem parallelen Charakter der Prozesse, der auf dieser Abstraktionsebene eine Darstellung der zeitlichen Abfolge der einzelnen Prozesse und Teilprozesse obsolet macht und die Modellierung der zeitlichen Dimension erst auf einer tieferen Abstraktionsebene sinnvoll erscheinen lässt. Diesem Umstand wird z.B. im Abschnitt *2.4 Prozessspezifische Sichten* Rechnung getragen.

2.2 Metamodell der zu betrachtenden Entitäten

Die isolierte Betrachtung einzelner Aspekte, respektive der soziologischen oder der technologischen Aspekte, könnte nach [Scacchi01b][43] im Rahmen der Ableitung von verallgemeinerbaren Aussagen zu fehlerhaften Rückschlüssen und Ergebnissen führen, daher erscheint nur die gesamtheitliche Betrachtung der wichtigsten Entitäten im OSS-Entwicklungsprozess sinnvoll.

Die verschiedenen Typen der folgenden grundlegenden Entitäten und ihre Beziehungen sollen identifiziert und im Rahmen der verschiedenen Sichten dargestellt werden:

- Rolle (*Role*)
- Artefakt (*Artifact*)
- Werkzeug (*Tool*)
- Prozess (*Process*)

Die folgende Grafik visualisiert das Metamodell der zu analysierenden und zu modellierenden Entitäten, wobei die Dekomposition der Prozesse und deren Integration in dieses Metamodell in einem expliziten Abschnitt dargestellt wird:

[43] „A comparative empirical study of open software work structures, processes and practices from socio-technical perspective […] must concurrently examine the social arrangements, work processes and practices from which open software communities and artifacts arise, as well as the technical information infrastructure through which these communities and artifacts are articulated and shared." [Scacchi01b]

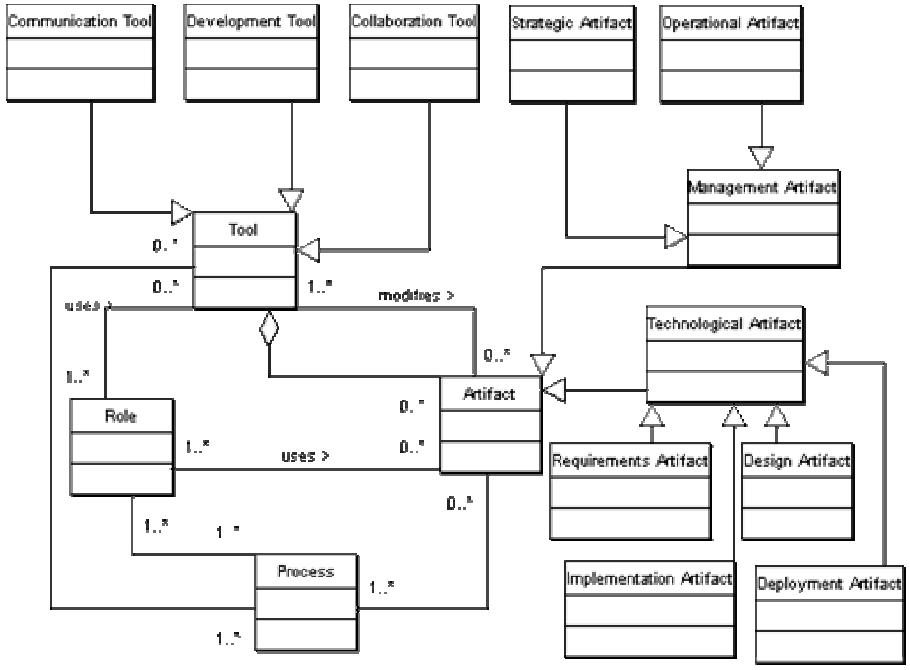

Abb. 2.1: Metamodell der zu betrachtenden Entitäten

Das hier dargestellte Metamodell orientiert sich in den Grundzügen an dem Metamodell der klassischen Softwareentwicklung in [AcFe01] und wurde um verschiedene allgemeine und OS-spezifische Elemente erweitert. Im folgenden werden die einzelnen Entitäten explizit beschrieben.

2.2.1 Prozesse

Prozesse repräsentieren im Rahmen dieser Arbeit eine Menge logisch zusammenhängender Anwendungsfälle bzw. Teilprozesse. In einen Prozess sind i.d.R. mehrere Rollen, Werkzeuge und Artefakte involviert, wie bereits in *Abb. 2.1* dargestellt wurde.

Das im folgenden dargestellte Klassendiagramm repräsentiert die Dekomposition des Prozessmodells in Prozesse (*Process*), Teilprozesse (*Sub Process*), Aktivitäten (*Activity*) und atomare Aktivitäten (*Atomic Activity*) in Abhängigkeit vom Abstraktionsniveau und visualisiert die Spezialisierungen eines Prozesses in infrastrukturelle (*Environment Process*), Entwicklungs- (*Development Process*) und Managementprozesse (*Management Process*).

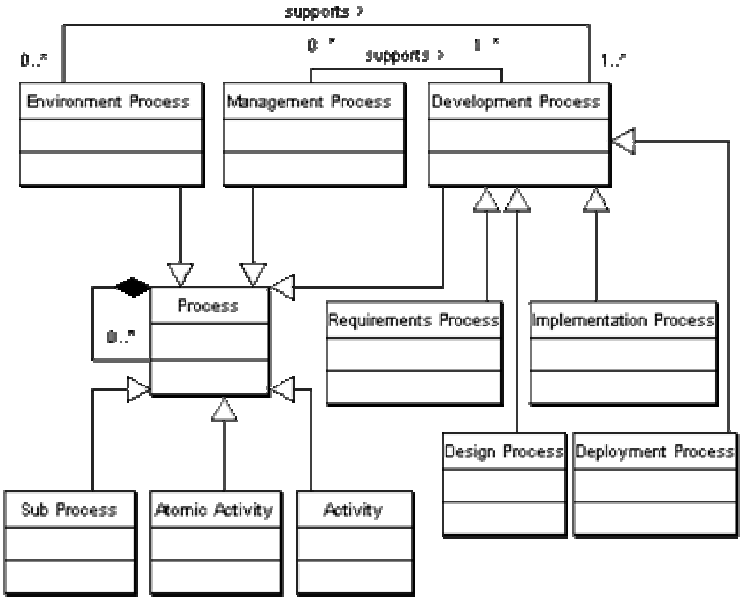

Abb. 2.2: Prozessdekomposition und -spezialisierung

Prozesse lassen sich im Rahmen dieses Metamodells basierend auf einer ersten Klassifizierung in Anlehnung an [Scacchi02] in die folgenden Kategorien einteilen:

Entwicklungsprozesse (Mesostructure)
Alle Prozesse zur Softwareentwicklung werden in der Kategorie der Entwicklungsprozesse vereint und beinhalten primär alle dezentralen Aktivitäten einzelner Individuen der Community, wie Anwender oder Entwickler und die von ihnen erbrachten Beiträge zu dem spezifischen Projektziel, respektive der Weiterentwicklung der jeweiligen Software bzw. peripherer Artefakte. Alle Prozesse, welche direkt zur Weiterentwicklung der OSS bzw. peripherer Artefakte beitragen, werden somit der Kategorie der Entwicklungsprozesse zugeordnet. Zudem zeichnen sich Entwicklungsprozesse dadurch aus, dass sie primär durch die dezentral verteilten Akteure der Projektgemeinde ausgeführt werden und i.d.R. die Erstellung, Modifikation oder Verwendung eines technologischen Artefakts (vgl. *2.2.3 Artefakt*) zum Ziel haben.

Diese Prozesse beinhalten z.B. das Entwickeln von Patches, den Beitrag von Dokumentationsartefakten oder die Erstellung von Anforderungsartefakten. Die einzelnen Entwicklungsprozesse lassen sich somit den klassischen Phasen der Softwareentwicklung zuordnen, was zur Definition der folgenden spezialisierten Prozesskategorien geführt hat:

- Anforderungsanalyse Prozess (*Requirements Process*)
- Entwurfsprozess (*Design Process*)
- Implementationsprozess (*Implementation Process*)
- Einsatz- bzw. Anwendungstest-Prozess (*Deployment Process*)

Managementprozesse (Suprastructure)
Diese Klasse der Prozesse beinhaltet alle relevanten Prozesse zur Organisation und Koordination der Entwicklungsprozesse und wird nicht phasenspezifisch sondern –übergreifend ausgeführt. Diese Prozesse werden i.d.R. ausschließlich durch die zentralen Rollen zur Maintenance des Projekts ausgeführt und schaffen die organisatorischen Rahmenbedingungen für die Entwicklungsprozesse der Community. Durch die Definition der strategischen und operativen Managementartefakte, einem wichtigen Teilziel, welches mit den Managementprozessen verfolgt wird, werden die grundlegenden Voraussetzungen für die Entwicklung der OSS

geschaffen. Zudem ist die Etablierung, Organisation und Koordination der Community und der involvierten Entwicklungsprozesse ein wichtiger Bestandteil dieser Prozesse.

Neben dem koordinativen Aspekt umfassen diese Prozesse insbesondere die Entscheidungsprozesse, die von strategischer und operativer Bedeutung sind, wie z.B. Entscheidungen über die strategische Zielsetzung des Projekts oder operative Entscheidungen über den Zeitpunkt, die Art und den Umfang eines neuen Release.

Infrastrukturelle Prozesse (Infrastructure)
Die infrastrukturellen Prozesse werden auch Environment Processes genannt und beinhalten alle Aktivitäten, welche ausschließlich die Entwicklung und Verwaltung einer adäquaten Infrastruktur für die Durchführung der kollaborativen Entwicklungsprozesse des OSS-Projektes verfolgen. Wie auch die Managementprozesse werden Prozesse dieser Metaklasse durch die zentralen Maintenance Instanzen ausgeführt bzw. koordiniert und dienen lediglich der Unterstützung der dezentralen Entwicklungsprozesse.

Ein Prozess definiert die oberste Betrachtungsebene, wie in *Abb. 2.2* dargestellt wurde, und wird auf einem detaillierteren Betrachtungsniveau in Teilprozesse unterteilt, welche wiederum aus Aktivitäten und atomaren Aktivitäten bestehen können. Es ist hierbei zu beachten, dass alle Bestandteile der Prozessdekomposition auch Bestandteil von sich selbst sein können. Dies bedeutet, dass ein Teilprozess z.B. auch Bestandteil eines Teilprozesses sein kann, aber auf einer anderen Abstraktionsebene betrachtet wird. Die genaue Differenzierung zwischen diesen Begriffen erfolgt in [Dietze03b].

2.2.2 Rolle

[AcFe01] definiert den Rollenbegriff als *„a set of agent or group responsibilities, rights and skills required to perform a specific software process activity.”*. Rollen repräsentieren somit eine abgrenzbare Menge von Aufgabenbereichen, Privilegien und Eigenschaften, die sinnvoll zu einer expliziten Entität aggregiert werden können.

2.2.3 Artefakt

[KrKr03][44] beschreibt ein Artefakt als ein Informationsobjekt, welches produziert, verändert oder verwendet wird. Die Metaklasse *Artifact* repräsentiert alle identifizierten Informationsobjekte, welche für Prozesse relevant sind. Damit stellen von dieser Klasse abgeleitete Artefakttypen die Input- und Outputobjekte von Aktivitäten dar.

Sofern im folgenden die Begriffe *Modul* bzw. *Component* verwendet werden, bezeichnen diese lediglich ein funktional abgrenzbares Quellcode-Artefakt entsprechend der Terminologie einiger untersuchter OSS-Projekte und werden zudem synonym verwendet.

Artefakte bezeichnen im Kontext dieses Metamodells lediglich eine bestimmte Klasse von Informationsobjekten, also Inhalten, und nicht die Medien, die als Container für diese Inhalte dienen. Von Relevanz für das Artefakt sind in diesem Zusammenhang lediglich die Restriktionen, welche das verwendete Werkzeug an das Format des Artefakts stellt.

Im folgenden werden verschiedene Kategorien beschrieben, welche die evoluierenden Artefakte im Rahmen eines iterativen Software-Entwicklungsprozesses in verschiedene Teilmengen mit jeweils spezifischen Eigenschaften unterteilen. Hierbei ist zu beachten, dass auf der Ebene des zu erstellenden Prozessmodells nie die Instanzebene modelliert wird, sondern lediglich identifizierte Typen einer Klasse in den verschiedenen Sichten auf der Modellebene dargestellt werden (vgl. *Tab. 1.1*).

[44] „An artifact is a piece of information that is produced, modified, or used by a process.“ [KrKr03]

Managementartefakte (*Management Artifacts*)
Diese Artefakte dienen der Planung, Kontrolle und Steuerung des Software-Entwicklungsprozesses und basieren i.d.R. auf einer informellen Notation wie Freitext, Grafiken oder Tabellen. Die Managementartefakte werden i.d.R. zentral durch die Maintenance-Instanzen des Projekts erstellt und verfolgen lediglich eine unterstützende bzw. koordinierende Zielsetzung, indem sie die primären Entwicklungsprozesse ermöglichen, unterstützen oder koordinieren.

Die Managementartefakte werden im Kontext dieses Metamodells in die folgenden Teilmengen unterteilt:

- Strategische Artefakte (*Strategic Artifacts*)
- Operationale Artefakte (*Operational Artifacts*)

Die strategischen Artefakte repräsentieren hierbei langfristig definierte Informationsobjekte, wie z.B. Softwarelizenzen, Programmierstandards oder Projektleitlinien, die im Laufe eines OSS-Projekt nur selten modifiziert werden. Aufgrunddessen sind sie von strategischer Bedeutung für das Gesamtprojekt und definieren die Rahmenbedingungen für die gesamten Entwicklungsprozesse.

Die operationalen Managementartefakte umfassen alle organisatorisch eingesetzten Artefakte, welche nur temporär über einen bestimmten Zeitraum wirksam sind und die Kontrolle und Steuerung der operativ ausgeführten Entwicklungsprozesse zum Ziel haben. Die operativen Managementartefakte werden vergleichsweise häufig im Rahmen einzelner Aktivitäten durch die Akteure in den individuellen Entwicklungsprozessen aktualisiert.

Technologische Artefakte (*Technological Artifact*)
Die Entwicklung der technologischen Artefakte ist das primäre Ziel der OSS-Entwicklungsprojekte. Technologische Artefakte werden im Rahmen der Entwicklungsprozesse entwickelt, verwendet und modifiziert. Ein technologisches Artefakt kann stets als deskriptives bzw. präskriptives Modell betrachtet werden [Ludewig02][45]. Diese Menge der Artefakte orientiert sich an den dargestellten Phasen des Softwareentwicklungsprozesses und wird daher in die folgenden Kategorien eingeteilt:

- Anforderungsartefakt (*Requirements Artifact*)
- Entwurfsartefakt (*Design Artifact*)
- Implementationsartefakt (*Implementation Artifact*)
- Einsatzartefakte (*Deployment Artifact*)

Die Anforderungsartefakte spezifizieren somit explizit oder implizit die Anforderungen an den zu erstellenden Softwarebaustein und sind i.d.R. im Rahmen der OSS-Entwicklung nicht formal spezifiziert.

Designartefakte beinhalten Angaben über die Art und Weise der Realisierung eines Subsystems und definieren die Rahmenbedingung für die Implementierung, wie z.B. Architektur, Modularisierung oder Schnittstellen.

Implementierungsartefakte beinhalten alle implementationsabhängigen Artefakte und sind somit von besonderer Bedeutung in der OSS-Entwicklung, die sich stark an der Entwicklung und sukzessiven Konkretisierung von prototypischen Softwaremodulen orientiert.

[45] „In einer systematischen Software-Entwicklung [...] entstehen nacheinander Dokumente, die Modell-Eigenschaften aufweisen [...]." [Ludewig02]

Die Klasse der Einsatzartefakte hingegen enthält alle Informationsobjekte, welche sich auf die Anwendung der Software beziehen und umfassen somit auch die kompilierten und ausführbaren Binärdateien oder Dokumentationsartefakte.

Diese Subkategorien der Artefakte werden ebenfalls als Metaklassen in das Metamodell integriert, wobei jedes identifizierte Artefakt eindeutig einer dieser Kategorien zugeordnet werden sollte.

2.2.4 Werkzeuge

Verwendete Software-Werkzeuge zur Software-Entwicklung und sekundären Aktivitäten werden in der abstrakten Metaklasse *Tool* repräsentiert. Werkzeuge sind in Prozesse, Teilprozesse und Aktivitäten involviert und werden von Akteuren verwendet oder auch modifiziert, da es im Rahmen von OSS-Projekten eine gängige Praxis darstellt, entsprechende Tools innerhalb der eigenen Projekt-Community auf OSS-Basis eigenständig zu entwickeln oder zu modifizieren. Weiterhin werden diese Werkzeuge dazu genutzt, die Artefakte zu verwenden oder zu modifizieren und können somit zu einer Zustandsänderung eines dieser Objekte beitragen.

Die Software-Werkzeuge wurden entsprechend ihres Verwendungszwecks und in Anlehnung an [Scacchi02] in die folgenden Kategorien eingeteilt:

- Kollaborationswerkzeug (*Collaboration Tool*)
- Kommunikationswerkzeug (*Communication Tool*)
- Entwicklungswerkzeug (*Development Tool*)

Diese Kategorien werden ebenfalls durch von der Oberklasse *Tool* abgeleitete Unterklassen in das Metamodell integriert.

Kollaborationswerkzeuge repräsentieren somit die peripheren und projektweit angewendeten Werkzeuge, welche die kooperative Arbeit der Community unterstützen sollen, wie z.B. Werkzeuge für das Konfigurationsmanagement und die kooperative Softwareentwicklung (z.B. CVS). Die Metaklasse der Kommunikationswerkzeuge umfasst alle Werkzeuge, welche primär zur projektinternen und –externen Kommunikation verwendet werden, wie z.B. Mailing Listen oder Newsgroups. *Development Tool* hingegen bildet die Oberklasse für alle relevanten Werkzeuge, die direkt in den Entwicklungsprozess integriert sind und somit primär auf Seiten der Entwickler und Anwender zum Einsatz kommen, wie z.B. Editoren oder Compiler.

2.3 Sichtendekomposition

Das Metamodell dieser Arbeit strukturiert die darzustellenden Aspekte anhand von Sichten, welche zum Teil an den Sichten des UML-Metamodells ausgerichtet sind und deren Modellelemente durch verschiedene Einschränkungen und Erweiterungen modifizieren. Dadurch werden spezifische Sichten definiert, welche möglichst optimal die Modellierung von Softwareentwicklungsprozessen im allgemeinen und der OSS-Entwicklung im speziellen unterstützen. Da diese Sichten lediglich der Prozessmodellierung dienen, werden die implementationspezifischen Darstellungsformen der UML dabei nicht berücksichtigt.

Die Sichten werden unterteilt in prozessbezogene (*Processrelated View*) und entitätsbezogene Sichten (*Entityrelated View*), wobei die prozessbezogenen zur Darstellung von einem oder mehreren bestimmten (Teil-)Prozessen bzw. Aktivitäten dienen und die entitätsbezogenen Sichten zur Darstellung verschiedener Aspekte einer oder mehrerer Entitäten über das gesamte Prozessmodell hinweg verwendet werden können.

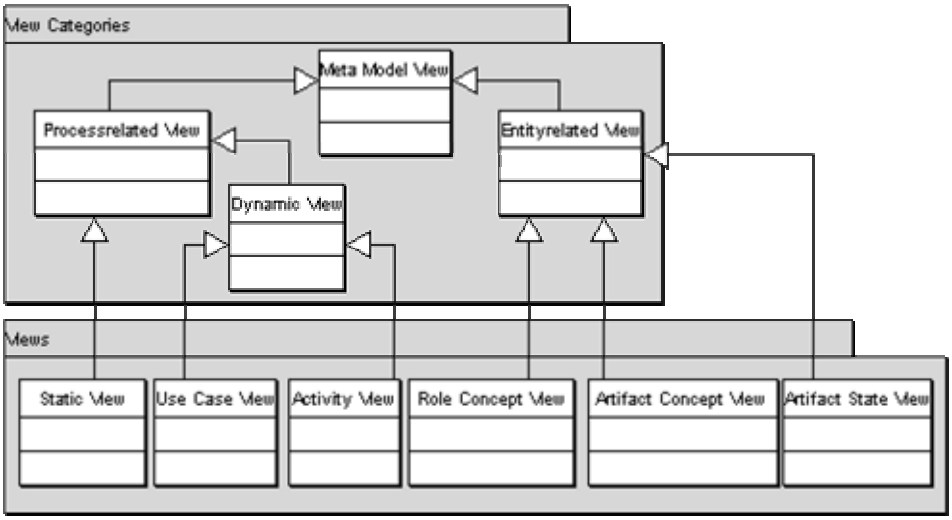

Abb. 2.3: Dekomposition der Sichten des Metamodells

Die prozessbezogenen Sichten werden weiter unterteilt in die statische Sicht (*Static View*) und verschiedene dynamische Sichten *(Dynamic View*). Die dynamischen Sichten setzen sich dabei aus der Anwendungsfall- oder auch Use Case-Sicht (*Use Case View*) und der Aktivitätssicht (*Activity View*) zusammen, die sich sehr eng am Metamodell der UML orientieren. Die prozessübergreifenden, entitätsbezogenen Sichten werden unterteilt in Sichten welche die prozessübergreifenden Beziehungen der Rollen (*Role Concept View*) und der Artefakte (*Artifact Concept View*) und die Zustände bzw. Lebenszyklen einzelner Artefakte (*Artifact State View*) darstellen.

Ein wichtiger Grund für die Notwendigkeit zur Modifikation der Sichten des Metamodells der UML ist die Vielzahl der Überschneidungen der UML-Sichten, die somit zu etwaigen Redundanzen führen können [Wortmann01][46]. Zudem ist zur Modellierung von Prozessen und deren inhärenter Strukturen nur eine Teilmenge der UML-Diagrammtypen notwendig. Physische Sichten anhand von Component Diagrams oder Deployment Diagrams wurden hierbei ausgegrenzt, da dieses Metamodell nicht die Implementation eines Systems, sondern die Analyse respektive die Spezifikation von Prozessen zum Ziel hat und somit implementationsbezogene Sichten keine Relevanz für das hier beschriebene Metamodell besitzen. Damit orientiert sich das verwendete Metamodell weitgehend an den Sichten und Diagrammen, welche im Metamodell der UML zur Definition der Anforderungen an ein Software-System, respektive den zugrundeliegenden Prozessen und Workflows, definiert wurden.

Im folgenden werden die verschiedenen prozessspezifischen Sichten detailliert beschrieben und die zur Erstellung notwendigen Modellelemente der UML dargestellt. Auf die Darstellung der Metamodelle der entitätsbezogenen, prozessübergreifenden Sichten, die in [Dietze03b] detailliert vorgestellt werden, wird im folgenden verzichtet, da diese i.d.R. nur einmalig erstellt werden und daher von geringerer Bedeutung als die prozessbezogenen Sichten sind.

2.4 Prozessspezifische Sichten

Wie bereits im vorangegangenen Abschnitt dargestellt wurde, werden die identifizierten Prozesse, Teilprozesse und Aktivitäten anhand verschiedener Sichten modelliert, die eine möglichst verlust- und auch redundanzfreie Modellierung der Strukturen auf Basis des Metamodells der UML ermöglichen sollen.

[46] „Due to conceptual overlaps [...], UML viewpoints cannot be considered as orthogonal.“ [Wortmann01]

Das nachfolgende Modell repräsentiert die Beziehungen zwischen den einzelnen Prozesselementen, den zu modellierenden, prozessbezogenen Sichten und den ausgewählten Darstellungsmethoden der UML:

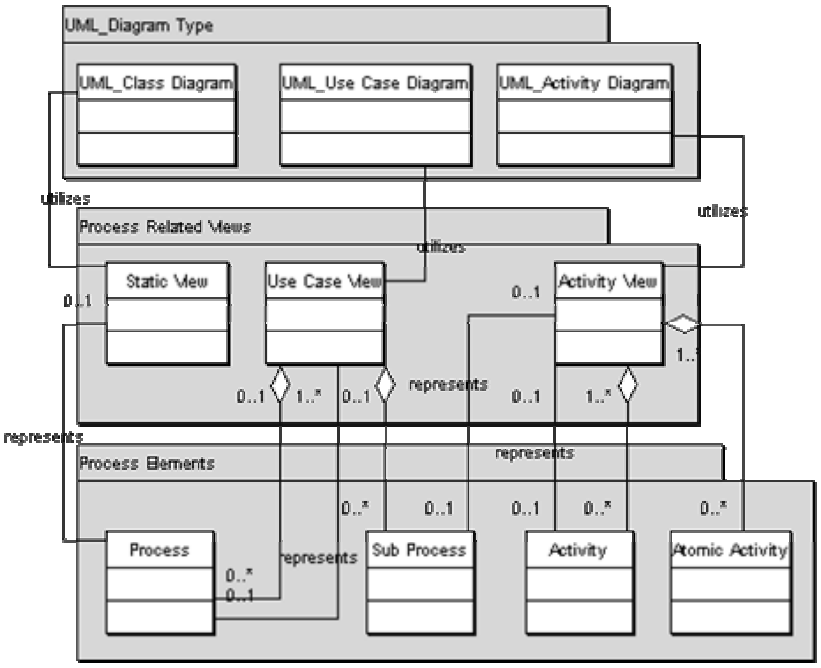

Abb. 2.4: Abbildungsmöglichkeiten der Prozesselemente auf prozessbezogene Sichten

Prozesse
Ein bestimmter Prozess wird anhand von einer oder mehreren Use Case-Sichten abgebildet. Die Verwendung der Anwendungsfallsicht zur Dekomposition eines Prozesses in Prozesse und Teilprozesse erscheint im Open Source-Kontext als besonders geeignetes Mittel, da die Prozesse und Teilprozesse weitgehend parallel zueinander ausgeführt werden und somit die Berücksichtigung der zeitlichen Dimension auf dieser Abstraktionsebene, respektive bei der Dekomposition eines Prozesses, obsolet erscheint.

Die statischen Beziehungen der involvierten Rollen, Artefakte und Werkzeuge werden in maximal einer statischen Sicht definiert, welche in den folgenden Abschnitten noch konkret dargestellt wird.

Teilprozesse
Zur Modellierung der Ablaufdynamik des Teilprozesses, also zur Darstellung der sequentiellen bzw. parallelen Abfolge der involvierten Aktivitäten entsprechend ihrer Ausführungsreihenfolge, wird auf der Abstraktionsebene eines Teilprozesses eine Aktivitätssicht für einen spezifischen Teilprozess erstellt. Hierbei können voneinander abhängige Teilprozesse in einer gemeinsamen Aktivitätssicht modelliert werden, um deren logische Verknüpfung zum Ausdruck zu bringen.

Aktivität und Atomare Aktivität
Aktivitäten und atomare Aktivitäten können als Bestandteil der Aktivitätssicht eines Teilprozesses modelliert werden. Weiterhin besteht die Möglichkeit, eine bestimmte Aktivitität auf einer granulareren Modellierungsebene in einer detaillierteren Aktivitätssicht zu modellieren. Diese Konkretisierung kann bis zu einer Detaillierungsebene durchgeführt werden, auf welcher die modellierte Aktivitätssicht ausschließlich aus atomaren Aktivitäten besteht. Bezüglich der UML-Notation wird in den Modellen dieser Arbeit nicht zwischen Action- und Activity States unterschieden und lediglich insofern zwischen atomaren und nicht atomaren Aktivitäten differenziert, als dass für atomare Aktivitäten keine weitere Detaillierung der Aktivität vorgenommen wird.

Im Rahmen der Wahlmöglichkeiten zwischen verschiedenen Darstellungsformen (Sichten) bzw. der Anzahl der zu modellierenden Sichten für einen bestimmten (Teil-)Prozess kann im Einzelfall entschieden werden, welche Sichten zur effizienten und sinnvollen Modellierung verwendet werden, ohne bedeutende Informationsverluste in Kauf zu nehmen. Außerdem erscheint es sinnvoll, einige Teilprozesse in einer gemeinsamen Prozessbetrachtung zu modellieren, sofern Schnittstellen bzw. Abhängigkeiten zwischen diesen Prozessen identifiziert wurden.

2.4.1 Statische Sicht

Eine statische Sicht auf einen individuellen Prozess wird anhand von Klassendiagrammen modelliert, welche die relevanten Entitäten und deren statische Beziehungen darstellen. Dazu werden die standardisierten Modellelemente der UML-Klassendiagramme der Spezifikation 1.3 verwendet.
Nach [CoDa94] kann ein Klassendiagramm innerhalb einer statischen Sicht verschiedene Perspektiven auf eine Problemstellung einnehmen, die aber nicht klar voneinander abgegrenzt und nicht formaler Bestandteil der UML sind:

- Konzeptuelle Perspektive
- Spezifikationsperspektive
- Implementationsperspektive

Die konzeptuelle Perspektive repräsentiert dabei die Konzepte der modellierten Problemdomäne und sollte implementationsunabhängig modelliert werden [Fowler00][47]. Im Gegensatz dazu dient die Spezifikationsperspektive bereits der Definition von Interfaces, während die Implementationsperspektive die konkreten Details der Implementierung betrachtet. Für den Rahmen dieser Arbeit, in welcher die Klassendiagramme lediglich die statischen Strukturen der modellierten Entitäten darstellen werden, ist daher lediglich die konzeptuelle Perspektive von Bedeutung.

Im Rahmen der Modellierung der statischen Sicht eines spezifischen Prozesses werden i.d.R. mehrere Teilprozesse bzw. Aktivitäten und deren involvierte Rollen, Werkzeuge und Artefakte modelliert. Zum besseren Verständnis der Modelle werden im Rahmen der statischen Sicht alle Klassen mit einem Präfix markiert, der die Zugehörigkeit zu der Metaklasse der jeweiligen Entität signalisiert:

- Rolle: *R*
- Artefakt: *A*
- Tool: *T*
- Teilprozess bzw. Aktivität: *P*

Zwischen diesen Entitäten können Beziehungen dargestellt werden. So kann eine Rolle innerhalb eines modellierten Prozesses z.B. Beziehungen mit mehreren Werkzeugen, Aktivitäten und Artefakten aufweisen. Die Beziehungen zwischen den bezeichneten Entitäten werden durch die Modellelemente für Relationen in UML Klassendiagrammen wie Assoziationen, Aggregationen, Kompositionen oder Generalisierungen modelliert. Assoziationen dienen der Modellierung statischer Relationen zwischen verschiedenen Klassen innerhalb des modellierten Prozesses [Fowler00][48]. Navigierbare Assoziationen spielen im Kontext konzeptueller Modelle nur eine untergeordnete Rolle und wurden daher nicht in das Metamodell integriert. Außerdem ist zu beachten, dass eine Klasse Beziehungen mit sich selbst eingehen kann, was durch die entsprechenden Kardinalitäten der Assoziation zwischen den Metaklassen *UML_Class* und *UML_Relationship* berücksichtigt wurde.

[47] „[…] a conceptual model should be drawn with little or no regard for the software that might implement it, so it can be considered language-independent." [Fowler00]

[48] „From the conceptual perspective, associations represent conceptual relationships between classes." [Fowler00]

Die folgende Grafik stellt die zu verwendenden Modellelemente im Rahmen der statischen Sicht dar, welche eine Teilmenge der Modellelemente der statischen Sicht des UML-Metamodells repräsentieren.

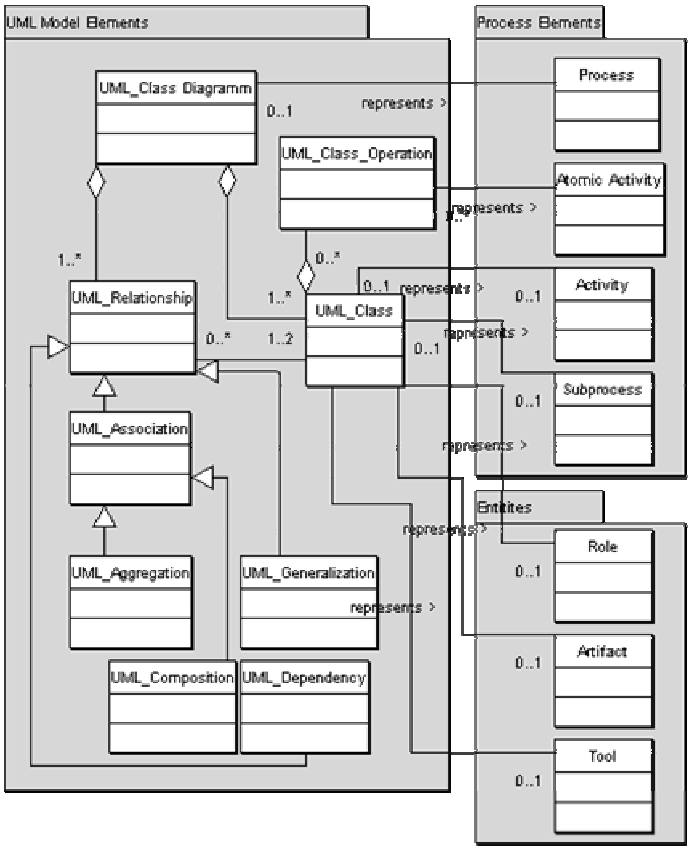

Abb. 2.5: Metamodell zur Modellierung der statischen Sicht

2.4.2 Dynamische Sichten

Neben der statischen Sicht wurden im Rahmen dieses Metamodells auch verschiedene dynamische Sichten auf den Prozess definiert.

2.4.2.1 Use Case-Sicht

Die Use Case-Sicht im Rahmen dieses Metamodells basiert auf der Anwendungsfallsicht des UML-Metamodells und wird somit anhand von Use Case Diagrammen abgebildet. Innerhalb dieser Perspektive werden die involvierten Rollen als Akteure und die von ihnen ausgelösten Prozesse bzw. Teilprozesse als Use Case modelliert. In Abgrenzung dazu werden innerhalb dieser Sicht keine (atomaren) Aktivitäten modelliert, da diese erst in der Konkretisierung eines Teilprozesses anhand von Aktivitätsdiagrammen als *Action*- bzw. *Activity States* Berücksichtigung finden. Die Anwendungsfallsicht visualisiert die Aufgabenbereiche und Verantwortlichkeiten der einzelnen Rollen im Prozessmodell und modelliert anhand der Bezeichnungen der einzelnen Use Cases auch implizit die Rechte und Pflichten der jeweiligen Rolle. Use Case Diagramme können über verschiedene Abstraktionsebenen hinweg modelliert werden und somit Prozesse entlang verschiedener Konkretisierungsebenen in Prozesse oder Teilprozesse unterteilen.

Akteur

„Ein Akteur repräsentiert eine kohärente Menge von Rollen, die Anwender eines Anwendungsfalles spielen, wenn sie mit diesen Anwendungsfällen interagieren." [BoRuJa99] Um vollständige Kohärenz zu gewährleisten, repräsentiert ein Akteur im Rahmen dieses Metamodells genau eine Rolle innerhalb des Prozesses und muß somit in der statischen Sicht auch eine Entsprechung basierend auf der Metaklasse *Role* aufweisen.

Use Case
Ein Use Case repräsentiert i.d.R. einen Prozess, einen Teilprozess bzw. einen Spezialfall eines (Teil-)Prozesses und wird von einem Akteur ausgelöst. Als Use Case wird somit ein Anwendungsfall bzw. ein (Teil-)Prozess dargestellt, von dem ebenfalls verschiedene Varianten abgeleitet werden können und der Beziehungen zu anderen Use Cases aufweisen kann. Use Case Modelle können nach [BoJuRa99] auch zur Modellierung von verschiedenen Varianten eines spezifischen Prozesses verwendet werden. Ziel bei der Modellierung der Use Case-Sicht im Rahmen dieser Arbeit ist es aber, nur die primären und idealtypischen Szenarien bzw. Teilprozesse zu definieren und sekundäre Varianten dieser Szenarien nur in das Modell aufzunehmen, wenn diese von elementarer Bedeutung für den Prozess als Gesamtheit sind.

Zwischen den einzelnen (Teil-)Prozessen können Beziehungen existieren, welche basierend auf den zur Verfügung stehenden Relationen zwischen Use Cases dargestellt werden können. So kann ein Anwendungsfall einen anderen erweitern (*UML_Extension*), einen anderen direkt beinhalten (*UML_Inclusion*) oder von einem anderen Teilprozess abgeleitet sein (*UML_Generalization*).

Das folgende Modell stellt die verwendete Teilmenge der Modellelemente der UML dar, welche zur Modellierung der Anwendungsfallsicht verwendet werden und zeigt die möglichen Beziehungen zu den zu modellierenden Entitäten und Prozesselementen auf:

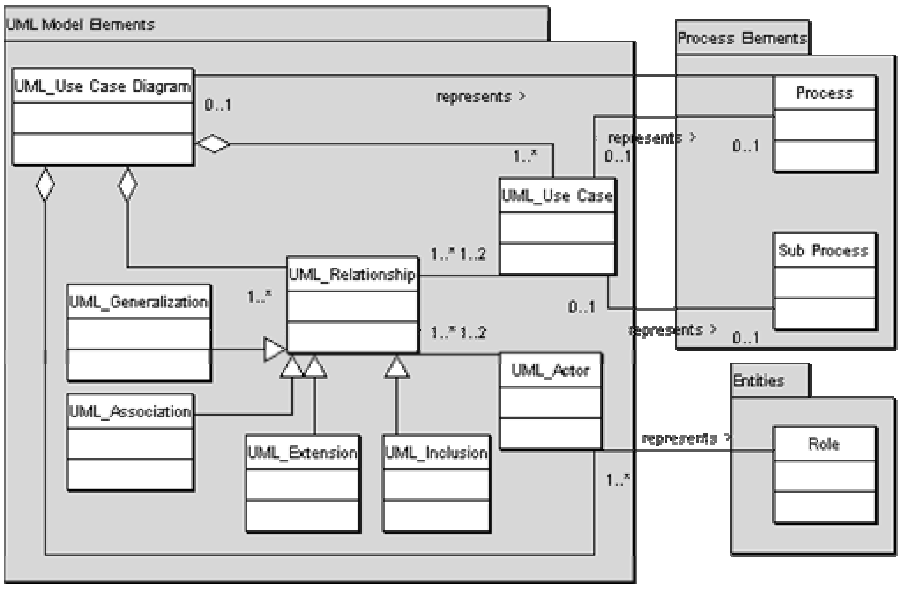

Abb. 2.6: Metamodell zur Modellierung der Anwendungsfallsicht

2.4.2.2 Aktivitätssicht

Die Aktivitätssicht dient der Visualisierung der zeitlichen Abfolge der Ausführung von Aktivitäten und der dafür verantwortlichen Rollen. Sie wird anhand von Aktivitätsdiagrammen der UML modelliert, welche entweder einen Teilprozess repräsentieren oder auch zur granulareren Konkretisierung einer Aktivität verwendet werden können. Aktivitätsdiagramme der UML stellen eine probate Methode zur Spezifikation von (Geschäfts-)prozessen dar (vgl.

[JaBoRu99]) und sind daher auch eine geeignete Methode zur Konkretisierung einzelner Teilprozesse und Workflows, da sie es ermöglichen, die zeitlich geordnete Abfolge von Teilprozessen und Aktivitäten zu modellieren.

Hauptbestandteil der Aktivitätsdiagramme sind die sogenannten Action-States der UML, welche jeweils eine einzelne Aktivität repräsentieren. Bezugnehmend auf [Fowler00] wird im Rahmen dieser Arbeit nicht zwischen Action- und Activity- States unterschieden, und sowohl Aktivitäten als auch atomare Aktivitäten werden als Action-States modelliert. Aktivitäten können bis zu einer Detaillierungsebene in weiteren Aktivitätsdiagrammen konkretisiert werden, auf welcher das erstellte Diagramm ausschließlich aus atomaren Aktivitäten besteht.

Neben den sequentiellen Aktivitäten lassen sich anhand der in Abschnitt 2.1 dargestellten Modellelemente *UML_Fork* bzw. *UML_Join* parallel ausgeführte Aktivitäten visualisieren. Die Elemente *UML_Branch* bzw. *UML_Merge* dienen der Modellierung von bedingt ausgeführten Aktivitäten. Anhand von *UML_Swimlanes* lassen sich Verantwortlichkeiten modellieren. Somit ist auch die Zuordnung von Rollen, welche die modellierten Aktivitäten ausführen, in dieser Ansicht modellierbar.

Der Zusammenhang zwischen den modellierten Teilprozessen bzw. Aktivitäten und den involvierten Artefakten und Werkzeugen wird bereits in der statischen Sicht des gesamten Prozesses verdeutlicht, in welcher alle relevanten Teilprozesse und Aktivitäten des Prozesses als Klasse modelliert werden und die Beziehungen zu den jeweils involvierten Artefakten, Werkzeugen und Rollen in Form von Assoziationen dargestellt werden. Weiterhin läßt sich nach [BoRuJa99][49] auch der Objektfluß bzw. die an einem Kontrollfluss beteiligten Objekte in der Aktivitätssicht modellieren, um die Aussagekraft der Modelle zu verbessern. Dies kann Verwendung finden, um die involvierten Artefakte und deren Zustandsänderung direkt in dieser Sicht zu formulieren.

Eine weitere Möglichkeit zur Integration zusätzlicher Informationen ist die Entwicklung von sogenannten Tagged Values, welche als Container für Informationen dienen können. Bei der Modellierung von Workflows bzw. Geschäftsprozessen „*liegt das Hauptinteresse auf den Aktivitäten, wie sie von Akteuren wahrgenommen werden, die mit dem System kollaborieren.*“. [BoRuJa99]. Daher sind Aktivitätsdiagramme im Rahmen der Prozessmodellierung von besonderer Bedeutung.

Das folgende Modell definiert die Entitäten, welche in der Aktivitätssicht modelliert werden sollen und die dazu zu verwendenden Modellelemente der UML. Diese repräsentieren eine Teilmenge der Modellelemente, welche die UML-Spezifikation zur Modellierung von Aktivitätsdiagrammen vorsieht.

[49] „Diese Verwendung von Abhängigkeitsbeziehungen und Objekten heißt Objektfluß (object flow), da sie die Beteiligung von Objekten an einem Kontrollfluß darstellt.” [BoRuJa99]

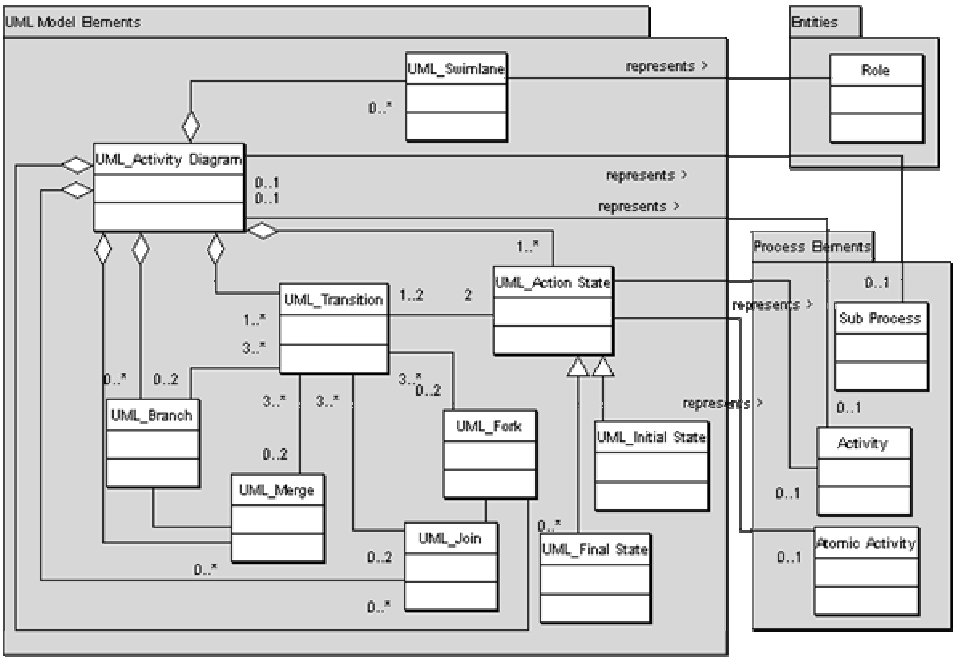

Abb. 2.7: Metamodell der Aktivitätssicht basierend auf Aktivitätsdiagrammen der UML

Auf einer vollständigen Erläuterung des Metamodells wird an dieser Stelle verzichtet, da dies bereits in [Dietze03b] dokumentiert wurde, wobei dies auch eine detaillierte Darstellung der entitätsbezogenen Sichten und eine Beschreibung der Vorgehensweise zur Prozessmodellierung basierend auf diesem Metamodell umfasst.

3 Charakteristika und gesamtheitliches Modell der OSS-Entwicklung

„There's a vague sense abroad that each such project is sui generis and stands or falls on the group dynamic of its particular members, but is this true or are there replicable strategies a group can follow?“ [Raymond98]

In den folgenden Kapiteln wird ein generalisiertes Prozessmodell beschrieben, welches die verallgemeinerbaren Strukturen in OSS-Entwicklungsprojekten formal beschreibt. Hierzu wurden generalisierbare Prozesse, Teilprozesse und Aktivitäten und deren relevante Entitäten wie die involvierten Rollen, Artefakte und Werkzeuge identifiziert und basierend auf dem erstellten Metamodell in verschiedenen Sichten modelliert, wobei alle dargestellten Modelle der Meta-Modellebene (vgl. *1.2.2.2 Metamodellierung*) zuzuordnen sind. Einführend werden in diesem Kapitel einige grundlegende Charakteristika des OSSD-Modells vorgestellt und ein gesamtheitlicher Überblick über das identifizierte Prozessmodell entwickelt, während die nachfolgenden Kapitel detaillierte Darstellungen der Prozesse und Prozessentitäten enthalten.

Ausgangspunkt für die Entwicklung eines allgemeinen Prozessmodells im OSS-Kontext ist die fallstudienbasierte Analyse und informelle Dokumentation der Entwicklungsprozesse verschiedener OSSD-Projekte in [Dietze03]. Diese Prozessanalysen wurden auf einer vergleichbaren Detaillierungstiefe mit Hilfe des dargestellten Metamodells durchgeführt und beschreiben die im Metamodell definierten Entitäten und Prozesse durch verbale und semiformale Beschreibungsmethoden. Basierend auf diesen analysierten Prozessen und Strukturen wurden anschließend die allen Fallstudien gemeinsamen Prozesse und Entitäten identifiziert und die generalisierbaren Aspekte anhand des Metamodells modelliert und formalisiert. Die im folgenden dargestellten Strukturen und Prozesse beschreiben somit generalisierbare und meist idealtypische Prozesse eines OSS-Entwicklungsprojekts, die in realen OSS-Projekten oft nur in angepasster Form praktiziert werden, aber auf der dargestellten Abstraktionsebene i.d.R. als verallgemeinerbar betrachtet werden können.

3.1 Distinktive Charakteristika des OSSD-Modells

In diesem Abschnitt werden verschiedene allgemeine Charakteristika und Prämissen der Systementwicklung im Open Source Kontext definiert, welche über alle analysierten Projekte hinweg verallgemeinert werden konnten und von Bedeutung für den weiteren Entwicklungsprozess sind.

3.1.1 Initiale Prototypentwicklung vs. sukzessiver Verbesserungsprozess

Von großer Bedeutung ist die Abgrenzung der initialen Prototypentwicklung und –veröffentlichung durch den Initiator des Projekts vom weiteren sukzessiven Softwareverbesserungsprozess im OSS-Kontext. Beide Prozesse basieren auf sehr verschiedenen Ansätzen und können strikt voneinander abgegrenzt werden. Diese Zweiteilung des Prozessmodells wird in *3.2.2 Phasen im Lebenszyklus eines OSS-Projekts* noch explizit dargestellt, wobei als repräsentativ für die eigentliche Systementwicklung im OSS-Kontext primär der sukzessive Verbesserungsprozess durch die Community der Projekte betrachtet wird, da dieser die originären Eigenschaften der OSS-Entwicklung repräsentiert.

3.1.2 Prozessparallelisierung und Prozessautonomie

[Weber00][50] beschreibt die Tendenz von OSS-Projekten zur konsequenten Parallelisierung aller Entwicklungsprozesse. Dies wird dadurch ermöglicht, dass die jeweiligen Entwicklungsaktivitäten i.d.R. vollständig autonom durch die einzelnen Akteure ausgeführt werden können und keinem gemeinsamen zeitlichen Kontext untergeordnet werden müssen, was in den folgenden Abschnitten noch detailliert dargestellt wird. [FeFi02][51] leitet aus dem hohen Grad der Parallelisierung der Aktivitäten zudem die Notwendigkeit und auch die immanente Tendenz zu einer *modularen* (vgl. [HoRe02]) Architektur der entsprechenden OSS ab.

Brooks' Law vs. Linus' Law

Die Möglichkeit zur parallelen Entwicklung ermöglicht überhaupt erst die verteilte Systementwicklung durch eine sehr große und heterogene Projektcommunity [FeFi02][52]. Brooks' Law[53] lässt sich dementsprechend nur auf proprietäre Softwareprojekte anwenden, in denen die Integration neuer Entwickler zu jedem Zeitpunkt immer auch einen Mehraufwand an koordinativen Aufgaben nach sich zieht, da die Einstiegsbarrieren vergleichsweise hoch sind und eine Vielzahl von Wechselwirkungen zwischen den einzelnen Prozessen existieren. Dies trifft auf die Software Entwicklung im OSS-Kontext nicht im selben Maße zu [Grassmuck02][54]. Dadurch erlangt Linus' Law im OSS-Kontext Gültigkeit, welches besagt, dass proportional zur Anzahl der Akteure in einem OSS-Projekt die Qualität der jeweiligen OSS ansteigt [FeFi02][55].

3.1.3 Dezidierte Quellcodepfade

Ein weiteres wichtiges Merkmal der Softwareentwicklung unter dem Paradigma von OS ist die parallele Verwaltung zweier Entwicklungslinien des Quellcodes in Form von eigenständigen Quellcodeverzeichnissen und die darauf basierende Generierung von dedizierten Software-Releases. I.d.R. werden Quellcodeverzeichnisse für die Verwaltung sowohl eines stabilen und getesteten Quellcodepfades, als auch eines Entwicklungsquellcodepfades generiert. Die Notwendigkeit zur Etablierung dieser beiden Quellcodelinien resultiert aus einem Interessenskonflikt der verschiedenen Stakeholder an dem verwalteten Quellcode. Während die reinen Anwender des Systems stets über stabilen Quellcode (*stable* oder *production*) verfügen möchten, sind die Entwickler der Software daran interessiert, möglichst direkt auf den aktuellsten Quellcode (*development* bzw. *experimental*) zugreifen zu können und entwickelte Patches sofort in den Quellcode zu integrieren. Zudem stellt es eine elementare Voraussetzung für den Erfolg eines OSS-Projekts dar, Entwicklern Zugriff auf den aktuellsten Quellcode zu ermöglichen und ihnen die Möglichkeit zur direkten Integration ihrer Patches in das Gesamtsystem zur Verfügung zu stellen.

Der stabile Pfad dient zur Generierung von stabilen Software-Releases, während der im Entwicklungspfad verwaltete Quellcode zur Veröffentlichung von Software-Releases zur Unterstützung fühzeitiger Software-Tests verwendet werden kann, welche in den entsprechenden Artefaktbeschreibungen in [Dietze03c] explizit dargestellt werden.

---

[50] „If there is an important problem in the project, a significant bug, or a feature that has become widely desired, many different people or perhaps teams of people will be working on it -- in many different places, at the same time." [Weber00]

[51] „Parallel, rather than linear development is a key characteristic of the OSS process, and is enabled by (and also perpetuates) the highly modular nature of many OSS products." [FeFi02]

[52] „Specifically, parallel development makes it possible for a very large number of developers to collaborate efficiently - since they are working simultaneously rather than waiting on each other." [FeFi02]

[53] „Adding manpower to a late software makes it later." [Brooks95]

[54] „Das Debugging kann hochgradig parallelisiert werden, ohne dass der positive Effekt durch erhöhten Koordinationsaufwand und steigende Komplexität konterkariert würde." [Grassmuck02]

[55] „[...] OSS appears to reverse Brook's Law [...], and the OSS community has proposed their own law, Linus' Law (i.e. given enough eyeballs, all bugs are shallow)." [FeFi02]

3.1.4 Webbasierte, asynchrone Kommunikationskanäle

Die effektive Kommunikation zwischen den dezentral organisierten Anwendern des Systems ist als elementare Voraussetzung für die Durchführung der Softwareentwicklungsprozesse im OS-Kontext zu betrachten. Daher wird in allen analysierten Projekten versucht, dedizierte Kommunikationskanäle durch das Management des Projekts bereitzustellen, welche die zielgerichtete Kommunikation mit verschiedenen Akteuren ermöglichen und helfen, die diffuse Informationsflut, welche im Rahmen eines Softwareentwicklungsprojekts entsteht, entsprechend der Inhalte und Zielgruppen der Kommunikation zu organisieren. Die ständige Möglichkeit zur Interaktion der Akteure, welche durch aktive Partzipation von passiven Konsumenten zu aktiven Akteuren des Entwicklungsprozesses werden, kann als wichtige Bedingung für den Erfolg eines OSSD-Projekts angesehen werden.
Dementsprechend werden meist verschiedene asynchrone, web-basierte Kommunikationskanäle für die verschiedenen Anwendungsszenarien, Inhalte und Zielgruppen bereitgestellt, wie z.B. zur Adressierung aller Developer für die Diskussion entwicklungsrelevanter Fragen. Die Kommunikation über die verschiedenen Kommunikationswege wird außerdem i.d.R. durch entsprechende Guidelines reglementiert, welche den Prozess der Kommunikation beschreiben und die verschiedenen Kanäle und ihren Verwendungszweck definieren.

3.1.5 Community

Ein weiteres Merkmal des OSSD sind die verteilten und heterogenen Projekt-Communities. In diesem Abschnitt werden einige fundamentale Aussagen über die Community von OSS-Projekten getroffen, aus denen sich verschiedene Implikationen bzw. Anforderungen für das gesamte Prozessmodell und auch eine unterstützende Software-Infrastruktur ableiten lassen.

3.1.5.1 Motivationsfaktoren der Akteure

Die verteilten Akteure des OSS-Entwicklungsprozesses agieren ausschließlich auf freiwilliger Basis und daher aus einer Motivation heraus, die teilweise von dem klassischen, z.T. monetären Anreizsystem der kommerziellen Softwareentwicklung abweicht. In verschiedenen Studien ([Weber00], [Hertel02]) wurde dieser Aspekt des sozio-ökonomischen Systems einer OSS-Community bereits untersucht und dabei verschiedene Erkenntnisse hergeleitet. Die primäre Motivation eines OSS-Entwicklers wird durch die Möglichkeit zur Befriedigung eines persönlichen Bedürfnisses repräsentiert. Dies wird nach [Hertel02] ergänzt durch verschiedene sozial- oder belohnungsorientierte Motivationsfaktoren:

- Anerkennung
- Kommunikation
- Nutzen (Wissenserwerb, Bedürfnisbefriedigung)

Daher sind nach [Hertel02] verschiedene Faktoren ableitbar, welche die Motivation eines Akteurs beeinflussen:

- Nutzen/Aufwand-Verhältnis
- Erwartetes Feedback der Community
- Status der auszuführenden Aufgabe („*Valence*“)
- Subjektiv wahrgenommene Durchführbarkeit bzw. Wichtigkeit der Aufgabe („*Self Efficiancy*“ bzw. „*Instrumentality*“)
- Vertrauen in die Community

Nach [Weber00] ist es für einen Akteur wichtig, dass die von ihm durchgeführte Aktivität wahrgenommen wird und ergebnisorientiert ist, die Aufgabe als wertvoll betrachtet wird oder durch die Ausführung zusätzliches Wissen

erworben werden kann. Die zu erwartende Reputation in der Community stellt ebenfalls eine wichtige Einflusskomponente dar [Weber00][56].

Diese Faktoren sind im Motivationssystem der kommerziellen Softwareentwicklung natürlich auch enthalten, werden aber durch ein monetäres Leistungsvergütungsmodell ergänzt und erhalten dadurch einen geringeren Stellenwert. Da sie in der OSS-Entwicklung die ausschließlichen Motivationsfaktoren darstellen, ist ihre besondere Berücksichtigung notwendig.

3.1.5.2 Core Developer

[Pavlicek00][57] definiert die Hypothese, dass jedes OSS-Projekt ein Team von Entwicklern herausbildet, welches defacto für die überwiegende Mehrheit der Quellcodemodifikationen verantwortlich zeichnet. Diese These konnte im Rahmen der Studien bestätigt werden (vgl. [Dietze03]), in denen ein Kern von Entwicklern identifiziert wird, die für einen Großteil der Quellcodeänderungen verantwortlich sind. Dies wird desweiteren auch durch verschiedene andere Studien (vgl. [MoFiHe00], [WaAb03][58]) gestützt.
Außerdem wird vermutet, dass dieser Entwicklerkern lediglich im Kontext der Implementation von Quellcode existiert und die Durchführung sonstiger Aktivitäten, wie zum Beispiel die Erstellung von Change Requests (vgl. *5.1.1 Generierung von Change Requests – Kollaborative Anforderungsdefinition*), wesentlich gleichmäßiger über die gesamte Community verteilt ist (vgl. [MoFiHe00]).

Zudem konnten in allen Projekten auch verschiedene Rollen identifiziert werden, welche der zentralen Managementinstanz nahestehen, über dezidierte Privilegien und Verantwortlichkeiten verfügen und typischerweise diesem Kreis der Core Developer zugeordnet werden können. Diese umfassen z.B. die Committer und den Release Manager und werden im Abschnitt *4.1 Rollen* eingehender dargestellt.

3.1.5.3 Code Ownership

Der Aspekt der Code Ownership wurde nicht näher untersucht, wobei aber bereits einige Tendenzen identifiziert werden konnten. So konnten z.B. im Mozilla-Projekt die sogenannten Modul- und Component Owner identifiziert werden, welche die primären Ansprechpartner und Verantwortlichkeiten für bestimmte Quellcodesegmente repräsentieren, aber nicht notwendigerweise auch für den Großteil der Quellcodemodifikationen verantwortlich zeichnen. [MoFiHe00][59] untersucht diesen Aspekt detailliert im Kontext des Apache HTTPD-Projekts und kommt zu dem Ergebnis, dass zwar keine direkte Modul Ownership, aber trotzdem ein Entwicklerkern der bereits darsgestellten Core Developer existiert, der für die Mehrheit an Quellcodeänderungen in allen Modulen verantwortlich zeichnet und somit auch eine schwache Form der Code Ownership darstellt.

Außerdem definiert [MoFiHe00][60] die Hypothese, dass für ein Projekt dessen Umfang so groß ist, dass eine Entwicklergruppe von typischerweise 10-15 Personen (Core Developer) nicht für 80% des Quellcodes zuständig sein kann, eine strikte Code Ownership Policy definiert werden muss, um die Verantwortung auf verschiedene Akteure zu verteilen. Auf einem höheren Strukturierungsniveau als dem der Module bzw. Components im Mozilla-Projekt konnte dies z.B. auch im Rahmen der Linux Kernel Entwicklung identifiziert werden, da hier die

[56] „An open source process will work more effectively when […] contributors value status and reputation at least in part as symbolic rewards, in addition to the instrumental (ie monetizable) aspects." [Weber00]

[57] „The average project has a core team of a few central developers." [Pavlicek00]

[58] „Even though hundreds of volunteers may be participating in the OSS development projects, typically there is only a small group of developers performing the main part of the work." [WaAb03]

[59] „[...] rather than any single individual writing all the code for a given module, those in the core group have a sufficient level of mutual trust that they contribute code to various modules as needed." [MoFiHe00]

[60] „For projects that are so large that 10-15 developers cannot write 80% of the code in a reasonable time frame, a strict code ownership policy will have to be adopted to separate the work of additional groups, [...]." [MoFiHe00]

Verantwortung bzw. Code Ownership für verschiedene Versionen des Kernels auf verschiedene Akteure verteilt wurde.

Die Definition einer Code Ownership kann auch als Multiplikator für die Tendenz zur Aufspaltung in Teilprojekte wirken, da dadurch der Verantwortungsbereich der bereits dargestellten Core Developer auf einen klar abgrenzbaren Bereich eingeschränkt wird, für den die Owner des Teilprojekts die höchste Verantwortlichkeitshierarchie darstellen. Dies konnte z.B im gesamten Apache- und dem Mozilla-Projekt identifiziert werden, in denen eine Vielzahl von Teilprojekten gebildet wurden (vgl. [Dietze03]).

3.1.5.4 Maintenance Rollen

Die Maintenance der Projekte umfasst, wie bereits dargestellt wurde, die Managementprozesse zur Koordination und Steuerung der Prozesse und der Community und die infrastrukturellen Prozesse zur Etablierung und Verwaltung einer adäquaten Infrastruktur, welche die Entwicklungsprozesse unterstützt. Diese Aufgaben werden entweder durch einen einzigen Maintainer oder durch einen klar abgegrenzten Personenkreis (Komitee) ausgeführt, was eine sehr häufige Variante darstellt. Außerdem ist bekannt, dass die Rolle des Maintainers im Rahmen einiger Projekte zwischen verschiedenen Akteuren des Entwicklerkerns rotiert und keiner Person fix zugewiesen ist [Weber00][61]. Häufig werden im Verlauf eines Projekts Aufgaben durch den Maintainer zunehmend an andere Akteure delegiert, was direkt die Bildung der bereits dargestellten Core Developer begünstigt und von einer autoritären Maintenance durch einen einzelnen Akteur zu einer Maintenance durch ein Komitee führt.

Alle Aufgaben der Projektkoordination und -organisation werden durch diese Maintenance-Akteure ausgeführt oder aber zumindest gesteuert und koordiniert, wobei diese zentralen Instanzen durch die bereits dargestellten Core Developer unterstützt werden, welche in diesem Kontext ebenfalls eine sehr wichtige Rolle darstellen. Die strikte Abgrenzung zwischen den Maintenance-Akteuren und der Gruppe der Core Developer ist nicht immer eindeutig durchführbar und vor allem im Falle der Maintenance durch ein Komitee nur schwer möglich.

In allen untersuchten Projekten repräsentierten die Initiatoren eines Projekts zumindest für einen gewissen Zeitraum auch die Maintainer eines Projekts und üben die Maintenance-Aufgaben dadurch historisch bedingt aus (vgl. [FoBa02][62]). Die Maintainer und Core Developer eines Projekts üben in allen Projekten einen erheblichen Einfluss auf die Evolution des Projekts und der Software aus und sind stark an der langfristigen Orientierung und Entwicklung des Projekts beteiligt (vgl. [ArGaLa00][63]).

3.2 Gesamtheitlicher Überblick über das Prozessmodell

Im Mittelpunkt des OSS-Ansatzes stehen die kollaborativen Entwicklungsprozesse der Community des OSS-Projekts, welche den eigentlichen Prozess der sukzessiven Softwareverbesserung repräsentieren und durch die unterstützenden Management- und Infrastrukturprozesse ergänzt werden. Die folgende Grafik visualisiert diese Kernprozesse und die identifizierten Rollen auf der obersten Abstraktionsebene in der Anwendungsfallsicht:

[61] „Perl relies on a 'rotating dictatorship' where control of the core software is passed from one member to another inside an inner circle of key developers." [Weber00]

[62] „In fast allen Fällen ist der ursprüngliche Autor der Software, der sie veröffentlicht hat, der Betreuer der Software." [FoBa02]

[63] „Even in *‚shared-leadership'* situations, such as the Apache web server, investigations have established that the coredevelopers still exercise the major influence over the design and direction of OSS development." [ArGaLa00]

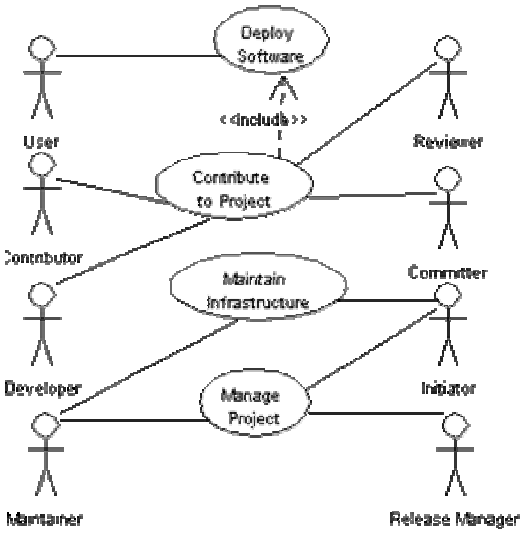

Abb. 3.1: Anwendungsfallsicht der Kernprozesse des OSS-Modells

Die kollaborativen Entwicklungsprozesse werden repräsentiert durch den Use Case *Contribute to Project* und werden dezentral durch die verteilte Community ausgeführt, während die infrastrukturellen und die Managementprozesse durch die zentralistischer organisierten Akteure (Maintainer, Core Developer) ausgeführt werden. Die verschiedenen Rollen und Verantwortlichkeiten werden Kapitel *4.1 Rollen* noch detailliert dargestellt.

3.2.1 Kernprozesse des OSSD-Modells

Die folgende Grafik visualisiert die Kernprozesse des identifizierten Prozessmodells in einer modifizierten Aktivitätssicht des UML-Metamodells:

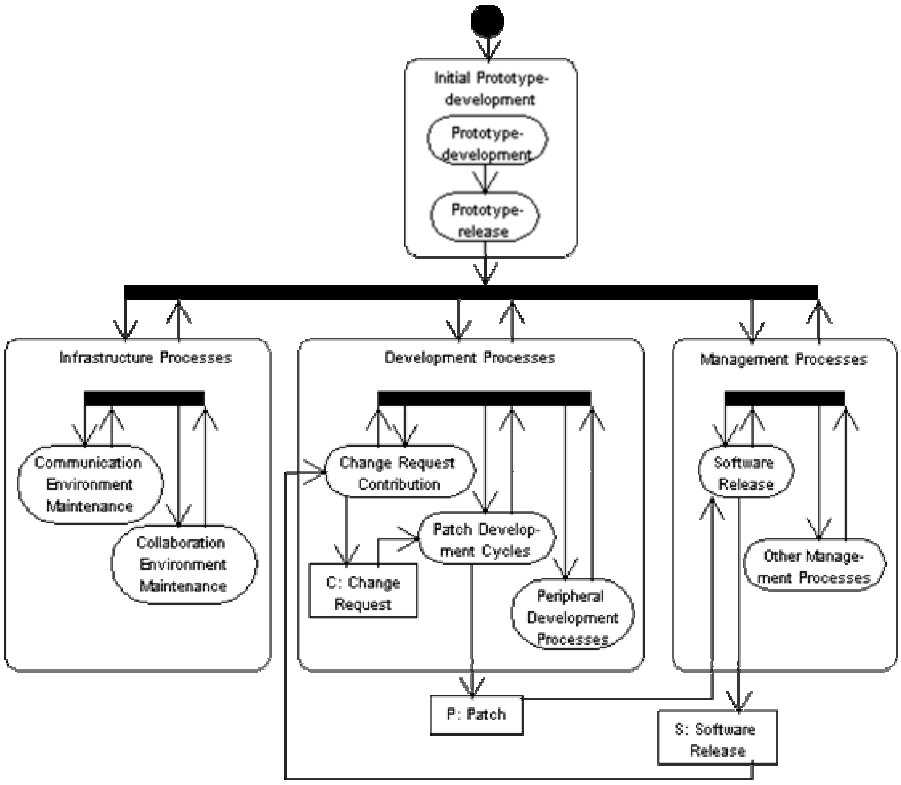

Abb. 3.2: Aktivitätssicht auf die Kernprozesse eines OSS-Projekts

Die hier dargestellten Prozesse werden in den folgenden Kapiteln detailliert dargestellt und anhand der Modellierungsvorgaben des Metamodells mit Modellelementen der UML spezifiziert.

3.2.1.1 Initiale Prototypentwicklung und -veröffentlichung

Den Ausgangspunkt für den OSS-typischen Prozess der sukzessiven Systemverbesserung definiert die initiale Entwicklung und Veröffentlichung eines Prototyps (*Initial Prototypedevelopment*) der Software, die von ersten infrastrukturellen und Managementprozessen begleitet werden. Dies umfasst z.B. die Bereitstellung einer ersten, meist rudimentären Projektinfrastruktur (z.B. Website, Mailing Liste) und die Bekanntgabe der OSS-Veröffentlichung in einschlägigen Medien zur Etablierung einer Community.

3.2.1.2 Parallelität und Autonomie der Prozesse als zentrales Distinktionsmerkmal

In den anschließenden Prozessen der sukzessiven Systemverbesserung, also dem OSS-typischen Entwicklungsprozess, werden die einzelnen Prozesse stark parallelisiert, kontinuierlich und weitgehend autonom ausgeführt. Dies betrifft auf der obersten Abstraktionsebene die Managementprozesse, die infrastrukturellen Prozesse und die Entwicklungsprozesse, welche im Mittelpunkt des gesamten Modells stehen. Im Rahmen der Entwicklungsprozesse wurden die Aktivitäten zur Anforderungsdefinition (*Change Request Contribution*) und zur Patchentwicklung (*Patch Development Cycles*) als eigenständige Prozesse dargestellt, da sie die Kernprozesse eines OSS-Entwicklungsprojekts darstellen. Auch sie werden prinzipiell autonom und parallel zueinander ausgeführt.

Die einzigen identifizierten Abhängigkeiten zwischen diesen Prozessen werden durch die Objektflüsse zwischen den einzelnen Prozessen repräsentiert. So werden zwar spezifische Aktivititäten dieser Prozesse absolut unabhängig voneinander ausgeführt, aber die dabei entstehenden Artefakte repräsentieren zum Teil die

Inputobjekte für die weiteren Prozesse. Beispielsweise werden einzelne Change Requests komplett unabhängig voneinander und auch von ihrer späterer Implementierung entwickelt, bilden aber als Artefakte die Inputobjekte für darauf basierende Patchentwicklungsprozesse. Gleiches gilt für die individuell entwickelten Patches, welche in ihrer Gesamtheit die Basis für einen neuen Software-Release-Prozess darstellen.

3.2.1.3 Kollaborative Anforderungsdefinition - *Change Request Contribution*

Die Prozesse der kollaborativen Anforderungsdefinition durch die gesamte Community finden in Form von individuellen Aktivitäten zur Definition von sogenannten Change Requests statt (vgl. *4.3.1 Change Request*) und werden permanent und stark parallelisiert durch die Akteure der Community ausgeführt.

Die grundlegende Ausrichtung und Funktionalität der Software wurde bereits im Rahmen der Anforderungsdefinition während der initialen Prototypentwicklungsprozesse definiert. Dem schließt sich der für das OSS-Modell charakteristische Prozess der kollaborativen und parallelisierten Erstellung von Change Requests durch die dezentralen und unabhängigen Akteure an, der auch als kollaborative Anforderungsdefinition bezeichnet werden kann. Im Gegensatz dazu ist die individuelle Anforderungsdefinition direkter Bestandteil eines auf einem spezifischen Change Request aufbauenden und durch den jeweiligen Entwickler durchzuführenden, expliziten Patchentwicklungszyklus, während die kollaborative Anforderungsdefinition und der Review der Anforderungen, als eigenständige und unabhängig von Entwicklungsaktivitäten ausführbare Aktivitäten betrachtet werden können.

Im Rahmen der kollaborativen Anforderungsdefinition werden zum Teil nicht nur die operationalen Anforderungen für individuelle Patchentwicklungszyklen erfasst, sondern auch strategische Anforderungen bzw. Ausrichtungen des Projekts bzw. des entwickelten Softwareprodukts definiert, welche die Rahmenbedingungen für die operationale Anforderungsdefinition darstellen. Diese strategischer orientierten Anforderungen werden i.d.R. durch die Management-Akteure in Form spezieller Dokumente oder Artefakte definiert und sind im OS-Kontext von eher sekundärer Bedeutung, da sie nicht notwendigerweise zu Konsequenzen für die Patchentwicklung der Community führen müssen und daher eher als unverbindliches, steuerndes Element im OSS-Kontext betrachtet werden können, was nicht in allen OSS-Projekten Verwendung findet.

3.2.1.4 Patchentwicklungszyklen - *Patch Development Cycles*

Jegliche Quellcodemodifikation wird in Form eines Patches erstellt und verteilt. Die Entwicklungsaktivitäten zur Erstellung eines Patches im Rahmen eines spezifischen Patchentwicklungszyklus basieren i.d.R. auf genau einem der kollaborativ entwickelten Change Requests. Der Prozess der Selektion eines der Change Request kann wie bereits erwähnt auch als individuelle Anforderungsdefinition beschrieben werden.
Die Prozesse der einzelnen Patchentwicklungszyklen ähneln dabei stark dem Modell des Prototyping (vgl. [NoSt98]) wobei die Aktivität des Review des Patches durch die gesamte Community bzw. dedizierte Reviewer ausgeführt wird. Die Quellcodemodifikation wird in Form eines Patches an die Community kommuniziert und anschließend in mehreren Iterationen verbessert, bis alle involvierten Akteure einen Konsens über dessen Qualität erreicht haben.

3.2.1.5 Management- und infrastrukturelle Prozesse

Die beschriebenen Entwicklungsprozesse werden durch verschiedene Management- und Infrastrukturprozesse begleitet, welche den Prozess der sukzessiven Systemverbesserung ermöglichen und unterstützen. Diese Prozesse werden primär durch die Maintainer bzw. die Core Developer eines OSS-Projekts durchgeführt und sind nicht explizit einzelnen Phasen oder Prozessen des Prozessmodells zuzuordnen, sondern werden übergreifend über alle Entwicklungsprozesse hinweg und weitgehend unabhängig von diesen ausgeführt. Die Managementprozesse und die infrastrukturellen Prozesse werden weitgehend autonom und parallel zueinander und zu den Entwicklungsprozessen ausgeführt. Der in *5.2.1 Software-Release* explizit dargestellte Prozess der Erstellung eines Software-Releases wird per definition als Managementprozess dargestellt, da er durch einen zentralen Akteur, respektive den Release Manager, ausgeführt wird und die dezentralen Entwicklungsprozesse unterstützen bzw. ermöglichen soll (vgl. *5.2.1 Software-Release*).

3.2.2 Phasen im Lebenszyklus eines OSS-Projekts

Der gesamtheitliche Lebenszyklus eines OSS-Projekts umfasst die grundlegende Unterscheidung verschiedener Entwicklungsphasen, wobei besonders die Abgrenzung des Prozesses der initialen Prototypentwicklung und des initialen Prototyprelease von den Prozessen der schrittweisen Softwareverbesserung durch eine Projekt-Community von Bedeutung ist. Diese beiden Abschnitte des gesamtheitlichen Prozessmodells repräsentieren die wesentlichen, in einer gesamtheitlichen Prozessdarstellung als zeitlich aufeinander folgende und klar voneinander abgrenzbare Phasen des gesamten OSS-Ansatzes.

Identifizierbare Phasen
Die folgenden Phasen wurden dabei identifiziert:

- Initialisierung (*Initialisation*)
- Sukzessive Softwareverbesserung (*Gradual Software Improvement*)
  - Etablierung des Projekts (*Establishment*)
  - Kontinuierliche Entwicklung (*Advancement*)
  - Projektende (*Discontinuation*)

Die initialen Entwicklungs- und Veröffentlichungsprozesse *(Initialisation)* finden i.d.R. in einem geschlossenen System statt und werden durch den Initiator bzw. die Initiatoren eines Projekts ausgeführt, während der charakteristische Entwicklungsprozess im Open Source Kontext auf dem in diesem Rahmen veröffentlichten Protoyp basiert und auch als sukzessiver Softwareverbesserungsprozess (*Gradual Software Improvement*) bezeichnet werden kann. Häufig findet im Rahmen des initialen Release einer Software bzw. im Rahmen der ersten Iterationen des sukzessiven Softwareverbesserungsprozesses eine Phase des *Re-Engineering* der Software statt, welche dazu dient, die Software, respektive die Systemarchitektur, den Anforderungen des sukzessiven Softwareverbesserungsprozesses anzupassen.

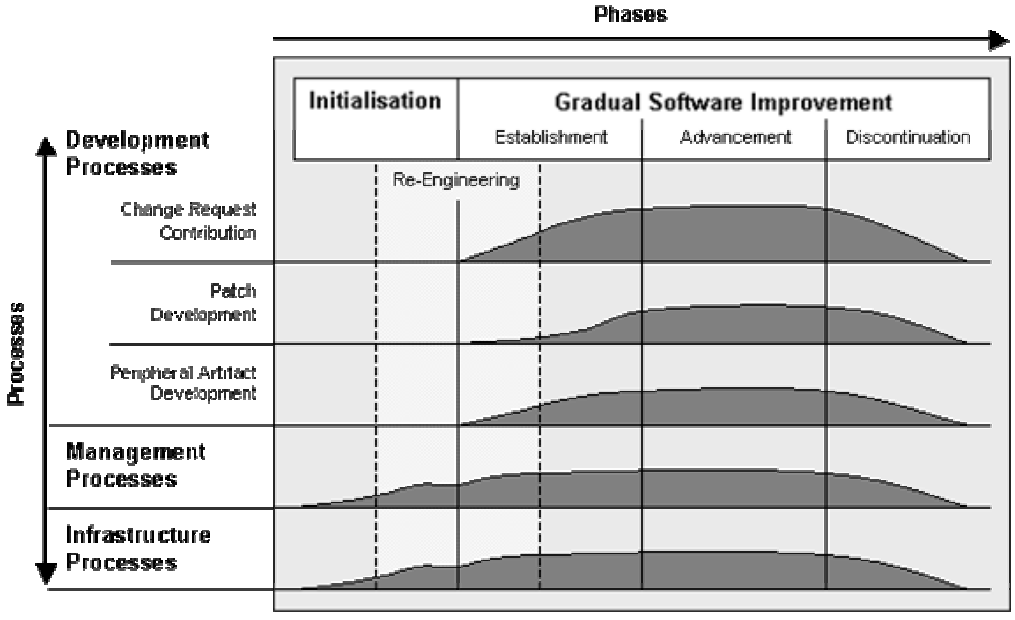

Abb. 3.3: Lebenszyklus eines OSSD-Projekts (Darstellungsweise angelehnt an [JaBoRu99])

In *Abb. 3.3* wurde in Anlehnung an die Darstellungsweise des Rational Unified Process (RUP, vgl. [Kruchten00]) bzw. den USDP (vgl. [JaBoRu99]) der typische Lebenszyklus eines OSSD-Projekts dargestellt und dabei die identifizierten Phasen entlang einer horizontalen Achse und die bedeutendsten OSSD-Prozesse analog zu den

sogenannten *Disziplinen* in [KrKr03] entlang einer vertikalen Achse dargestellt. Aus Gründen der Komplexitätsreduktion wurden lediglich die elementaren Kernprozesse der Entwicklungsprozesse (*Change Request Contribution, Patch Development*) in die Darstellung aufgenommen. Die dargestellten Kurvenverläufe innerhalb der Matrix visualisieren den Grad der Intensivität mit der die einzelnen Prozesse im Rahmen der jeweiligen Phase typischerweise ausgeübt werden. Dadurch grenzt sich die Darstellungsform von der Darstellung in [Kruchten00] und [JaBoRu99] ab, da hier die Kurvenverläufe die Anzahl der durchgeführten Iterationen visualisieren. Die einzelnen Kurvenverläufe wurden dabei in *Abb. 3.3* stark vereinfacht dargestellt und unterliegen in der Realität natürlich erheblichen Schwankungen.

3.2.2.1 Initialisierung

Die initiale Entwicklung eines Prototyps wird i.d.R. durch den Initiator des Projekts und in einem geschlossenen Entwicklungsprozess durchgeführt. Dieser Prozess unterliegt somit nicht den spezifischen Anforderungen der Softwareentwicklung durch eine OSS-Community und wird daher im Rahmen dieser Arbeit auch nicht eingehend analysiert. Für die spätere OSS-Entwicklung sind in dieser Phase primär die mit der Veröffentlichung des Prototyps verbundenen Managementprozesse, wie z.B. die Entwicklung eines geeigneten Lizenzmodells oder die Etablierung einer initialen Infrastruktur von Bedeutung. Daher werden diese, durch den Initiator ausgeführten Aktivitäten im Rahmen der Managementprozesse (Kapitel *5.2 Managementprozesse*) näher erläutert. Zudem besteht aufgrund der Geschlossenheit dieses Prozesses und dessen oft proprietären Charakters nur geringe Transparenz über diesen Aspekt eines Open Source Software Projekts. Häufig repräsentieren die Initiatoren im weiteren Verlauf des sukzessiven Softwareverbesserungsprozesses auch die Akteure der späteren Maintenance des Projekts und sind somit auch für die Durchführung der in diesem Rahmen praktizierten Managementprozesse zuständig. In einigen Projekten wurden die initialen Entwicklungsprozesse auch in einem kommerziellen und proprietären Kontext durch ein profitorientiertes Unternehmen durchgeführt, wobei erst im weiteren Projektverlauf der Quellcode des entwickelten Softwareprodukts offengelegt und der Entwicklung im Open Source Kontext zugeführt wurde (vgl. [Dietze03]). Dies repräsentiert eine durchaus typische Vorgehensweise im Rahmen der initialen Systementwicklung im OSS-Kontext (vgl. [GoTu00][64]).

Re-Engineering

Da die initiale Prototypentwicklung nicht notwendigerweise von Beginn an auf die Weiterentwicklung der Software nach OS- Prinzipien ausgerichtet ist, genügt die entwickelte Software häufig nicht den Anforderungen des heterogenen und kollaborativen OSS-Entwicklungsprozesses. Dies kann zu einer Phase des Re-Engineerings der Software führen (vgl. [Dietze03]), um den Anforderungen eines dezentralen Entwicklungsprozesses zu entsprechen. Primäre Zielsetzung ist es hierbei, den Quellcode einem Refactoring zu unterziehen, um die Lesbarkeit für eine sehr heterogene Entwicklergemeinde zu verbessern, die Softwarearchitektur weitgehend zu modularisieren (vgl. [HoRe02]) und ein den klassischen Kriterien der Softwarequalität entsprechendes Softwardesign zu realisieren, um die verteilte, kollaborative Weiterentwicklung des Systems und einzelner Komponenten zu ermöglichen. Die so realisierten Softwaremodule sollten im Idealfall den Anforderungen einer minimalen Kopplung und einer maximalen Kohäsion genügen, verfügen über abstrakte Schnittstellen und basieren auf weit verbreiteten und möglichst frei verfügbaren Standards, um die Komplexität weitestgehend zu minimieren und eine größtmögliche Zielgruppe erreichen zu können. Dies beinhaltet u.a. den vollständigen Verzicht auf jegliche proprietäre Quellcodeelemente oder Softwarebibliotheken.

Diese Phase kann entweder im Rahmen der initialen Entwicklungsprozesse, also in einem geschlossenen Kontext, oder nach der OSS-Veröffentlichung und somit unterstützt durch die entstehende Community ausgeführt werden (vgl. *Abb. 3.3*). Häufig wird diese Phase als implizites Element sowohl der Projektinitialisierung als auch des graduellen Softwareverbesserungsprozesses ausgeführt.

[64] „Many corporations and individuals have developed source code in-house as a proprietary project only to release later it as ‘open source’ or with a license that allows great freedom for personal use of the system.” [GoTu00]

3.2.2.2 Prozess der sukzessiven Systemverbesserung

Der charakteristische OSS-Entwicklungsprozess kann in Anlehnung an [Tuomi01] auch als Prozess der sukzessiven Systemverbesserung durch die Community betrachtet werden und beinhaltet als Hauptbestandteil die kontinuierlich und parallel durchgeführten (Patch-)Entwicklungsprozesse durch die Akteure der Community und alle unterstützenden Aktivitäten.

Der initiale Prototyp wird durch die kontinuierliche Entwicklung von Patches um Softwarefehler bereinigt und um zusätzliche Funktionalitäten und Systemeigenschaften bereichert. [Godfrey00] teilt die Aktivitäten im Rahmen der sukzessiven Softwareverbesserung in die folgenden Kategorien ein, die auch im Rahmen der Fallstudien bestätigt werden konnten:

- Adaptive Aktivitäten: Hinzufügen neuer Funktionalitäten, Unterstützung weiterer Plattformen
- Korrektive Aktivitäten: Beheben von Softwarefehlern oder nicht korrekt implementierten Anforderungen
- Perfektionierende Aktivitäten: Performance Tuning
- Präventive Aktivitäten: Refactoring, Restrukturierung des Quellcodes und der Architektur, Schnittstellenimplementierung

Etablierung des Projekts
Die Phase der Etablierung des Projekts ist idealtypischerweise von einem stetigen Wachstum der Projekt-Community geprägt, deren verteilte Akteure mit wachsender Intensität die typischen Entwicklungsprozesse eines OSS-Projekts ausführen. Dies geht einher mit einer steten Zunahme der unterstützenden Management- und infrastrukturellen Prozesse, die zur Koordination der Entwicklungsprozesse und zur Bereitstellung und Wartung einer geeigneten Entwicklungsumgebung notwendig sind.

Kontinuierliche Verbesserung der OSS
Typischerweise schließt sich an eine Phase des stetigen Wachstums der Community eines Projekts eine Phase der Stagnation auf einem vergleichsweise stabilen Niveau an. Idealerweise konnte zu diesem Zeitpunkt bereits eine hinreichend große Anzahl von aktiven Community-Mitgliedern erreicht werden, die eine Weiterentwicklung der Software über einen möglichst großen Zeitraum hinweg sicherstellen.

Projektende
Die Entwicklungsprozesse in OSS-Projekten finden kontinuierlich und ohne ein klar definiertes Projektende statt [WaAb03][65]. Auch in allen analysierten Projekten konnte kein Projektende identifiziert oder prognostiziert werden (vgl. [Dietze03]). Vielmehr wird das Ende eines OSS-Projektes i.d.R. nicht von einem finalen Abschluss gekennzeichnet, sondern zeichnet sich durch eine stetig abnehmende Anzahl der Iterationen des sukzessiven Softwareverbesserungsprozesses und eine ständig abnehmende Zahl von aktiven Beiträgen zu den Entwicklungsprozessen durch die Akteure aus. Dies führt dazu, dass der Fortschritt in der Weiterentwicklung der OSS stetig abnimmt und diese somit auch auf abnehmende Verbreitung und Akzeptanz trifft. Eine weitere Ursache für die Beendigung eines OSS-Projekts kann die Einstellung der Maintenance des Projekts durch den oder die Maintainer darstellen, sofern kein neuer Akteur diese Rolle übernimmt.

Da sich in den einzelnen Phasen lediglich der Grad der Intensität, nicht aber die Art der Ausführung der verschiedenen OSSD-Prozesse in den verschiedenen Phasen unterscheidet, wird im Rahmen der Prozessbeschreibung der folgenden Kapitel nicht weiter zwischen den einzelnen Phasen unterschieden.

[65] „The process is continuous, as software development is evolving [seemingly] endlessly." [WaAb03]

3.2.3 Distinktionsmerkmale proprietärer Software-Entwicklung

Die verteilte Softwareentwicklung in einem proprietären Kontext weist zwar zum Teil Merkmale auf, die denen des OSS-Modells z.T. gleichen, trotzdem grenzen die folgenden Distinktionsmerkmale den Ansatz der proprietären Softwareentwicklung von dem Ansatz des OSSD ab:

- Zentrale Planung und Management
- Direkte Abhängigkeiten zwischen Prozessen
- Kundenspezifische Anforderungsdefinition
- Planbarkeit der personellen Ressourcen
- Klares Projektende (Ziel, Zeitpunkt)

Zwar interagieren in einem proprietären verteilten Softwareprojekt häufig auch dezentral verteilte Entwickler im Rahmen eines kollaborativen Entwicklungsprojekts, wobei aber in diesem Fall eine zentrale Planung und ein zentrales Management existieren, wodurch diese Prozesse umfassend geplant, kontrolliert und gesteuert werden. Zwar finden auch in einem proprietären Kontext viele Iterationen und zum Teil auch ad-hoc-Entwicklungsprozesse statt, aber in Distinktion zum OSS-Modell existieren in proprietären und zentral gesteuerten Prozessen starke Abhängigkeiten zwischen den einzelnen Prozessen. Beispielsweise werden einmal definierte Anforderungen typischerweise auch immer als Quellcode implementiert, was im Kontext von OS nicht gewährleistet ist. Dies schließt natürlich nicht aus, dass einmal implementierte Anforderungen auch im proprietären Kontext im Rahmen einer iterativen Softwareverbesserung ständig überarbeitet werden.

Aus der zentralen Planung und der Ausrichtung der Entwicklungsprozesse an den Interessen eines spezifischen Kunden resultiert, dass die Vorgaben des geplanten Funktionsumfangs eines zu erstellenden Software-Release maßgeblich für jede Anforderung sind, die im Rahmen von proprietären Softwareprojekten definiert sind. Demgegenüber steht der freie und unreglementiert Prozess der Anforderungsdefinition im Rahmen des OSS-Modells, der die Definition von prinzipiell jeglicher Anforderung ermöglicht und absolut unabhängig von den Prozessen des Software-Release oder der Patch-Implementation ausgeführt wird.

Als weiteres Distinktionsmerkmal kann die freiwillige und nicht planbare Mitwirkung der Akteure in einem OSS-Projekt betrachtet werden. Demgegenüber sind die personellen Ressourcen in einem proprietären Projekt planbar und können je nach Erforderlichkeit erweitert werden. Direkt aus dieser Eigenschaft der OSS-Entwicklung resultiert auch der hohe Grad der Parallelisierung und der Autonomie der Aktivitäten im Rahmen von typischen OSS-Entwicklungsprojekten, bei denen die verschiedenen Akteure typischerweise unabhängig voneinander und autonom agieren und in parallelisierten Prozessen Artefakte beitragen oder modifizieren. Demgegenüber steht ein hoher Grad der Abhängigkeiten zwischen den Prozessen proprietärer und zentral gesteuerter Softwareentwicklungsprojekte.

Das Projektende unterscheidet sich bei beiden Ansätzen ebenfalls fundamental voneinander. Während ein proprietäres Projekt idealtypischerweise mit der Abnahme durch den Kunden (Individualsoftware) oder der Fertigstellung eines Softwareprodukts (Standardsoftware) endet, kann das Ende eines OSS-Projekts nicht genau definiert werden, wie bereits in *3.2.2.2 Prozess der sukzessiven Systemverbesserung* dargestellt wurde.

4 Identifizierte Entitäten: Rollen, Artefakte und Software-Werkzeuge

In diesem Kapitel werden alle Entitäten (Rollen, Artefakte und Werkzeuge) eingeführt, welche im deskriptiven und generalisierten Modell identifiziert wurden, wobei deren konkrete Darstellung in [Dietze03c] enthalten ist.

4.1 Rollen

In diesem Abschnitt werden die identifizierten Rollen dargestellt, welche zum Teil explizit durch die Maintainer eines Projekts definiert werden oder lediglich aufgrund der praktizierten Prozesse als implizite Rolle identifiziert wurden. Eine Rolle repräsentiert in dieser Arbeit die Aggregation von spezifischen Aktivitäten und Verantwortlichkeitsbereichen einer abgrenzbaren Menge von Akteuren und beschreibt deren generalisierbare Aufgabenbereiche und Charakteristika. Es wurden somit Teilmengen von Akteuren, die signifikante gemeinsame Eigenschaften aufweisen oder ähnliche Tätigkeiten ausführen, zu spezifischen Rollen zusammengefasst, die meist nicht explizit im Rahmen der Projekte definiert wurden.

Als wichtige Eigenschaft von Akteuren im OS-Kontext ist die Möglichkeit zur parallelen Zugehörigkeit zu mehreren Rollen zu betrachten. Häufig impliziert die Zugehörigkeit zu einer bestimmten Rolle sogar die Zugehörigkeit zu einer weiteren Rolle. Beispielsweise repräsentiert ein Entwickler im OSSD-Kontext (Rolle: Developer) zugleich auch einen Anwender bzw. Contributor. Eine derartig hierarchische Beziehung mehrerer Rollen kann somit als Vererbung betrachtet und modelliert werden, da die untergeordnete Rolle die mit ihr assoziierten Aufgabenbereiche und Privilegien an die übergeordnete Rolle vererbt und somit eine mehrstufige Vererbungshierarchie entsteht, die in *Abb. 4.1* modelliert wurde.

Neben diesen Vererbungsstrukturen, existiert auch die Möglichkeit, dass ein Akteur mehrere Rollen ausübt, die nicht direkt durch eine Vererbungshierarchie verbunden sind. So wird die Rolle des Release Managers z.B. häufig durch einen Akteur wahrgenommen, der parallel auch andere Management Rollen ausübt.
Die Aggregationen zu den im folgenden dargestellten Rollen ist z.T. noch weiter detaillierbar bzw. in spezifischere Rollen unterteilbar. Zur Reduktion der Komplexität wurde darauf im Rahmen dieses Modells verzichtet, um eine möglichst sinnvoll generalisierbare Abstraktionsebene zu erreichen.

Meritokratisches System
Alle analysierten OSSD-Projekte ermöglichen Akteuren den leistungsbezogenen Zugewinn von Status und Nutzerrechten und können daher als meritokratische, soziale Systeme betrachtet werden, in welchen der Status eines Akteurs direkt von seinen Beiträgen zum Projekt abhängig ist [Cubranic01][66].

Implizit identifizierte Rollen
Während der Durchführung der Fallstudien konnten die folgenden impliziten Rollen identifiziert werden:

- User
- Contributor
- Developer
- Reviewer
- Committer

[66] „Over time, contributors who have distinguished themselves by the quality and frequency of their work may be invited to join the core group and gain more responsibility in the project." [Cubranic01]

- Release Manager
- Maintainer
- Initiator

Diese Rollen werden i.d.R. nicht explizit in den verschiedenen Projekten definiert, sondern konnten lediglich implizit identifiziert werden. Ein detaillierte Erläuterungen der identifizierten Rollen erfolgt in [Dietze03c].

Generalisierte Rollenhierarchie
In der folgenden Grafik werden die identifizierten Rollen entsprechend dem Metamodell der Rollenkonzeptsicht (vgl. [Dietze03b]) dargestellt, wobei typische Aufgabenbereiche (Aktivitäten) als Operationen dargestellt wurden:

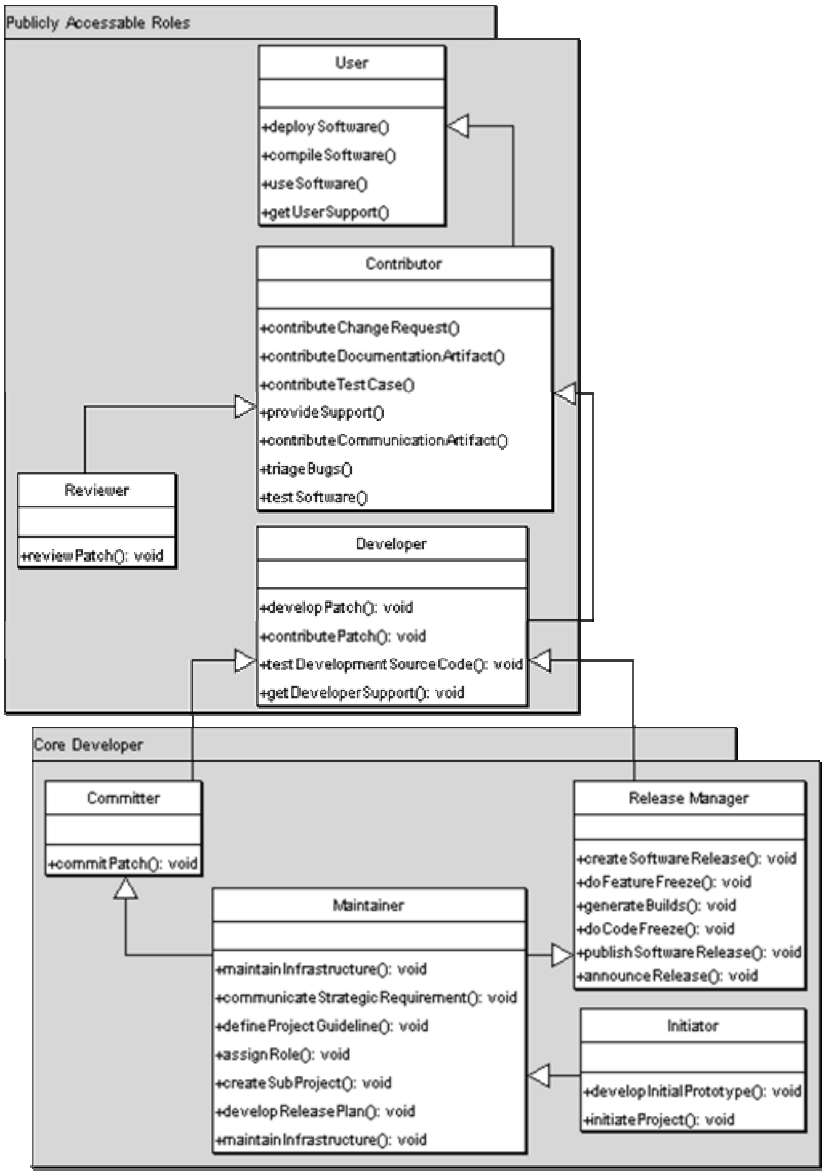

Abb. 4.1: Identifiziertes Rollenmodell

Die dargestellten Operationen der Rollenklassen repräsentieren typische Aktivitäten bzw. Prozesse, welche durch die jeweilige Klasse ausgeführt und in den konkreten Prozessdarstellungen noch eingehend dargestellt werden.

Frei zugängliche vs. dediziert ernannte Rollen
Ein wichtiger Aspekt ist die Unterscheidung frei zugänglicher Rollen von Rollen, die ausschließlich durch entsprechend privilegierte Akteure zugewiesen werden können. Die Rollen User, Contributor, Developer und auch die Rolle des implizit identifizierten Reviewers sind für jeden Akteur frei zugänglich und werden in einem inhärenten Prozess vergeben, sobald ein Akteur die typischen Aktivitäten der Rolle ausführt. Ein Akteur, der bespielsweise einen Change Request, also eine Änderungsanforderung des Quellcodes beiträgt, kann automatisch als Contributor betrachtet werden.

Die speziellen Rollen der Core Developer unterscheiden sich von diesen Rollen dahingehend, dass sie nicht frei zugänglich sind und somit nicht durch jeden interessierten Akteur eingenommen werden, sondern nur durch die Ernennung durch einen entsprechend privilegierten Akteur eingenommen werden könnn. Dies geschieht, indem beispielsweise einem Entwickler durch den Maintainer Commit-Privilegien im Quellcodeverwaltungssystem eingeräumt werden und dieser Akteur dadurch automatisch den Status eines Committers erhält. Auf diese Prozesse wird in den folgenden Kapiteln noch explizit eingegangen.

4.2 Identifizierte Software-Werkzeuge

Im Kontext der infrastrukturellen Prozesse werden verschiedene Werkzeuge instrumentalisiert, welche die Entwicklungsprozesse unterstützen und entsprechend dem definierten Metamodell in verschiedene Kategorien (Kollaborations-, Kommunikations-, Entwicklungswerkzeuge) eingeteilt werden können. I.d.R. sind alle zentral verwendeten Werkzeuge unter einer OSS-konformen Lizenz verfügbar und stellen damit eine Ressource dar, deren projektweite Verwendungsmöglichkeit über einen längeren Zeitraum Bestand hat. Zudem stehen diese Werkzeuge somit allen Mitgliedern der Community kostenlos zur Verfügung.

Die gemeinsam genutzten Ressourcen wie die Kollaborations- und Kommunikationsinfrastrukturen werden zentral durch das Management bereitgestellt und verwaltet, während die individuell genutzten Entwicklungswerkzeuge, welche ausschließlich die individuell durch die Entwickler ausgeführten Entwicklungsprozesse unterstützen, explizit durch jeden einzelnen Akteur instrumentalisiert werden. Außerdem wurden Werkzeuge, welche im Kontext von OSS-Projekten eingesetzt werden, häufig auch selbst durch die Community des entsprechenden Projekts und z.T. auch im Rahmen eines eigenständigen, explizit zu diesem Zweck gegründeten OSS-Projekts entwickelt, wie dies z.B. für das Bugzilla-System durch die Mozilla-Community oder für die Software LXR[67] durch die Linux-Kernel Community praktiziert wurde (vgl. [Dietze03]).

4.2.1 Kommunikationswerkzeuge

Eine sehr wichtige Ressource für die Durchführung eines OSS-Projekts sind die Kommunikationswerkzeuge, welche der heterogenen und global verteilten Community eines Projekts die projektinterne Kommunikation ermöglichen und damit die Voraussetzung für die Durchführung der Entwicklungsprozesse schafft. Die Notwendigkeit für eine zweckmäßige Kommunikationsinfrastruktur resultiert vor allem aus dem Fehlen der Möglichkeit zu direkter persönlicher Kommunikation unter den Projektmitgliedern. Daraus läßt sich die primäre Anforderung an alle instrumentalisierten Werkzeuge ableiten, dass sie über entsprechende Webschnittstellen verfügen und über die Protokolle des Internet ausführbar sein müssen.

Die folgenden Kommunikationswerkzeuge konnten projektübergreifend identifiziert werden, wobei deren Charakteristika und Verwendungsweise in [Dietze03c] konkret dargestellt und z.T. auch aus den konkreten Prozessdarstellungen ersichtlich wird:

[67] URL: http://lxr.linux.no; Abfrage: 12.03.2003

- HTTP-Server zur Bereitstellung der Website des Projekts
- Mailing Listen
- Newsgroups
- Internet Relay Chat (IRC)

Ein klare Abgrenzung zwischen den Kategorien der Kommunikations- und Kollaborationswerkzeuge läßt sich zwar nur schwer durchführen, da die meisten Werkzeuge sowohl kollaborative als auch kommunikative Aufgaben unterstützen, aber ein primärer Verwendungszweck ist i.d.R. durch die jeweiligen Merkmale eines Werkzeugs gegeben und dient somit als Zuordnungskriterium zu der jeweiligen Kategorie.

4.2.2 Kollaborationswerkzeuge

Neben der Klasse der Kommunikationswerkzeuge wurde die Klasse der Kollaborationswerkzeuge definiert, welche primär der Unterstützung der kollaborativen und dezentral verteilten Entwicklungsprozesse dienen. Auch in diesem Kontext ist deren Verwendbarkeit über die Protokolle des Internet von besonderer Bedeutung, so dass alle identifizierten Tools zumindest über eine webfähige, browserbasierte Benutzerschnittstelle verfügen.

Die folgenden Software-Werkzeuge konnten in diesem Rahmen projektübergreifend identifiziert werden und werden in [Dietze03c] eingehender beschrieben:

- Bug Tracking System
- Quellcodeverwaltungs- und Konfigurationsmanagementsystem (z.B. CVS)

Diese Systeme repräsentieren die maßgeblichen Elemente der typischen Software-Infrastruktur, die durch ihre jeweiligen Charakteristika die Entwicklungsprozesse und die erstellten Artefakte, respektive die Patches und Change Requests, maßgeblich determinieren.

4.2.3 Entwicklungswerkzeuge

Die einzelnen Tools der Klasse der Entwicklungswerkzeuge werden nicht zentral durch die Maintainer, sondern individuell durch jeden Entwickler instrumentalisiert und können daher kaum projektweit oder gar projektübergreifend verallgemeinert bzw. generalisiert werden. Daher werden in diesem Kontext in [Dietze03c] lediglich einige wichtige Kategorien der Entwicklungswerkzeuge und weithin etablierte Ausprägungen dieser Kategorien dargestellt:

- Editoren
- Compiler
- Debugger
- Diff (zur Patchextraktion)

Determiniert wird die Wahl der verschiedenen, lokal verwendeten Entwicklungswerkzeuge lediglich durch die Anforderungen des Projekts oder durch Empfehlungen der Maintenance-Instanzen bezüglich einzelner Werkzeuge. Die Wahl einiger Werkzeuge wird z.T. auch implizit durch den praktizierten Entwicklungsprozess bzw. die zentrale Kollaborationsinfrastruktur determiniert, wie dies z.B. im Falle des Werkzeugs zur Patch-Extraktion (diff) der Fall ist, da mit dieser Software die erstellten Patches für eine Integration in das zentrale CVS-Repository aufbereitet werden.

4.3 Identifizierte technologische Artefakte

Entsprechend dem Metamodell werden die Artefakte in die Kategorien Management- und Technologische Artefakte eingeteilt und entsprechend dieser Kategorien detailliert in [Dietze03c] dargestellt. In diesem Abschnitt wird lediglich ein Überblick über die zentralen technologischen Artefakte gegeben, die entscheidend für das

weitere Verständnis des deskriptiven Modells sind und für weiterführende Informationen auf [Dietze03c] verwiesen. Da die Managementartefakte in OSS-Projekten bisher nur sehr rudimentär entwickelt sind, wurde an dieser Stelle auf die konkrete Darstellung der verschiedenen Ausprägungen verzichtet.

Während die Managementartefakte übergreifende und koordinierende Zielsetzungen zur Unterstützung der Entwicklungsprozesse verfolgen, sind die technologischen Artefakte direkt in den Entwicklungsprozess involviert und werden daher entsprechend ihrem primären Verwendungszweck in die Kategorien *Requirements-, Design-, Implementation-* oder *Deployment Artifact* eingeordnet. Das folgende Modell stellt eine gesamtheitliche Übersicht über alle identifizierten technologischen Artefakte und ihre elementaren Beziehungen zur Verfügung:

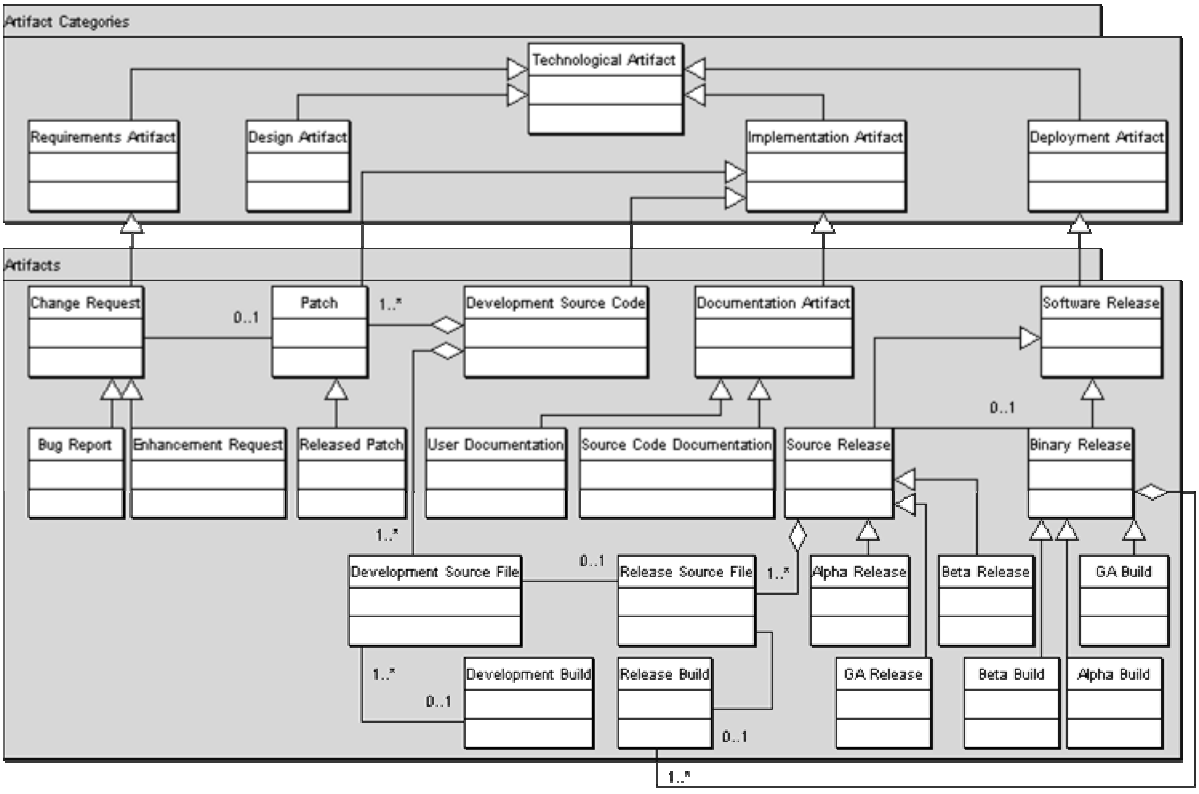

Abb. 4.2: Artefaktmodell der technologischen Artefakte

Alle Artefakte wurden einer Artefakt-Kategorie entsprechend dem Metamodell (vgl. *2.2.3 Artefakt*) zugeordnet, was durch die Spezialisierungsbeziehungen dargestellt wird. Exemplarisch wird im folgenden auf die zentralen Artefakte der Entwicklungsprozesse eingegangen, die durch ihre identifizierten Charakteristika maßgeblich die Entwicklungsprozesse determinieren:

- Change Request
- Patch

4.3.1 Change Request

Das primäre Artefakt zur Spezifikation von Anforderungen wird durch den sogenannten Change Request repräsentiert. Ein Change Request repräsentiert einen Datenbankeintrag in einem Bug Tracking System, wobei in den analysierten Projekten die Systeme Jitterbug und Bugzilla identifiziert wurden und letzteres als ein im OS-Kontext besonders weitreichend etabliertes System betrachtet werden kann. Diese Systeme dienen der

strukturierten Verwaltung von Änderungsanforderungen in einem datenbankbasierten System über eine Web-Schnittstelle und ermöglichen allen Anwendern des Systems die Beschreibung einer Anforderung basierend auf Metadaten. Change Requests können in zwei grundlegende Kategorien eingeteilt werden:

- Bug Report
- Enhancement Request (RFE – Request For Enhancement)

Ein RFE, also die Anforderung einer funktionalen Systemerweiterung, wird genau wie ein Bug Report verwaltet und unterscheidet sich lediglich bezüglicher einiger Attribute. Beispielsweise wird im Rahmen des Apache- bzw. Mozilla-Projekts einem Attribut *(Severity)* eines RFE der Wert *Enhancement* zugewiesen.

Die Rahmenbedingungen für die Erstellung eines individuellen Change Requests werden durch den existierenden Quellcode und die formulierten Anforderungen der Maintainer, z.B. bezüglich der Kompatibilität zu verschiedenen Plattformen, gebildet. Ergänzt werden diese datenbankbasierten Change Requests durch informell über die Kommunikationskanäle kommunizierte und diskutierte Änderungsanforderungen oder Fehlerberichte, welche aber idealtypischerweise vor dem Beginn eines darauf basierenden Patchentwicklungszyklus immer zu einem Eintrag im Bug Tracking System des Projekts führen.

4.3.1.1 Format eines Change Request - Metadaten

Ein Bug Tracking System ermöglicht die Verwaltung von Change Requests über verschiedene Metadaten und Attribute, wodurch ein entsprechender Eintrag sehr konkret spezifiziert werden kann und die erfassten Informationen in strukturierter Form vorliegen. Zur Generalisierung eines verallgemeinerbaren Formats und Lebenszyklus' wurde versucht, die generalisierbare Schnittmenge der individuell in den durchgeführten Fallstudien identifizierten Metadaten eines Change Request und deren Ausprägungen zu identifizieren (vgl. [Dietze03c]). Die folgenden typischerweise erfassten Attribute wurden identifiziert (vgl. [Dietze03]):

- *Owner:* Inititator des Eintrages
- *Version:* Versionsnummer der Software auf die sich die Anforderung bezieht
- *Component:* Abgrenzbare Problem-Domäne, welcher der Change Request zugeordnet werden kann
- *OS:* Betriebssystem der betreffenden Software
- *Plattform:* Verwendete Hardwareplattform
- *Severity:* Auswirkungen eines Softwarefehlers
- *State:* Status der Bearbeitung des Softwarefehlers
- *Assigned:* Name des Entwicklers, dem die Bearbeitung zugeordnet ist
- *Summary:* Zusammenfassung
- *Description:* Umfassende Beschreibung
- *URL:* URL mit weiterführenden Informationen oder zu Reproduktion des Fehlers

Das Attribut *Component* beschreibt eine Problem-Domäne der Software, welcher sich ein Change Request zuordnen läßt und die i.d.R. aus einem oder mehreren Quellcode-Artefakten besteht (vgl. auch [ReFo02][68].)

Häufig konnten auch Attribute identifiziert werden, welche dazu dienen, den Change Request mit Schlagwörtern zu versehen und dadurch die Recherche im Bug Tracking System zu erleichtern (*Keywords*) oder zur Verknüpfung von ergänzenden Dateien (z.B. Log Files, Systemfehlermeldungen o.ä.) mit dem Change Request (*Attachments*).

Das Attribut *Severity* beschreibt die Auswirkungen eines Softwarefehlers und wird anhand der folgenden Ausprägungen eingestuft (vgl. [Dietze03]).

[68] „Most code modules in Mozilla have one or more associated components in the Bugzilla bug tracking tool." [ReFo02]

- *Blocker:* Fehler behindert weitere Entwicklungsarbeiten
- *Critical:* Fehler bewirkt Systemabstürze oder Datenverlust
- *Major:* Grundlegende Funktionalität wird verhindert
- *Normal:* Durchschnittlicher Softwarefehler
- *Minor:* Problem kann mit minimalem Aufwand umgangen werden
- *Trivial:* Trivialer Fehler, wie z.B. Rechtschreibfehler
- *Enhancement:* Change Request beschreibt keinen Fehler, sondern die Anforderung einer neuen Funktionalität oder Systemerweiterung

Dieses Attribut dient zudem der Definition von Prioritäten der Bearbeitung der Change Requests und steuert dadurch die darauf basierenden Patchentwicklungsprozesse in entscheidendem Maße.

In einigen Projekten konnte beobachtet werden, dass zwar die meisten Attribute durch jeden Akteur bei der Erstellung definiert werden können, sofern ein Nutzer-Account angelegt wurde, aber die Modifikation von bereits erstellten Metadaten häufig nur entsprechend privilegierten Entwicklern vorbehalten ist.

4.3.1.2 Lebenszyklus eines Change Request

Eines der wichtigsten Attribute im Kontext beider Projekte ist das *State*-Attribut, welches über verschiedene Stati den Lebenszyklus eines Bug Reports bzw. Change Requests definiert und damit auch den Lebenszyklus eines Patches entscheidend determiniert.

Das auf Basis der verschiedene Ausprägungen des Statusattributs in den untersuchten Entwicklungsprozessen generalisierte Modell des Change Request-Lebenszyklus wird in der folgenden Grafik in der Zustandssicht modelliert:

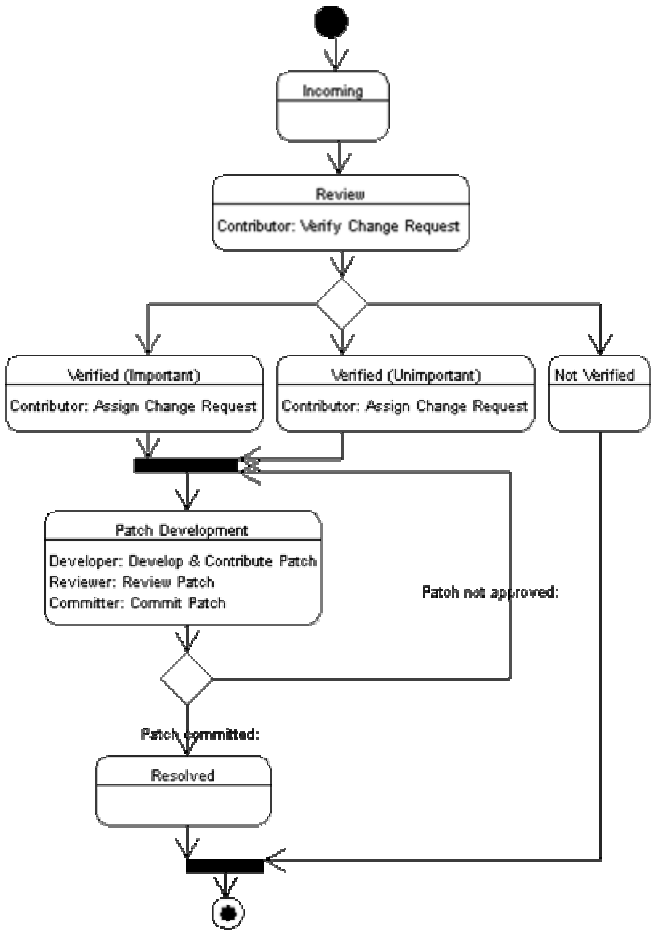

Abb. 4.3: Zustandssicht eines Change Request

Unter der Bezeichnung der jeweiligen Zustände wurden die in der Phase dieses Zustandes auszuführenden Aktivitäten und die idealtypischerweise dafür zuständigen Rollen angeführt. Die Durchführung dieser Aktivitäten, welche im Rahmen der Entwicklungsprozesse näher dargestellt werden (vgl. auch [Dietze03c]), ist für die Transition des Artefakts in den nächsten Zustand notwendig. Eine detailliertere Erläuterung dieses Lebenszyklus' und des zugrundeliegenden Generalisierungsprozesses findet sich in [Dietze03c].

4.3.2 Patch

Das wichtigste Implementationsartefakt im Rahmen des sukzessiven Softwareverbesserungsprozesses ist das Patch. Ein Patchentwicklungsprozess basiert stets auf einem dedizierten Change Request, welcher durch das Patch implementiert wird. Dies bedeutet, dass die Aktivitäten zur kollaborativen und zur individuellen Anforderungsdefinition wichtige Voraussetzungen für die Entwicklung eines Patches darstellen. Patches werden im ASCII-Text Format durch die Ausführung der explizit in *5.1.3 Patchentwicklungsprozess* dargestellten Aktivitäten erstellt und beinhalten lediglich die Differenzmenge des aktuellen, zentralen Entwicklungsquellcodes und des lokal durch den Patchentwickler modifizierten Quellcodes.

Patch-Lebenszyklus

Basierend auf den identifizierten Entwicklungsaktivitäten, welche mit der Erstellung und der Integration eines Patches verbunden sind, konnte ein Patch-Lebenszyklus identifiziert werden, welcher analog zum Lebenszyklus eines Change Requests in der Zustandssicht modelliert wird:

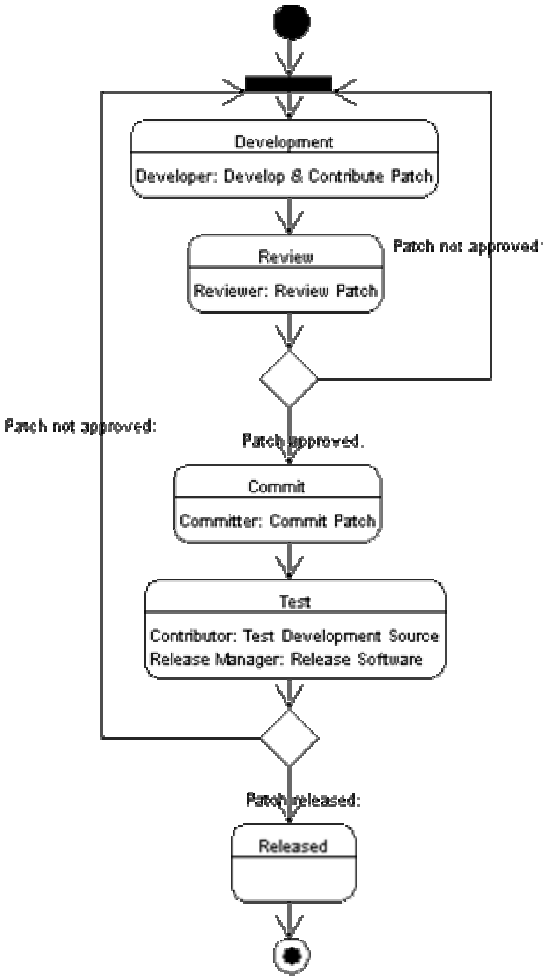

Abb. 4.4: Zustandssicht des Implementationsartefakts *Patch*

Hierbei ist zu beachten, dass der dargestellte Zyklus lediglich implizit aufgrund der typischen Patchentwicklungsprozesse identifiziert werden konnte und im Gegensatz zum Change Request nicht softwaretechnische durch die Definition von Metadaten abgebildet oder unterstützt wird. Die einzelnen Zustände des Patches wurden in der Abbildung durch die Aktivitäten und die dafür zuständigen Rollen erweitert, welche zur Transition in den nächsten Zustand notwendig sind. Der Lebenszyklus des Patches läßt sich z.T. direkt mit dem Lebenszyklus des zugrundeliegenden Anforderungsartefakts, respektive Change Requests verknüpfen. Dies bedeutet, dass der Zustandsübergang eines Patches direkt die Transition des Zustands des zugrundeliegenden Change Requests nach sich ziehen sollte. Der Zustand *Patch Development* wird von einem Change Request beispielsweise genau dann eingenommen, wenn sich das korrelierende Patch in den Zuständen *Development*, *Review*, oder *Commit* befindet. Außerdem nimmt ein Anforderungsartefakt den Zustand *Resolved* erst dann ein, wenn das damit assoziierte Patch die Zustände *In Test* oder *Released* einnimmt.

Ein Patch nimmt den Zustand *Released* ein, wenn es als Bestandteil eines Software-Release oder als explizites Artefakt *Released Patch* veröffentlicht wird. Nach dem Release eines Patches ist es nicht mehr möglich, diese spezielle Instanz eines Patches zu modifizieren. Nachträglich identifizierte Mängel können somit nur durch die Ausführung eines erneuten Patchentwicklungsprozesses und der damit verbundenen, vorausgehenden Anforderungsdefinition behoben werden.

Released Patches
Sogenannte Released Patches repräsentieren Patches, welche bereits in den aktuellen Entwicklungsquellcode integriert und getestet wurden und in Form eines expliziten Release-Artefakts über die HTTP- und FTP-Server des Projekts zum allgemeinen Download zur Verfügung gestellt werden, um einen Softwarefehler im momentan aktuellen General Availability(GA)-Release (vgl. [Dietze03c]) zu beheben. Diese Form von explizit veröffentlichten Patches ist von dem Release eines Patches als Bestandteil eines neuen Software-Release abzugrenzen und wird typischerweise nur für besonders kritische Softwarefehler durchgeführt.

5 Deskriptives Modell der identifizierten Prozesse

In diesem Kapitel werden exemplarisch einige der identifizierten Teilprozesse dargestellt, wobei die Gesamtheit aller Prozesse, Teilprozesse und Aktivitäten in den verschiedenen Sichten des Metamodells detailliert in [Dietze03c] beschrieben ist. Zudem sind in [Dietze03c] die statischen Sichten enthalten, welche den Zusammenhang zwischen Artefakten, Rollen, Softwarewerkzeugen und einzelnen Teilprozessen darstellen. Eine Übersicht über die identifizierten Prozesse und die darin involvierten Artefakte und Rollen ist in *5.4 Überblick über die identfizierten Prozesse* enthalten.

5.1 Entwicklungsprozesse

Dieses Kapitel beschreibt anhand detaillierter Prozessdarstellungen entsprechend dem Metamodell ein deskriptives Modell der im OSS-Kontext praktizierten Entwicklungsprozesse.

In der folgenden Anwendungsfallsicht wurden die Entwicklungsprozesse und die involvierten Prozesse und Teilprozesse erfasst:

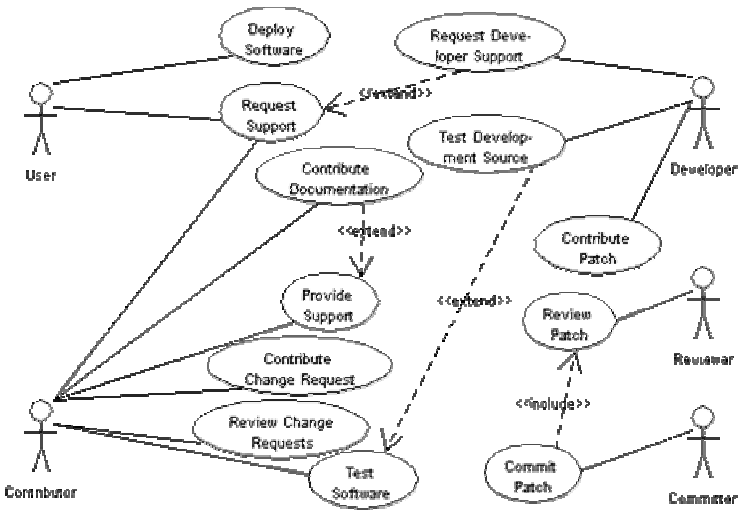

Abb. 5.1: Use Case View der Entwicklungsprozesse

Die in der Grafik dargestellten Anwendungsfälle repräsentieren die einzelnen Prozesse und Teilprozesse der Entwicklungsprozesse, welche im folgenden bzw. in [Dietze03c] eingehender dargestellt werden:

- Deployment der Software (*Deploy Software*)
- Software-Test (*Test Software*)
- Anwender- und Entwicklersupport (*Request Support* und *Provide Support*)
- Beitrag von Change Requests (Kollaborative Anforderungsdefinition) (*Contribute Change Request*)
- Review der Anforderungsartefakte (*Review Change Requests*)
- Patchentwicklung (*Contribute Patch*, *Review Patch*, *Commit Patch*)
- Beitrag von Dokumentationsartefakten (*Contribute Documentation*)

Die verschiedenen Prozesse stehen nur zum Teil in einem gemeinsamen zeitlichen Kontext. Dies bedeutet, dass die Durchführung einzelner Teilprozesse meist unabhängig von anderen Prozessen erfolgt. Daher wird die Modellierung der Prozesse auf diesem Abstraktionsniveau in der Anwendungsfallsicht in Form eigenständiger Anwendungsfälle als repräsentativ betrachtet. Sofern diese Unabhängigkeit nicht gegeben sein sollte, wie dies z.B. bei den Teilprozessen *Contribute Patch, Review Patch* und *Commit Patch* der Fall ist, werden diese Teilprozesse im folgenden in einem gemeinsamen Kontext betrachtet und modelliert.

Im folgenden werden nur einige exemplarische Sichten auf zentrale Teilprozesse der Entwicklungsprozesse dargestellt, während die Gesamtheit aller Sichten auf alle identifizierten Entwicklungsprozesse entsprechend dem dargestellten Metamodell in [Dietze03c] enthalten ist.

5.1.1 Generierung von Change Requests – Kollaborative Anforderungsdefinition

Der Prozess des Beitragens von Change Requests kann auch als kollaborative Anforderungsdefinition bezeichnet werden, da durch die kollaborative Generierung von Change Requests Anforderungen definiert werden, die an alle Entwickler der Community adressiert sind und in dieser Phase somit noch nicht als Vorgabe für einen bestimmten Entwickler dienen. Dies ist von der individuellen Anforderungsdefinition abzugrenzen, bei der ein Change Request einem spezifischen Entwickler zugeordnet bzw. von diesem ausgewählt wird und somit die individuellen Anforderungen für einen spezifischen Patchentwicklungsprozess konkretisiert werden.

I.d.R. werden beide Kategorien der Change Requests (Bug Report, Enhancement Request) in einem dedizierten Bug Tracking System verwaltet. Die Aktivitäten zur Generierung und Verbreitung eines Change Requests unterscheiden sich aber für beide Teilprozesse nur marginal und werden in allen evaluierten Projekten durch sogenannte Bug Reporting Guidelines reglementiert (vgl. [Dietze03]), welche im Prinzip ein präskriptives Modell dieses Teilprozesses definieren.

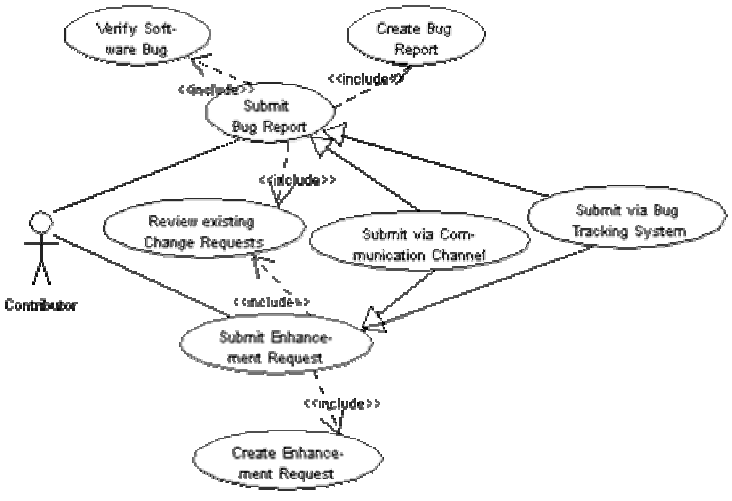

Abb. 5.2: Use Cases *Contribute Change Requests*

Im Falle eines identifizierten Softwarefehlers wird dieser idealtypischerweise zuerst lokal durch den Contributor verifiziert, indem dieser das Fehlverhalten in der aktuellen Version des Quellcodes reproduziert. Dem schließt sich die Recherche in den existierenden Change Requests an, um die Neuheit des Change Request zu verifizieren, was bei positivem Resultat zur Generierung eines neuen Change Request bzw. Bug Reports führt.

Change Requests können durch einen Contributor direkt im Bug Tracking System generiert und zentral zur Verfügung gestellt werden oder auch über klassische Kommunikationskanäle wie z.B. dedizierte Entwickler Mailing Listen oder -Newsgroups kommuniziert werden. Auch falls der Anwender den Change Request vorerst nur über die Kommunikationskanäle kommuniziert, dient dies lediglich einer weiteren Validierung des Change Request durch die Adressaten der Mailing List und führt nach seiner Verifikation durch die Adressaten idealtypischerweise ebenfalls zur Generierung eines Eintrags im Bug Tracking System.

Das konkrete Format eines Change Requests und dessen teilprozessübergreifender Lebenszyklus wird in *4.3.1 Change Request* näher konkretisiert. Die folgende Grafik visualisiert die Teilprozesse *Contribute Bug Report* und *Contribute Enhancement Request* in einer allgemeinen Aktivitätssicht für die Generierung eines Change Request.

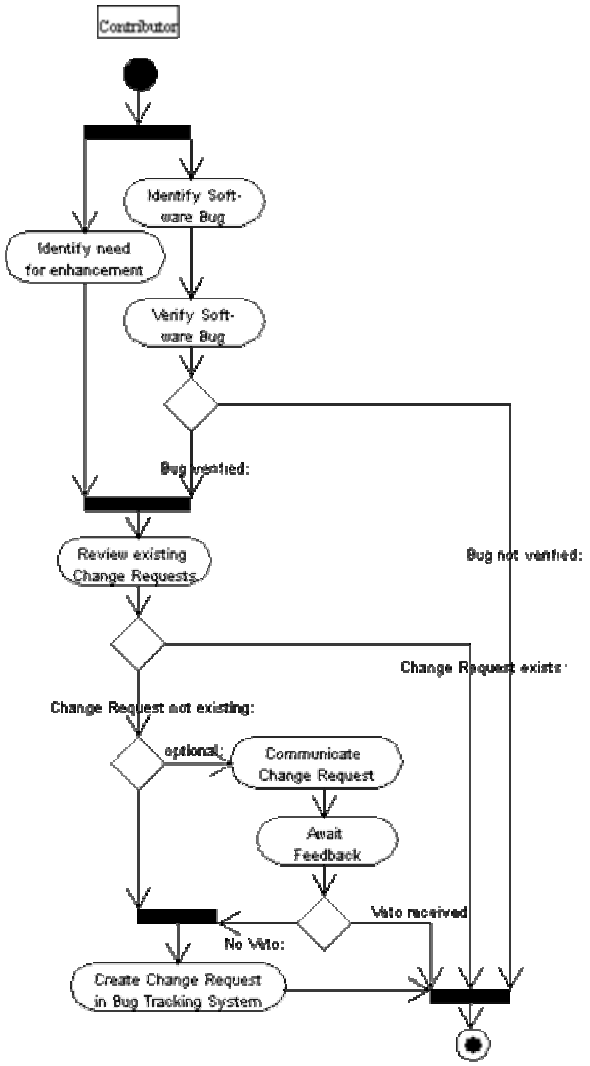

Abb. 5.3: Aktivitätsdiagramm des Prozesses *Contribute Change Request*

Im folgenden werden die Aktivitäten, welche einer näheren Betrachtung bedürfen, eingehender konkretisiert.

5.1.1.1 Verifikation des Change Request

Bevor ein Change Request generiert wird, muss dessen Relevanz für die Community durch den Contributor validiert werden, wobei im Falle eines Softwarefehlers sichergestellt werden sollte, dass das zu beschreibende

Fehlverhalten in der aktuellsten und unveränderten Version der Software und im optimalen Fall auch im Entwicklungsquellcode reproduziert werden kann und somit von Bedeutung für die gesamte Community der Software ist. Im Kontext eines RFE ist lediglich sicherzustellen, dass die Anforderung auch in der aktuellsten Softwareversion existiert und noch nicht als zusätzliche Funktionalität eines neuen Releases enthalten ist.

5.1.1.2 Recherche in existierenden Change Requests

Nachdem der Change Request lokal durch den Contributor verifiziert wurde, wird die Aktualität des Beitrags überprüft, indem die existierenden Change Requests nach Einträgen mit verwandtem Inhalt durchsucht werden. Hierzu wird v.a. die Datenbasis des Bug Tracking Systems verwendet, wobei aber auch existierende Mailing Listen- oder Newsgroup-Archive nach relevanten Einträgen durchsucht werden können, in welchen die zu definierende Anforderung evtl. bereits dokumentiert und somit obsolet ist. Häufig ist dies im Falle von vermeintlichen Softwarefehlern der Fall, welche bereits im Rahmen einer vorangegangenen Diskussion als Anwenderfehler eingestuft wurden. In diesen Recherche-Prozess sollten sowohl alle aktiven und offenen, als auch die breits geschlossenen Change Requests einbezogen werden.

5.1.1.3 Kommunikation des Change Request

Vor der eigentlichen Generierung eines Change Requests kann dieser über verschiedene Kommunikationskanäle kommuniziert werden, um dadurch im Vorfeld einen Review des Change Request durch die Community zu forcieren. Dazu können z.B. dedizierte Mailing Listen und Newsgroups verwendet werden, welche speziell Entwickler oder Kompetenzträger bezüglich des betroffenen Subsystems adressieren. Außerdem sollten neben der gesamten Entwicklergemeinde im Rahmen dieser Aktivität direkt die Maintainer oder ursprünglichen Entwickler des Moduls angesprochen werden, um dadurch etwaige lokal bereits durchgeführte Entwicklungsarbeiten bezüglich der Anforderung zu identifizieren. Dies ermöglicht die Diskussion des Change Request durch die gesamte Community und unterstützt somit die Qualität der tatsächlich im Bug Tracking System generierten Einträge, indem nicht verifizierbare Change Requests bereits vor einer Erstellung eines Eintrags im Bug Tracking System verworfen werden können.

5.1.1.4 Erstellung des Change Request im Bug Tracking System

Nach der erfolgreichen Durchführung der beschriebenen Aktivitäten wird das Artefakt Change Request im Bug Tracking System durch den Contributor generiert. Der Akteur muß sich hierzu über einen Nutzer-Account identifizieren, bzw. diesen zuerst generieren und ist über seine E-Mail-Adresse eindeutig identifizierbar.
Zur genauen Spezifikation eines Change Requests existieren in den identifizierten Systemen, respektive Bugzilla und Jitterbug, verschiedene Metadaten in Form von Attributen, welche durch den Initiator eines Eintrags bereitgestellt werden müssen und in *4.3.1 Change Request* detailliert dargestellt wurden. Ein elementares Attribut repräsentiert in beiden Systemen der *Status* eines Change Request, da dieser den Lebenszyklus des Artefakts Change Request definiert. Weiterhin werden im Rahmen dieser Aktivität alle relevanten Informationen, wie z.B. Systemausgaben oder Log File-Einträge im Falle eines Bug Reports definiert und falls nötig in Form eines Anhangs mit dem Change Request assoziiert.

5.1.2 Kollaborativer Requirements Review und individuelle Anforderungsdefinition

Der Prozess des kollaborativen Review der Anforderungen kann durch alle interessierten Akteure ausgeführt werden und wird in einigen Projekten auch als Bug Triage bezeichnet. Im Rahmen dieses Prozesses werden die projektweit definierten Anforderungen, respektive die im Bug Tracking System generierten Change Requests, einer genaueren Untersuchung unterzogen. Hierbei wird die Zielsetzung verfolgt, die Bearbeitung offener Change Requests zu koordinieren, neu generierte Change Requests zu verifizieren und verifizierte Change Requests evtl. bestimmten Entwicklern zur weiteren Bearbeitung zuzuordnen. Diese Zuordnung kann auch als individuelle Anforderungsdefinition bezeichnet werden, da hierbei der Akteur einen Change Request einem bestimmten Entwickler oder sich selbst zuordnet und dadurch die individuellen Anforderungen für den darauf basierenden Patchentwicklungszyklus definiert.

Dieser Prozess kann durch alle Akteure ausgeführt werden, welche über das Privileg verfügen, die entsprechenden Attribute des Artefakts Change Request zu definieren und zu modifizieren, wobei aber häufig festgestellt wurde, dass diese Tätigkeit durch einige wenige Entwickler in periodischen Abständen durchgeführt wird, um der Entwicklergemeinde die weitere Bearbeitung zu ermöglichen und zu vereinfachen. Beispielsweise wird dieser Teilprozess in einigen Projekten durch die Modul Owner bezüglich aller Change Requests ausgeführt, welche sich auf das von ihnen verwaltete Modul beziehen. Im Rahmen des Mozilla- und des Apache Projekts werden die Change Requests sogenannten Components zugeordnet, welche als spezielles Attribut des Artefakts Change Request verwaltet werden und eine bestimmte Problemdomäne abbilden (vgl. [Dietze03]). Diese Problemdomäne wird häufig durch eine bestimmte Teilmenge von Modulen abgebildet. Der sogenannte Component Owner ist in diesem Kontext für die Koordination und den Review aller Change Requests zuständig, die mit der ihm zugeordneten Component assoziiert werden können.

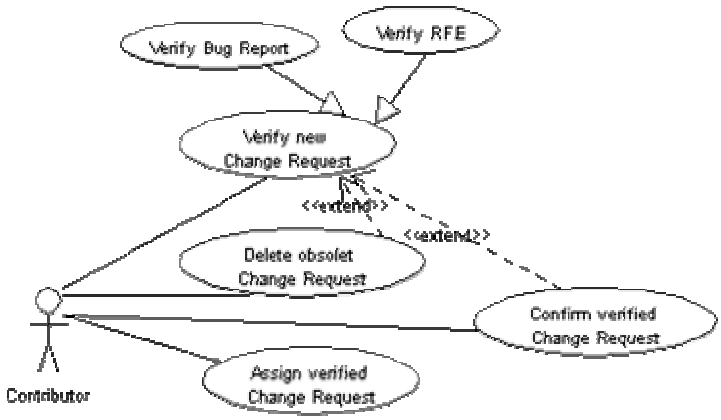

Abb. 5.4: Anwendungsfallsicht des Prozesses *Review Change Requests*

Der Prozess des Review der Change Requests setzt sich somit aus den Teilprozessen zur Verifikation eines neu generierten Change Requests, welche zur Bestätigung oder zur Löschung von nicht verifizierbaren Change Requests führen können, und der Zuordnung des Change Requests zu einem bestimmten Entwickler zusammen. Die folgende Aktivitätssicht modelliert diese beiden Teilprozesse in einem gemeinsamen Aktivitätsdiagramm.

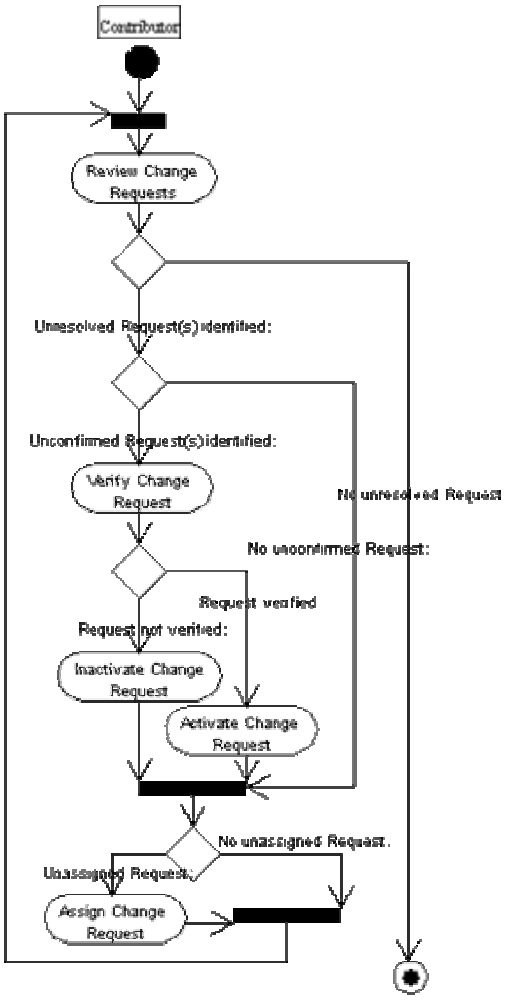

Abb. 5.5: Aktivitätssicht der Teilprozesse des Review von Change Requests

Dieser Prozess wird idealtypischerweise periodisch in bestimmten Abständen iterativ ausgeführt, um die Basis der zentralen Anforderungsartefakte in einen redundanzfreien, aktuellen und konsistenten Zustand zu bringen. Da aber i.d.R. keine explizite Rolle für diesen wichtigen Teilprozess identifiziert werden konnte, wird dieser Prozess in den meisten Projekten nur sehr ungenügend ausgeführt.

Die Aktivität *Inactivate Change Request* hat zur Folge, dass das entsprechende Artefakt entweder vollständig gelöscht oder die erfolglose Validierung über die Zuweisung eines entsprechenden Status signalisiert wird. Im Mozilla-Kontext wird hierfür z.B. dem *Status* der Wert *Resolved* zugewiesen mit einer entsprechenden Definition eines weiteren Attributs (*Resolution*) als *Invalid* oder *Duplicate* (vgl. [Dietze03]). Im Kontext der Linux Kernel Entwicklung wird dem Change Request in diesem Fall der Status *Discarded* zugewiesen.

Generell vereinfachen die Metadaten, die bereits explizit dargestellt wurden, den Prozess des Review der Change Requests und die Recherche in existierenden Change Requests immens, da anhand der vergebenen Attribute sukzessive verschieden Suchkriterien definiert werden können. Durch die Aufteilung des Gesamtsystems in Problemdomänen, respektive Components, ist eine problemdomänenspezifische Betrachtung der Anforderungsartefakte möglich.

In den folgenden Abschnitten werden die einzelnen Aktivitäten zur Verifikation des Change Requests und der Zuordung zu einem bestimmten Entwickler detailliert dargestellt, wobei auf dieser Abstraktionsebene auch zwischen Bug Reports und RFEs unterschieden wird.

5.1.2.1 Verifikation der Change Requests

Die Verifikation der Change Requests hat das Ziel, den Change Request zu bestätigen, falls er als reproduzierbare und noch nicht publizierte Anforderung eingestuft wird oder das Artefakt andernfalls zu deaktivieren.
Dies wird im Mozilla- bzw. Apachekontext, respektive im Bugzilla-System mit allen Change Requests, welche den Status *Unconfirmed* aufweisen durch die Zuweisung der Stati *New (inactive)* oder *Resolved* durchgeführt, falls die Verifikation des Change Request nicht möglich war. Im Falle von Jitterbug im Linux Kernel Projekt wird Change Requests mit dem Status *Incoming* der Status *Deferred* bzw. *Discarded* im Falle einer negativen Validierung und andernfalls der Status *Pending* zugewiesen (vgl. [Dietze03]).

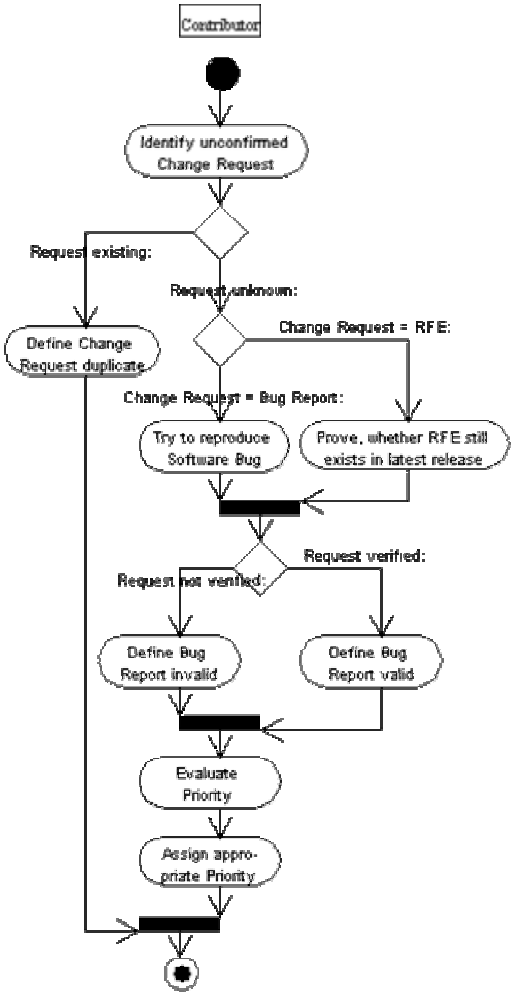

Abb. 5.6: Aktivitätssicht der Verifikation des Change Requests

Nach der erfolgreichen Verifikation des Change Request wird die bisher vergebene Priorität evaluiert und gegebenenfalls dem entsprechenden Attribut ein adäquater Wert zugewiesen. Dies kann indirekt über die Vergabe eines entsprechenden Status, z.B. *Deferred* oder *Discarded* bei Jitterbug, oder direkt über die Definition eines dedizierten Attributs *Priorität* erfolgen, wie dies mit dem Bugzilla-System praktiziert wird (vgl. [Dietze03]).

Ergänzt wird dieses Attribut im Bugzilla-System durch die Eigenschaft *Severity*, welches die Auswirkungen eines Softwarefehlers definiert und im Falle eines RFE den Wert *Enhancement* aufweist.

5.1.2.2 Individuelle Anforderungsdefinition – Zuordnung der Change Requests

Ein bereits verifizierter Change Request wird einem bestimmten Entwickler zur Bearbeitung zugeordnet, wobei im Falle von Mozilla bzw. Apache der bereits beschriebene Component Owner als Bearbeiter angenommen wird, falls kein anderer Entwickler explizit definiert wurde. I.d.R. repräsentiert der Akteur dieser Aktivität zugleich auch den Entwickler des entsprechenden Patches, da ein Entwickler meist selbständig die Change Requests auswählt, welche er durch ein Patch bearbeiten möchte. Nur in seltenen Fällen erfolgt die Zuweisung durch einen anderen Akteur, z.B. durch einen Akteur, der den Review der Change Requests durchführt.
Diese Zuordnung zu einem Entwickler wird ebenfalls über ein dediziertes Attribut durchgeführt, welches auf den Bearbeiter verweist. Außerdem hat diese Aktivität typischerweise die Zuweisung der Stati *Assigned* im Bugzilla-Kontext und *Pending* im Jitterbug-Kontext zur Folge. Diese Aktivität kann auch als individuelle Anforderungsdefinition bezeichnet werden, da hier basierend auf den gesamten Anforderungen, respektive der Gesamtmenge der Change Requests, individuelle Anforderungen für spezifische Patchentwicklungsprozesse individueller Entwickler definiert werden. Vor der eigentlichen Assoziation eines Change Request mit einem bestimmten Entwickler sollten informelle Recherchen durchgeführt werden, ob evtl. schon Entwickler mit der Patchentwicklung beschäftigt sind, ohne dies im Bug Tracking System signalisiert zu haben.

5.1.3 Patchentwicklungsprozess

Aufgrund der engen Verknüpfung der Patchentwicklung mit den entsprechenden Review- und Commit-Teilprozessen und des gemeinsamen zeitlichen Kontexts der involvierten Aktivitäten, werden diese Teilprozesse in diesem Abschnitt gesamtheitlich thematisiert. Der Prozess der Patchentwicklung und –integration ist zu großen Teilen mit dem Prozessmodell-Ansatz des Prototyping nach [NoSt98] vergleichbar.

Jeder Patchentwicklungszyklus im sukzessiven Softwareverbesserungsprozess basiert typischerweise auf einem Bug Report oder einem RFE. Daher ist der Patchentwicklungszyklus direkt mit dem Lebenszyklus des korrelierenden Change Requests verknüpft (vgl. *4.3.1.2 Lebenszyklus eines Change Request*). Der Patchentwicklungsprozess und die damit verbundenen Review- und Integrationsprozesse werden i.d.R. durch entsprechende Guidelines, wie z.B. Programmierkonventionen oder prozedurale Richtlinien reglementiert.

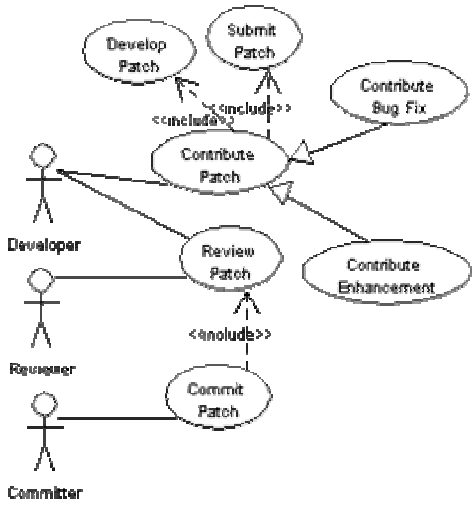

Abb. 5.7: Anwendungsfallsicht des Patchentwicklungszyklus

5.1.3.1 Entwicklungszyklus

Idealtypischerweise ist die Aktivität zur individuellen Anforderungsdefinition basierend auf den kollaborativ erstellten, im Bug Tracking System verwalteten Anforderungen Vorbedingung für die Ausführung eines Patchentwicklungszyklus. Die einzelnen Aktivitäten der Teilprozesse *Contribute Patch, Review Patch* und *Commit Patch,* aus denen sich der Entwicklungszyklus zusammensetzt, sind direkt mit dem Lebenszyklus eines Change Requests verknüpft und sind gemeinsam in der folgenden Aktivitätssicht spezifiziert:

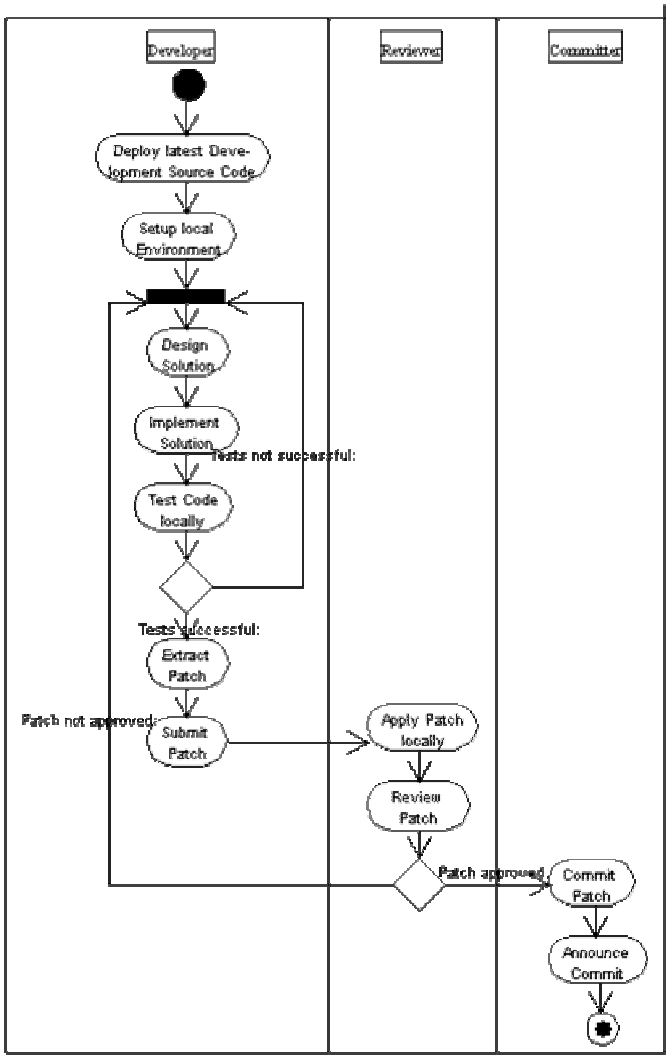

Abb. 5.8: Aktivitätssicht der Teilprozesse *Contribute, Review* und *Commit Patch*

Die in obiger Aktivitätssicht modellierten Teilprozesse variieren z.T. in verschiedenen Projekten in einigen Punkten. So beinhaltet das Mozilla-Projekt z.B. in Abhängigkeit von Art und Umfang des Patches verschiedene Varianten des Reviewteilprozesses (vgl. [Dietze03]). Im folgenden werden die einzelnen Aktivitäten konkret beschrieben.

5.1.3.2 Lokale Entwickung und Kommunikation von Patches

Die lokale Erstellung des Patches setzt sich aus den Aktivitäten zur Bereitstellung einer geeigneten Entwicklungsumgebung, lokalen Entwicklung (inkl. impliziter Entwurfsaktivitäten) und der Durchführung von lokalen Tests zusammen, woran sich die Extraktion und Formatierung des Patches anschließt, um eine weitere

Verbreitung und eine Integration in das zentrale Quellcode-Repository zu ermöglichen. Bevor die eigentlichen Entwicklungsaktivitäten beginnen, wird häufig durch den Entwickler direkt mit dem Inititaor des Change Requests bzw. dem Entwickler oder Maintainer des zu modifizierenden Moduls Kontakt aufgenommen, um etwaige Unklarheiten zu beseitigen und das weitere Vorgehen individuell zu koordinieren.

Einrichtung der Entwicklungsumgebung
Um die Quellcodemodifikationen anhand des aktuellsten Quellcodes vorzunehmen, muß dieser entsprechend des beschriebenen Teilprozesses aus dem aktuellen Entwicklungsarchiv bezogen werden. Anschließend wird das lokale Entwicklungsenvironment eingerichtet, was zur adäquaten Formatierung/Extraktion des Patches notwendig ist. Hierzu werden durch den Entwickler zwei identische Quellcodeverzeichnisse angelegt, welche den aktuellen Quellcode enthalten. Während ein Quellcodebaum nicht verändert wird und somit deckungsgleich zum zentral verwalteten Quellcode bleibt, werden alle Moifikationen im anderen Pfad, dem Entwicklungspfad, vorgenommen. Dies ermöglicht die spätere Extraktion der Quellcodemodifikationen, was im folgenden noch detailliert dargestellt wird.

Lokale Entwicklung
Nach der Etablierung der Entwicklungsumgebung ist die Modifikation des Quellcodes im Rahmen der Patchentwicklung möglich. Hierzu werden i.d.R. klassische Texteditoren verwendet, wobei die verwendeten Werkzeuge individuell differieren und von den persönlichen Präferenzen des Entwicklers und der verwendeten Programmiersprache abhängig sind. Die lokale Entwicklung wird meist durch verschiedene präskriptive Vorgaben determiniert. Diese umfassen z.B. Guidelines, welche den idealtypischen Prozess der Entwicklung und des Tests für ein OSS-Projekt beschreiben und dadurch formalisieren oder Programmierkonventionen, z.B. bezüglich zu verwendender Namenskonventionen oder der Modularisierung des Quellcodes. Zudem ist in diesem Zusammenhang die Bereitstellung von ergänzenden Metadaten durch den Entwickler von erheblicher Bedeutung, um die kollaborative und dezentrale Weiterentwicklung zu ermöglichen. Dies erfolgt z.B. über die Integration des Namen des Autors im Header der Quelltextdatei und die Integration der projektspezifischen Lizenzinformationen. I.d.R. wird in allen Projekten als Anforderung die Verwendung der jeweiligen Lizenz der Ausgangssoftware, respektive der GNU General Public License (GNU GPL) [FSF03][69], der Apache Software License (ASL) [ApacheOrg02][70], oder der Mozilla Public License (MPL) [MozillaOrg02][71] für den modifizierten Quellcode erwartet.

Weiterhin werden im Rahmen dieser Aktivität häufig ergänzende Dokumentationsartefakte generiert oder modifiziert, welche direkt mit dem Patch assoziiert werden können. Dies beinhaltet z.B. ReadMe-Dateien, welche den Quellcode und die durchgeführten Änderungen beschreiben und ähnliche, direkt mit dem Quellcode verknüpfte Artefakte.

Lokaler Software-Test
Anschließend wird der modifizierte Quellcode lokal durch den Entwickler compiliert und ersten Software-Tests unterzogen. Sofern dies zur Identifikation von Fehlverhalten führt, wird der Prozess der lokalen Entwicklung solange iterativ wiederholt, bis die lokale Testphase zufriedenstellend durchgeführt werden konnte.

Extraktion und Formatierung des Patches
In Vorbereitung einer weiteren Verteilung des Patches wird, basierend auf beiden existierenden Quellcodeverzeichnissen, die Extraktion der Quellcodemodifikationen mit dem Werkzeug Diff durchgeführt, welches in [Dietze03] und [Dietze03c] näher erläutert wird. Diff ermöglicht die Extraktion der Differenzmenge beider lokaler Quellcodeverzeichnisse zu einem Patch im reinen ASCII-Text-Format. Dadurch erhält das Patch ein

[69] URL: http://www.fsf.org/licenses/gpl.txt; Abfrage: 12.03.2002
[70] URL: http://www.apache.org/LICENSE.txt; Abfrage: 23.06.2002
[71] URL: http://www.mozilla.org/MPL/MPL-1.1.html; Abfrage: 23.06.2002

standardisiertes Format, welches die spätere Integration des Patches in andere Quellcodearchive, respektive in den Ausgangsquellcode im zentralen CVS-Repository ermöglicht. Zudem stellt ein so generiertes Patch die kleinstmögliche zu übermittelnde Informationsmenge dar, um die Quellcodeänderungen eines Entwicklers der Community zur Verfügung zu stellen.

Kommunikation des Patches
Falls der Entwickler nicht direkt zur Integration des Patches berechtigt ist, kommuniziert er das Patch an die Community, wobei zwei verschiedene Optionen identifiziert werden konnten:

- Kommunikation über Kommunikationskanäle des Projekts
- Verknüpfung des Patch mit dem Change Request im Bug Tracking System

Meist werden beide Optionen parallel angewendet und ergänzen sich gegenseitig. Die Kommunikation des Patches über die Kommunikationskanäle des Projekts erfordert die Verwendung der projektspezifischen Mailing Listen bzw. Newsgroups, welche i.d.R. speziell die Entwickler bzw. Reviewer des Projekts adressieren. Zudem wird das Patch häufig direkt an den Initiator des Change Requests oder den jeweiligen Modul Owner des Quellcode-Moduls gesendet. Das Patch wird hierzu im ASCII-Text Format der Nachricht angehängt bzw. im Falle größerer Patches zentral im Internet veröffentlicht und die entsprechende URL in der Nachricht kommuniziert. Die Nachricht sollte zudem durch eine geeignete Betreffzeile signalisieren, dass sie als Container für ein Patch dient, damit das Patch in der z.T. diffusen Informationsflut stark frequentierter OSS-Projekte entsprechende Aufmerksamkeit findet.

Als weitere Option wird das Patch häufig direkt im Bug Tracking System dem korellierenden Change Request als Datei-Attachment beigefügt ([Dietze03]). Dies wird durch das spezielle Attribut *Attachment* ermöglicht, welches die Verknüpfung eines Anforderungsartefakts mit ergänzenden Dateien ermöglicht. Durch die direkte Verknüpfungsmöglichkeit des Anforderungsartefakts mit dem darauf basierenden Implementationsartefakt stellt diese Vorgehensweise eine probate Möglichkeit zur Verteilung des Patches dar und ermöglicht dadurch einen umfangreichen Reviewprozess, der über die einzelnen Stati des Change Request zentral dokumentiert wird. Diese Alternative der Verteilung von Patches wird i.d.R. ebenfalls durch eine Nachricht über die bereits beschriebenen Kommunikationskanäle begleitet, welche die Veröffentlichung des Patches an die relevante Zielgruppe kommuniziert.

5.1.3.3 Review und Commit

Bevor ein entsprechend privilegierter Committer das übermittelte Patch in den Entwicklungsquellcode integriert, wird es durch die Community, respektive die individuellen Adressaten des Patches, einem Review unterzogen und näher evaluiert. Häufig ist aufgrund der großen Informationsflut großer OSS-Projekte eine mehrfache Kommunikation des Patches notwendig, um entsprechendes Feedback durch die Entwicklergemeinde zu erhalten.

Die hier beschriebene Abfolge der Aktivitäten kann auch als *Review Then Commit*-Strategie (RTC) betrachtet werden (vgl. [AtGaGeLaRo02]). Diese wird häufig durch den *Commit Then Review* -Teilprozess (CTR), also dem Reviewprozess, der sich an den zentralen Commit anschließt, ergänzt und mitunter auch ersetzt. Diese Reviewaktivitäten, welche nach der Integration in den Entwicklungsquellcode ausgeführt werden, wird auch als Pre-Release-Test bezeichnet. Auch wenn der Entwickler über Commit-Rechte verfügt, somit also auch die Rolle des Committers ausübt, wird in diesem Entwicklungsstadium i.d.R. erst der RTC-Review ausgeführt, bevor das Patch in die zentrale Quellcodebasis integriert wird, um das Risiko der Destabilisierung des Quellcodes durch fehlerhafte Patches zu reduzieren. Im Rahmen der untersuchten Projekte werden zwar meist beide Strategien eingesetzt, aber häufig eine der beiden favorisiert. Lediglich im Apache-HTTPD-Projekt wurde eine ausschließliche CTR-Strategie identifiziert, welche aber auch nur für Patches mit sehr geringen Auswirkungen und Interdependenzen auf das Gesamtsystem angewandt wird (vgl. [Dietze03]).

Review (Pre-Commit Review, RTC)
Der Review-Prozess wird primär durch alle Projektakteure ausgeführt, welche Zugang zu dem Patch erhalten haben. Die so erreichte Entwicklergruppe wird z.B. im Rahmen der Apache- und Mozillaprojekte durch eine Gruppe ergänzt, deren Akteure den Status von dedizierten Reviewern besitzen, über eine spezielle Mailing Liste adressiert werden können und den dezentralen und unkoordinierten Review durch die Community ergänzen. Außerdem wird der Review von Patches in besonderem Maße durch etwaig vorhandene Modul Owner des entsprechenden Subsystems bzw. durch den Initiator des zugrundeliegenden Change Requests ausgeführt, welche die Einhaltung verschiedener Anforderungen verifizieren. Der Initiator des Anforderungsartefakts hat z.B. besonderes Interesse an der korrekten Implementation der durch ihn formulierten Anforderungen, während der Modul Owner vor allem an der Korrektheit und an der Stabilität des entwickelten Quellcodes interessiert ist.

Häufig entwickelt sich im Rahmen des Reviewprozesses eine Diskussion zwischen den genannten Stakeholdern (Reviewer, Entwickler, Modul Owner, Change Request Owner), in deren Rahmen der Patchentwicklungszyklus iterativ wiederholt wird, bis alle Interessenten das Patch verifiziert haben. Dieser Konsensbildungsprozess wird häufig über ein Abstimmungsverfahren praktiziert (Mozilla, Apache), wobei z.B. im Rahmen des HTTPD-Projekts eine einstimmige Akzeptanz durch die Reviewer erreicht werden muß (vgl. [Dietze03]). Im Rahmen des Mozillaprojekts basieren die Reviewaktivitäten auf einem mehrstufigen und stark formalisierten Prozess, was aber aufgrund der starken Reglementierung als eher untypisch für den demokratischen und heterogenen Entwicklungsprozess im OSS-Kontext zu betrachten ist.

Integration von Patches: Commit
Die Aktivität des Commit repräsentiert die Integration eines Patches in den zentral verwalteten Entwicklungsquellcode durch einen sogenannten Committer, der über die entsprechenden Schreibrechte auf das CVS-Repository verfügt. Dieser Prozess der Integration in den Entwicklungsquellcode ist von dem Prozess der Integration des Patches in ein offizielles Software-Release abzugrenzen, welcher im Rahmen der Managementprozesse explizit dargestellt wird. Die dedizierte Rolle des Committers wird in den meisten OSS-Projekten explizit zugewiesen und nur in seltenen Fällen wird allen Entwicklern ein uneingeschränktes Commit-Recht eingeräumt. Weiterhin verfügen meist speziell privilegierte Entwickler über CVS-Commit-Rechte bezüglich bestimmter Quellcodeverzeichnisse, Subsysteme oder Problem-Domänen. Häufig wird die Rolle des Committers für ein bestimmtes Modul/Component implizit durch die Modul/Component Owner des jeweiligen Quellcodesegments ausgeführt.

Durch die Kommunikation des Patches und die ausgeführten Reviewaktivitäten ist ein Committer i.d.R. bereits über die etwaig anstehende Ausführung eines Commitprozesses informiert und kann andernfalls über die entsprechenden Kommunikationskanäle direkt angesprochen oder identifiziert werden, indem eine entsprechende Anfrage an die Modul- bzw. Component-Owner bzw. eine dedizierte Committer-Mailing Liste gesendet wird.

Die Commit-Aktivität wird häufiger auf Quellcodesegmente, respektive Subsysteme angewandt, welche wenig Interdependenzen mit dem Kern des Systems bzw. den Kernfunktionalitäten aufweisen und somit ein geringes Risiko für eine mögliche Destabilisierung darstellen. Der Reviewprozess wird daher für Module mit starken Abhängigkeiten wesentlich ausgeprägter durchgeführt und z.B. im Kontext des Mozilla-Projekts durch explizit definierte und stark formalisierte Reviewprozesse durch eine dedizierte Qualitätssicherungsgruppe ergänzt, welche den hohen Anforderungen an diese Software-Module Rechnung tragen (vgl. [Dietze03]).

Die Aktivität eines Commits wird häufig durch eine Nachricht über die Entwickler-Mailing Liste begleitet, welche durch den Committer initiiert wird und über die Details des Commit informiert. Dies stellt für viele Entwickler den Ausgangspunkt für die Aktualisierung ihres lokal gespeicherten Quellcodes dar. Weiterhin konnten verschiedene operationale Managementartefakte identifiziert werden, welche zur Dokumentation der Commit-Historie von Quellcodeverzeichnissen dienen, wie z.B. die sogenannten Status-Files im Apache-Kontext (vgl. [Dietze03] und [Dietze03c]). Zudem wird der Status des Change Requests im Zuge eines Commits entsprechend geändert, um interessierten Akteuren den Status der Patchentwicklung zu signalisieren.

Falls gerade im Zuge eines neuen Release-Prozesses ein Feature- bzw. Code Freeze auf das entsprechende Quellcodearchiv ausgeführt wird (vgl. *5.2.1 Software-Release*) ist eine Integration von Patches i.d.R. nur durch einen speziell dazu privilegierten Committer ausführbar. In jedem Fall kann dies durch den Release Manager ausgeführt werden und wird z.B. im Mozilla-Projekt durch die sogenannten Driver ergänzt, welche Patches in dieser Phase des Projektes evaluieren und integrieren (vgl. [Dietze03]).

Pre-Release Software-Test
Die Aktivitäten des Pre-Release Software-Tests werden in [Dietze03c] als expliziter Teilprozess dargestellt, schließen sich direkt an die Commit-Aktivität an und werden durch alle Nutzer des aktuellen Entwicklungsquellcodes implizit und z.T. explizit ausgeführt. Dies beinhaltet alle Software-Tests, welche vor einem offiziellen Release bzw. Testrelease und nach der Integration in den Entwicklungsquellcode ausgeführt werden.

5.2 Managementprozesse

In diesem Kapitel werden die zentral und primär durch die Maintainer ausgeführten Managementprozesse dargestellt, welche lediglich eine unterstützende, koordinierende und sekundäre Funktion in Bezug auf die Entwicklungsprozesse ausüben. Die Managementprozesse beinhalten verschiedene Aufgaben zur Unterstützung, Kontrolle und Steuerung der dezentralen Aktivitäten durch die Maintainer der OSS-Projekte.

Verantwortlich für die Durchführung der Managementaktivitäten sind primär die Maintainer (Komitee oder Einzelperson), die zum Teil durch die Core Developer unterstützt werden. Diese supplementären Management-Rollen wurden z.B. in Form der Modul Maintainer, der bereits erwähnten Driver im Releaseprozess oder des Release Managers identifiziert. Falls die Maintenance-Aufgaben durch ein Komitee ausgeübt werden, werden i.d.R. entsprechende demokratische Abstimmungsprozesse zur Entscheidungsfindung und Konsensbildung angewendet. Die primären Maintainer eines Projekts repräsentieren in allen analysierten Projekten auch die ursprünglichen Initiatoren des Projekts, weswegen die initialen Entwicklungsprozesse ebenfalls in diesem Kontext dargestellt werden.

Anwendungsfallsicht auf die Managementprozesse
Die folgende Grafik modelliert die Managementprozesse auf einem sehr hohen Abstraktionsniveau, wobei eine detaillierte Darstellung der einzelnen Teilprozesse in [Dietze03c] enthalten ist:

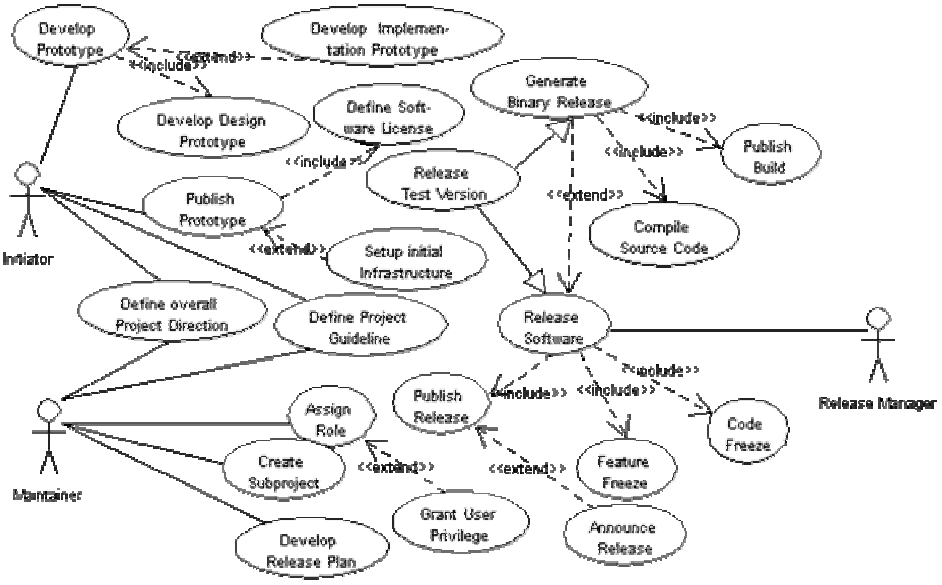

Abb. 5.9: Anwendungsfallsicht der Managementprozesse

Da die Veröffentlichung eines Software-Release zu den wichtigsten Managementprozessen zählt, wird dieser Teilprozess im folgenden exemplarisch vorgestellt.

5.2.1 Software-Release

Der Prozess des Software-Release wird durch die dedizierte Rolle des Release Managers ausgeführt, welcher u.U. identisch mit einem der Maintainer des Projekts ist und die operative Ausführung eines Releaseprozesses durchführt und steuert (vgl. [Erenkrantz03][72]).

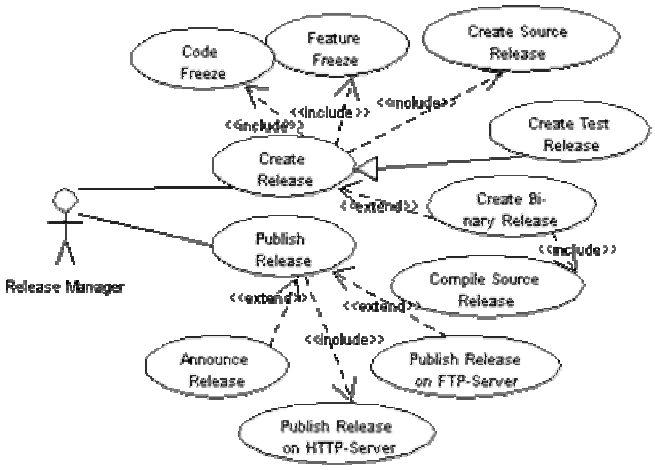

Abb. 5.10: Anwendungsfallsicht des Prozesses *Release Software*

[72] „The release manager has the responsibility for conducting the release.“ [Erenkrantz03]

Im Mittelpunkt eines Releaseprozesses steht die Generierung und Veröffentlichung des Artefakts Source-Release, wobei diese Teilprozesse durch die optionale Veröffentlichung von binären Versionen der Software, respektive von Artefakten des Typs Binary-Release oder von Test-Releases (Alpha- oder Beta-Releases) ergänzt werden können. Daher können Software-Releases in Alpha-, Beta- und sogenannte General Availability Releases (GA-Release) eingeteilt werden, welche in diesem Abschnitt und in [Dietze03c] genauer dargestellt werden. Die sogenannten Nightly Builds, eine spezielle Ausprägung von binären Test-Releases, wurden nicht in das Modell integriert, da sie lediglich eine sepzifische Form von Alpha-Builds repräsentieren und in dieser speziellen Form auch nicht in allen Projekten identifiziert werden konnten (vgl. [Dietze03]). Da sich der primäre Ablauf der Aktivitäten für die Generierung von Test-Releases von dem der Generierung eines offiziellen Release nur marginal unterscheidet, werden diese Teilprozesse in einer gesamtheitlichen Aktivitätssicht dargestellt, welche zudem die Aktivitäten zu deren Veröffentlichung enthält:

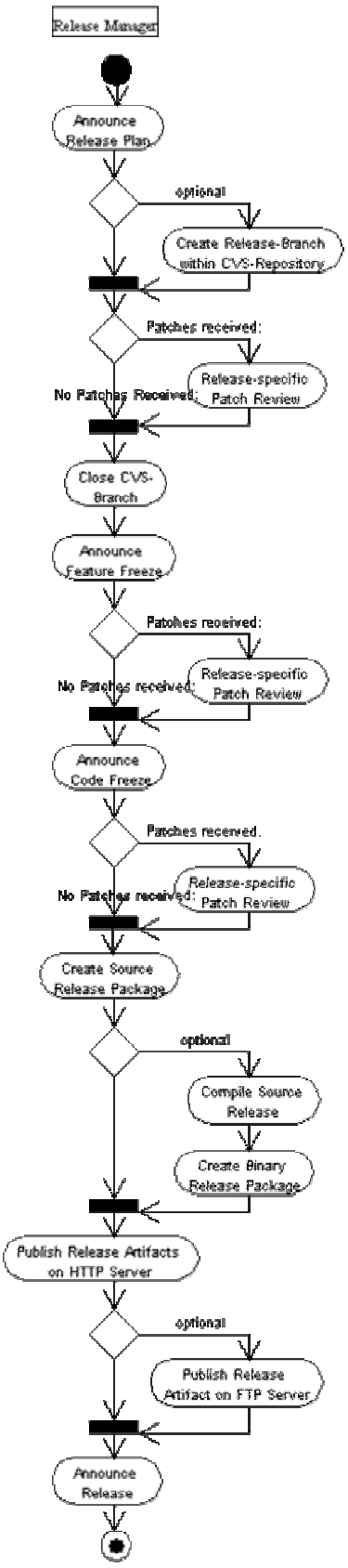

Abb. 5.11: Aktivitätssicht der Teilprozesse zur Generierung und Veröffentlichung eines Software-Release

Nachdem die optionale Aktivität der Generierung eines CVS-Branches durchgeführt wurde, wird das geplante Release über die entsprechenden Kommunikationskanäle und evtl. die Website des Projekts angekündigt. Dies umfasst z.B. den Verweis auf einen evtl. formulierten Release-Plan bzw. die Kommunikation der Inhalte, um dadurch die Entwickler über den Zeitplan der einzelnen Releasephasen und evtl. noch zu integrierende Patches zu informieren. Dies führt häufig dazu, dass eine überdurchschnittlich große Menge von Patches durch die Entwickler übermittelt wird, da diese eine Integration möglichst in das nächste Release forcieren möchten. Dies hat i.d.R. besondere Review-Aktivitäten zur Folge, welche im folgenden noch explizit dargestellt werden und sich von dem konventionellen, bereits dargestellten Patch Review-Prozessen unterscheiden. Anschließend werden die einzelnen Phasen der Schließung des Quellcodes, respektive der Feature- und Code Freeze durchgeführt, welche im folgenden noch explizit beschrieben werden.

5.2.1.1 Branching

Die Aktivität des Branchings beschreibt die Erstellung eines speziellen Zweigs im CVS-Repository der in diesem Kontext ausschließlich der Durchführung eines Release-Prozesses dient (vgl. auch [Erenkrantz03][73]). Dies ermöglicht z.B. die Schließung des releasespezifischen Quellcode-Archivs im Rahmen eines Release-Prozesses während parallel in einem separaten Zweig (Entwicklungszweig) die Entwicklung und Patchintegration wie gewohnt fortgeführt wird. Diese Vorgehensweise konnte bei verschiedenen Projekten identifiziert werden (vgl. [Dietze03]). Im Mozilla-Projekt werden z.B. Test-Releases (Alpha, Beta) direkt aus dem Entwicklungsquellcode generiert und für die Generierung eines GA-Release der getestete Entwicklungsquellcode in einen separaten, stabilen Branch integriert, während die Entwicklung parallel im Entwicklungs-Branch fortgeführt wird.

5.2.1.2 Patch Review im Kontext eines Release Prozesses

Patches, welche im Rahmen des Releaseprozesses übermittelt werden, werden in dieser Phase i.d.R. einem speziellen Review unterzogen. Um eine Destabilisierung des Quellcodes in der Releasephase zu vermeiden, sollen nur zwingend notwendige Patches in das aktuelle Release integriert werden. Dieser spezielle Review-Prozess wird i.d.R. nur durch dedizierte Rollen wie z.B. den Release-Manager, den Maintainer oder ähnlich privilegierte Akteure ausgeführt, um die Qualitätssicherung des zu veröffentlichenden Quellcodes zu gewährleisten. Dieser erweiterte Review-Prozess kann den Commit des Patches in den Entwicklungsquellcode, die Integration des Patches in den aktuell zu veröffentlichenden Quellcode oder die vollständige Verwerfung des Patches zur Folge haben. Dieser spezielle Review wird typischerweise für alle Patches im Kontext des Release Prozesses durchgeführt, wobei der Grad der Restriktivität der Evaluierung stark von der jeweiligen Phase des Release Prozesses abhängt.

[73] „Since open source projects often support concurrent development, there may be parallel branches of development.“ [Erenkrantz03]

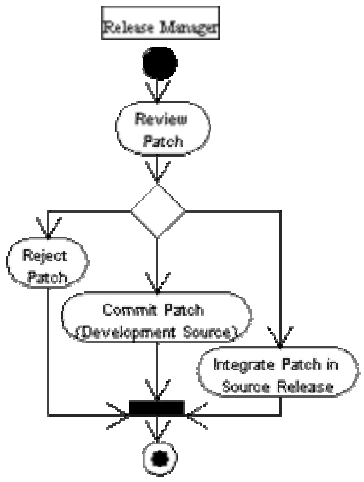

Abb. 5.12: Aktivitätssicht der spezifischen Review-Aktivitäten

5.2.1.3 Feature – und Code Freeze

Um den zu veröffentlichenden Quellcode zu stabilisieren und letzte Softwarefehler zu beseitigen, wird der Quellcode im Rahmen des Releaseprozesses in allen Projekten für einen bestimmten Zeitraum geschlossen, wobei zwischen den Varianten Feature Freeze und Code Freeze unterschieden werden kann, die sich im Grad ihrer Restriktivität unterscheiden. In diesen Phasen werden Patches nur noch durch Release Manager oder spezielle Projekt Management Akteure, wie z.B. die sogenannten Driver im Mozilla-Projekt, integriert, welche die Patches einer eingehenden Evaluierung unterziehen (vgl. [Dietze03]).

Feature Freeze
Im Rahmen des Feature Freeze werden keine neuen Funktionalitäten sondern nur noch Bug Fixes in den zu veröffentlichenden Quellcode integriert, um das Risiko einer Destabilisierung zu minimieren. Die sogenannten Enhancement-Patches können nichtsdestotrotz durch die Committer auf traditionelle Weise in den Entwicklungsquellcode integriert werden.

Code Freeze
Der Code Freeze verfolgt eine ähnliche Zielsetzung, wird aber noch restriktiver interpretiert. Dies bedeutet, dass selbst Bug Fixes nur noch integriert werden, wenn dies für die Durchführung des Release als absolut notwendig eingeschätzt wird.

Die hier beschriebenen Phasen konnten in allen Projekten identifiziert werden, auch wenn deren Bezeichnung, Handhabung und Interpretation bezüglich des Grades der Restriktivität oder ihrer Dauer stark differieren (vgl. [Dietze03]).

5.2.1.4 Erstellung und Release von Test-Releases

Neben den GA-Releases wurden in allen Projekten verschiedene Test-Releases (Alpha- und Beta-Releases) identifiziert, welche im Vorfeld der Veröffentlichung eines offiziellen Software-Release zur Verfügung gestellt werden, um im Rahmen der Testphase eine größtmögliche Anwendergemeinde zu adressieren. Es werden häufig verschiedene Termini für Test-Releases verwendet oder nur eine Ausprägung, beispielsweise die Beta-Releases, instrumentalisiert, wie dies bei der Linux Kernel Entwicklung der Fall ist (vgl. [Dietze03]). Ein Alpha-Release kann als relativ instabil und ungetestet betrachtet werden und richtet sich daher vornehmlich an interessierte Tester, Entwickler und sehr versierte Anwender. Das Beta-Release basiert auf dem Alpha-Release, wird in einer

fortgeschrittenen Projektphase veröffentlicht und ist somit als stabiler aber noch nicht hinreichend getestet anzusehen. Neben diesen Test-Releases existieren z.T. noch weitere Ausprägungen, wie z.B. die Nightly- oder Demo-Releases im Rahmen des Apache Projekts, welche sich aber lediglich bezüglich des Grads der Stabilität und Reife und bezüglich der Frequenz der Veröffentlichung von den beschriebenen Alpha- und Beta-Releases unterscheiden. Ergänzt werden diese Test-Releases durch kompilierte Binärversionen, den sogenannten Alpha- und Beta-Builds.

Durch die Generierung von Test-Releases wird ein größerer Personenkreis zur Durchführung von Software-Tests erreicht, da der Entwicklungsquellcode im CVS-Repository in Form eines Release über den HTTP- bzw. FTP-Server des Projekts veröffentlicht wird und somit eine größere Zielgruppe erreicht wird, die sich nicht auf die Entwickler beschränkt, welche den Quellcode aus dem CVS-Archiv verwenden. Die Veröffentlichung von Test-Releases werden typischerweise über die etablierten Kommunikationskanäle bekanntgegeben, um eine schnelle Aufnahme und Durchführung der impliziten Testprozesse zu ermöglichen.

Praktizierungsalternativen
Im Rahmen der Veröffentlichung von GA- und Test-Releases kommen verschiedene Praktizierungsalternativen zum Einsatz. Eine mögliche Option ist die Durchführung des dargestellten Releaseprozesses explizit für alle einzelnen Releases (Alpha-, Beta- und GA-Release) in mehreren Iterationen.

Eine andere und sehr gebräuchliche Alternative ist die einmalige Durchführung des modellierten Releaseprozesses über einen längeren Zeitraum und die Veröffentlichung der Alpha- und Beta-Releases als impliziter Bestandteil einer Iteration des dargestellten Prozesses. Dies beinhaltet die Veröffentlichung des Alpha-Release vor der Phase des Feature Freeze und die Veröffentlichung des Beta-Release vor der Phase des Code Freeze, während das GA-Release zum Abschluß des Releaseprozesses publiziert wird. Diese Variante wurde z.B. im Rahmen des Mozilla-Projekts identifiziert (vgl. [Eich02], [Dietze03]). Diese sukzessive Stabilisierung der Releases bzw. des Quellcodes basierend auf den durchgeführten Freeze-Phasen und der Veröffentlichung der Test-Releases führt zu einer relativen Stabilisierung des GA-Release bereits zum Zeitpunkt der Veröffentlichung.

Prozess der Nightly Builds
Aus Gründen der Vollständigkeit wird an dieser Stelle der Prozess der Nightly Builds vorgestellt, der sowohl im Mozilla- als auch im Apache Jakarta-Kontext identifiziert weren konnte (vgl. [Dietze03]). Dieser Prozess basiert auf einem definierten Workflow zur täglichen Bereitstellung von Test-Builds, den sogenannten Nightly Builds. Diese werden täglich durch eine dedizierte Rolle bzw. den Release Manager für verschiedene Plattformen kompiliert und auf der Projekt Website bereitgestellt, um mit dem aktuellen Entwicklungsquellcode täglich eine möglichst große Anzahl von Testern erreichen zu können und die Kompilierbarkeit des Entwicklungsquellcodes sicherzustellen. Falls bereits bei der Kompilation Fehler auftreten, werden die entsprechenden Patches entfernt oder die Fehlerbehebung mit den Autoren koordiniert. Dies führt dazu, dass der Entwicklungspfad im CVS-Repository im Idealfall stets über kompilierfähigen Entwicklungsquellcode verfügt. Nach der Publikation der Nightly Builds werden diese durch einen Teil der Community getestet und die identifizierten Fehler und Änderungsanforderungen als Change Request entsprechend dem beschriebenen Teilprozess kommuniziert.

5.2.1.5 Veröffentlichung eines GA-Release

Nach der Durchführung der Testprozesse wird das offizielle GA-Release durch den Release Manager generiert und u.U. entsprechende Binärversionen des Release bereitgestellt. Dies erfolgt, indem ein Package in Form einer gepackten Datei mit allen notwendigen Quellcode-Artefakten bereitgestellt wird, diese kompiliert werden und alle entsprechenden Dokumentationsartefakte aktualisiert werden. Anschließend werden die so generierten Artefakte auf der Website des Projekts bereitgestellt und evtl. auch über einen FTP-Server zur Verfügung gestellt. Ergänzt wird diese Aktivität durch die Bekanntgabe des neuen Release über die Kommunikationskanäle und in entsprechenden Foren oder externen Medien.

Nach einem offiziellen Release können Änderung am aktuellen Release nur durch ein neues Release oder durch die explizite Veröffentlichung von Patches über die Website vorgenommen werden.

5.3 Infrastrukturelle Prozesse

Die infrastrukturellen Prozesse werden primär durch die zentralen Management-Instanzen des Projekts wahrgenommen und sind von sekundärer aber trotzdem fundamentaler Bedeutung für den Entwicklungsprozess der OSS, da sie die Entwicklungsprozesse durch die Bereitstellung einer adäquaten Infrastruktur unterstützen und ermöglichen sollen. Aufgrund der sekundären Bedeutung und der geringen Recherchierbarkeit dieser internen, durch die zentral agierenden Maintainer ausgeführten Aktivitäten, konnten diese Teilprozesse im Rahmen dieser Arbeit nur sehr rudimentär analysiert werden. Zudem basieren diese Prozesse meist auf sehr informell und individuell verschieden praktizierten Aktivitäten, welche nur auf einer sehr hohen Abstraktionsebene verallgemeinert und formal beschrieben werden können.

5.3.1 Grundlegende Charakteristika der Infrastruktur

Um den heterogenen Prozess und die individuellen Aktivitäten der einzelnen Entwickler zu unterstützen und den Aufwand zur Wartung der Infrastruktur zu mininimieren, werden meist nur weitgehend autark funktionsfähige Werkzeuge implementiert, welche keiner umfangreichen, ständigen Wartung bedürfen. So generieren die verwendeten Bug Tracking Systeme z.B. nötige User Accounts i.d.R. selbständig nach der Anmeldung durch den Nutzer und ermöglichen anschließend eine individuell durchgeführte Verwaltung der Change Requests ohne zentrale Kontrolle. Dies trifft ebenso auf die Infrastruktur für das Konfigurationsmanagement (CVS) zu, welche die Rahmenbedingungen für die dezentralen, kollaborativen und nicht zentral gesteuerten Entwicklungsprozesse schafft.

Die zentralen, infrastrukturellen Ressourcen werden durch eine zentrale Maintenance-Instanz verwaltet, die häufig von den selben Akteuren ausgeübt werden, die auch für die Managementprozesse verantwortlich zeichnen.

Weiterhin konnte die Tendenz identifiziert werden, dass intern benötigte Werkzeuge z.T. auch eigenständig durch die Community des Projekts entwickelt werden, was u.U. auch zur Generierung eines Teilprojekts oder eines eigenständigen OSS-Projekts führen kann, wie dies bei Bugzilla im Mozilla-Projekt oder dem LXR-Werkzeug [LXR03][67] im Linux-Kontext identifiziert wurde (vgl. [Dietze03]).

Ein weiteres wichtiges Merkmal der identifizierten Infrastruktur ist deren Internetfähigkeit und Browserbasiertheit, da i.d.R. alle identifizierten Kollaborations- oder Kommunikationswerkzeuge den Datentransfer über Protokolle des Internets ermöglichen. Auch die Kollaborationswerkzeuge, wie CVS oder Bugzilla, verfügen über webbasierte Benutzerschnittstellen und lassen sich über standardisierte Web-Browser benutzen, womit der dezentralen Verteilung und den heterogenen Systemvoraussetzungen (Betriebssystem, Hardwareplattform) auf Seiten der dezentralen Akteure Rechnung getragen wird.

Außerdem konnte eine starke Tendenz zur wechselseitigen Beeinflussung zwischen den praktizierten Entwicklungsprozessen und den involvierten Werkzeugen festgestellt werden. Dies bedeutet, dass zwar die instrumentalisierten Werkzeuge die Charakteristika des dezentralen und heterogenen Entwicklungsprozesses im OS-Kontext unterstützen und dahingehend ausgewählt und implementiert wurden, aber auch die verwendeten Werkzeuge, respektive die Bug Tracking- oder Konfigurationsmanagementsysteme, den Entwicklungsprozess entscheidend determinieren und durch ihre systemspezifischen Eigenschaften reglementieren.

5.3.2 Übersicht über die infrastrukturellen Prozesse

In der folgenden Grafik wurden die infrastrukturellen Prozesse in der Use Case-Sicht dargestellt:

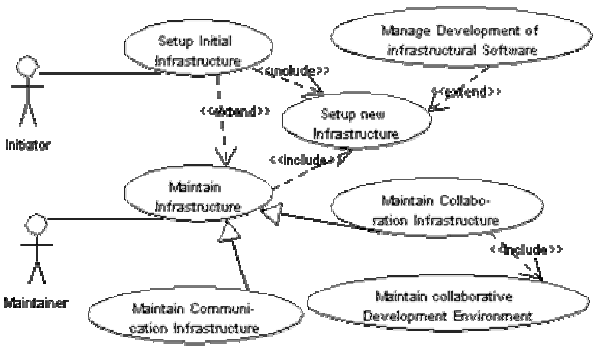

Abb. 5.13: Use Case Sicht der Infrastrukturellen Prozesse

Im Rahmen der infrastrukturellen Prozesse (vgl. *2.2.1 Prozesse*) kann die initiale Etablierung einer Infrastruktur von der langfristigen Verwaltung und Pflege der Infrastruktur unterschieden werden, wobei zur Verringerung der Komplexität im Rahmen dieser Arbeit lediglich die allgemeinen infrastrukturellen Prozesse betrachtet werden und auf diese Differenzierung verzichtet wird. Weiterhin werden entsprechend des Metamodells die Verwaltung der Kommunikations- von der Kollaborationsinfrastruktur als Spezialisierungen der allgemeinen infrastrukturellen Maintenance unterschieden. Eine detailliertere Darstellung dieser Prozesse ist in [Dietze03c] enthalten und wird im Rahmen dieser Arbeit nicht vorgenommen.

5.4 Überblick über die identfizierten Prozesse

In der folgenden Grafik wurden die bereits dargestellten Artefakte und Rollen in einer Matrix den identifizierten Prozessen bzw. Teilprozessen gegenübergestellt. Dadurch entsteht eine Übersicht über alle zentralen Prozesse eines OSS-Projekts, die dafür typischerweise verantwortlichen Rollen und die in diese Prozesse involvierten Artefakte. Dabei wurde in der Matrix unterschieden, ob ein Artefakt im Rahmen dieses Prozesses lediglich verwendet, oder ob das Artefakt in diesem Prozess typischerweise auch modifiziert wird. Die Modifikation beinhaltet die Erstellung, Veränderung oder Löschung des jeweiligen Artefakts.

Tab. 5.1: Beziehungen der identifizierten (Teil-)Prozesse, Rollen und Artefakte

		Rollen								Technologische Artefakte											Manage mentart.		
		User	Contributor	Developer	Reviewer	Committer	Release Manager	Maintainer	Initiator	Bug Report	Enhancement Request	Patch	Dokumentationsartefakt	Developement Source	Release Source File	Development Build	Release Build	Alpha Release	Beta Release	GA Release	Softwarelizenzmodell	Projektweite Guidelines	Operation. Managementart.
Entwicklungsprozesse	Software Deployment:																						
	Deployment von Software Releases	x											u		u		u	u	u	u	u	u	
	Deploym. von Entwicklungsquellcode			x										u		u					u	u	
	Software Review und Test:																						
	Pre-Commit Review			x	x	x						u										u	
	Pre-Release Review			x	x	x	x							u		u						u	
	Post-Release Test	x													u		u	u	u	u	u	u	
	Anwender- und Entwicklersupport:																						
	Request Support		x	x									u									u	
	Provide Support		x	x									u									u	
	Beitrag von Change Requests		x							m	m											u	
	Kollaborativer Request Review		x	x	x					m	m											u	
	Patchentwicklung:																						
	Lokale Patchentwicklung			x						m	m	m									u	u	
	Commit					x				m	m	u		m								u	
	Entwicklg. v. Dokumentationsartef.:																						
	Lokale Dokumentationsentwicklung		x										m	u	u	u	u	u	u	u		u	
	Review und Publikation der Dokum.						x	x					u									u	
Managementprozesse	Initiale Prototypentwicklung								x					m	m		m	m			m	m	m
	Initialer Prototyprelease								x					u	u		u	u			u	u	u
	Definition der strateg. Ausrichtung							x			m											m	m
	Etablierung projektweiter Richtlinien							x														m	
	Releaseplanung							x						u									m
	Software Release:																						
	Branching						x	x						u								u	
	Feature- und Code Freeze						x							u								u	m
	Spezifischer Patch Review / Commit					x	x	x				u		m								u	
	Publikation von Test Releases						x							u	m		m	m	m		u	u	m
	Publikation von GA-Releases						x										u	u	u	m	u	u	m
	Zuweisung von Rollen und Privilegien							x														u	
	Generierung neuer Teilprojekte							x														u	
Infrastruktur. Proz.	Initiale Infrastrukturentwicklung								x													m	
	Mainten. d. Kommunikationsinfrastr.:																						
	Website Maintenance							x					u									u	
	Mainten. der Kommunikationskanäle							x														u	
	Mainten. der Kollaborationsinfrastr.:																						
	Mainten. der Quellcodeverwaltung							x						m								u	
	Mainten. des Bug Tracking Systems							x		m	m											u	

x: Rolle führt typischerweise entsprechenden (Teil-)Prozess aus
u: usage (Artefakt wird verwendet)
m: modification (Artefakt wird verwendet und kann in diesem Prozess modifiziert werden)

5.5 Validierung des deskriptiven Prozessmodells

Um die inhaltliche Korrektheit und die Praxisrelevanz des dargestellten Prozessmodells zu validieren, wurde versucht, zentrale Aussagen des deskriptiven Modells an einem exemplarischen, real existierenden OSS-Entwicklungsprojekt zu verifizieren. Das allgemeine, deskriptive Prozessmodell wurde dazu im Kontext einer real existierenden Instanz dieses Modells betrachtet.
Ziel ist es dabei, an einer realen Ausprägung eines OSS-Entwicklungsprojekts elementare Aussagen des Prozessmodells zu verifizieren und etwaige Informationsverluste oder fehlerhafte Thesen zu identifizieren, die im Prozess der Generalisierung der Prozesse und Entitäten der verschiedenen Fallstudien entstanden sein können. Die

gesamte Validierung aller Aspekte des OSS-Ansatzes und eine konkrete Darstellung der instanziierten Prozesse des OSS-Projekts wurden in [Dietze03c] thematisiert, da an dieser Stelle aus Platzgründen lediglich die Ergebnisse des Validierungsprozesses dargestellt werden können.

5.5.1 PhOSCo

Als exemplarische OSS-Projektinstanz wurde die Software PhOSCo[74] und das damit verbundene Entwicklungsprojekt ausgewählt. Der Autor und das Fraunhofer ISST sind über das Projekt Telematikplattform für Medizinische Forschungsnetze (TMF) direkt in das Projekt involviert, wodurch viele Aspekte des Entwicklungsmodells bereits im Kenntnisbereich des Autors liegen und die weitere Informationsbeschaffung wesentlich vereinfacht wird.

Das Akronym PhOSCo steht für *Pharma Open Source Community* und ist ein eingetragenes Warenzeichen der Firma Guillemot Design Ltd. (vgl. [PhOSCoDe03][75]). PhOSCo bezeichnet eine Software zur Unterstützung klinischer Studien, respektive der elektronischen und verteilten Datenerfassung in diesem Kontext [PhOSCoCom03][76]. Diese Systeme werden auch als EDC- (Electronic Data Capture) oder RDE-Systeme (Remote Data Entry) bezeichnet und ermöglichen die verteilte Dateneingabe von Probanden und die gesamte Datenverwaltung und -auswertung im Rahmen medizinischer Studien.

Einschränkungen der Vergleichbarkeit
Das PhOSCo-Projekt weist verschiedene spezifische Merkmale auf, welche das Projekt und die entwickelte Software z.T. von typischen OSS-Projekten unterscheiden und nur eine eingeschränkte Verifizierbarkeit des generalisierten OSSD-Modells vermuten lassen:

- Spezifische Anwendungsdomäne
- Restriktives (GNU GPL-inkompatibles) Lizenzmodell
- Gesetzliche Rahmenbedingungen und Reglementierungen der Anwendungsdomäne
- Projektstatus

Diese Aspekte wurden in [Dietze03c] detailliert erläutert.

5.5.2 Fazit der Validierung

In diesem Abschnitt werden die wesentlichen Erkenntnisse der Validierung des deskriptiven Prozessmodells anhand des realen OSS-Projekts PhOSCo zusammengefasst.

5.5.2.1 Validierungsergebnisse

Nicht alle Entitäten des generalisierten Prozessmodells konnten umfassend und eindeutig verifiziert werden, wobei aber alle im PhOSCo-Projekt identifizierten Prozesse, Rollen, Artefakte und Werkzeuge den Spezifikationen des identifizierten OSSD-Prozessmodells weitgehend entsprechen und mit diesen nicht konfligieren. Dies bedeutet, dass bisher kein Aspekt des PhOSCo-Projekts im direkten Widerspruch zum eruierten OSS-Prozessmodell steht, aber aufgrund des frühen Status der Projektevolution und der in [Dietze03c] dargestellten Einschränkungen noch nicht alle Prozesse und Entitäten auf der detaillierten Abstraktionsebene des deskriptiven Prozessmodells ausgeprägt sind. Es ist mit hoher Wahrscheinlichkeit davon auszugehen, dass die zentralen Prozesse und Entitäten des generalisierten Prozessmodells im weiteren Projektverlauf des PhOSCo-Projekts ebenfalls instanziiert werden.

[74] URL: http://www.phosco.com

[75] URL: http://www.phosco.de/linziweb/index.php?ml=l; Abfrage: 12.03.2003

[76] „Briefly, PhOSCo is an Open Source package for Clinical Trials Electronic Data Capture (EDC).“; URL: http://www.phosco.com/info.html; Abfrage: 12.03.2003

Tab. 5.2: Zusammenfassung der Validierung des deskriptiven Prozessmodells

		Validierungsergebnis			
		Verifiziert	Indirekt bzw. teilweise verifiziert	Zukünftige Verifizierbarkeit prognostiziert	Nicht verifizierbar
Rollen	User	x			
	Contributor	x			
	Developer	x			
	Reviewer	x			
	Committer		x		
	Release Manager	x			
	Maintainer	x			
Werkzeuge	HTTP Server	x			
	Mailing Listen			x	
	Newsgroups				x
	IRC				x
	CVS			x	
	Bug Tracking System			x	
Artefakte	Software Lizenz	x			
	Guidelines		x		
	Change Request		x	x	
	Patch		x	x	
	Software Release		x	x	
	Dokumentationsartefakt	x			
Prozesse	OSS Deployment		x		
	Change Request Contribution		x	x	
	Patch Development		x	x	
	Support		x		
	Documentation Development		x	x	
	Initiale Prototypentwicklung	x			
	Zuordnung von Rollen			x	
	Entwicklung von Teilprojekten			x	
	Software Release		x		
	Infrastrukturelle Prozesse		x		

In der Abbildung *Tab. 5.2* wurden die primären Entitäten des OSS-Prozessmodells aufgeführt und ihre Verifizierbarkeit anhand des PhOSCo-Projekts bewertet. Die Spalte *Indirekt bzw. teilweise verifiziert* wurde für Prozesse oder Entitäten gewählt, die zwar im PhOSCo-Projekt identifiziert werden konnten, aber nur mit Einschränkungen allen im deskriptiven Prozessmodell identifizierten Charakteristika entsprechen. Beispielsweise werden Change Requests zwar im PhOSCo-Projekt erstellt und kommuniziert, da aber momentan noch kein Bug Tracking System zu deren strukturierter Beschreibung und Verwaltung verwendet wird, existiert keine erschöpfende Deckungsgleichheit mit dem generalisierten OSSD-Ansatz. Da aber die zukünftige Implementation eines entsprechenden Systems bereits durch den Maintainer des Projekts angekündigt wurde, wurde in diesem Beispiel zudem die Spalte *Zukünftige Verifizierbarkeit prognostiziert* ausgewählt.

Elementare Aspekte, wie z.B. die Etablierung und Verwendung der identifizierten Software Infrastruktur, bestehend aus einem CVS-Repository und einem Bug Tracking System, sind bereits explizit in Planung. Es ist zu erwarten, dass dies auch die Evolution der primären Entwicklungsprozesse, respektive der Patchentwicklung und des Beitrags von Change Requests, und somit auch der dabei erstellten Artefakte entsprechend dem identifizierten Prozessmodell nach sich zieht. Daher ist zukünftig von einem hohen Grad der Kongruenz zu dem generalisierten Prozessmodell auszugehen und eine sukzessive Angleichung an die zentralen Elemente des Prozessmodells zu erwarten. Somit konnten die zentralen Aspekte des definierten OSS-Prozessmodells direkt verifiziert oder eine sukzessive Annäherung an dieses Modell zumindest prognostiziert werden.

5.5.2.2 Implikationen für diese Arbeit

Die Identifikation und Beschreibung des OSSD-Modells und dessen Validierung hat zu verschiedenen Erkenntnissen geführt, die für diese Arbeit von großer Bedeutung sind und einige der zentralen Prämissen des Ansatzes dieser Arbeit bestätigen:

- Notwendigkeit eines allgemeinen OSSD-Prozessmodells
- Notwendigkeit einer adäquaten Software-Infrastruktur
- Optimierungspotentiale des praktizierten OSS-Entwicklungsmodells

Trotz der in [Dietze03c] detailliert dargestellten, schlechten Ausgangsbedingungen für eine Verifikation des generalisierten Prozessmodells anhand des PhOSCo-Projekts, konnte festgestellt werden, dass auch in diesem Projekt die zentralen Aspekte des generalisierten Prozessmodells bestätigt werden konnten. Wie in den meisten OSS-Projekten, werden die Prozesse, Artefakte, Infrastrukturen und Rollen schrittweise den wachsenden Anforderungen angepasst. Diese Anforderungen werden sich höchstwahrscheinlich im Verlaufe des Projekts sukzessive den allgemeinen Anforderungen großer OSS-Projekte annähern, wodurch eine Evolution der zentralen Aspekte des Entwicklungsprojekts PhOSCo in Richtung des generalisierten Prozessmodells stattfindet.

Aufgrund der lediglich sukzessiven Annäherung an dieses Modell und den damit verbundenen, langwierigen Evolutionsprozessen werden zu Beginn des Projekts häufig nur inadäquate und nicht optimale Infrastrukturen zur Verfügung gestellt, welche eine bestmögliche Entwicklung der Community behindern und die Entwicklungsprozesse nicht probat ermöglichen können. Dies repräsentiert die typische Entwicklung von OSS-Projekten, die somit erst nach einer langen Phase der Projektevolution notwendige Infrastrukturen bereitstellen und somit geeignete Entwicklungsprozesse ermöglichen. Dieser verzögerte Prozess läßt sich durch die Anwendung des bereits beschriebenen OSSD-Modells wesentlich beschleunigen, da es die Erfahrungswerte der Evolution von OSS-Projekten bereits weitgehend dokumentiert und formalisiert beschreibt. Dies stellt somit eine mögliche Basis für die Instanziierung neuer OSS-Projekte dar, um bereits direkt nach dem Prozess der initialen Prototypveröffentlichung adäquate, verteilte Entwicklungsprozesse zu ermöglichen. Dies kann zudem durch die Entwicklung einer verbesserten Softwareinfrastruktur wesentlich unterstützt werden, welche die definierten Funktionalitäten bereitstellt und um weitere sinnvolle Funktionen ergänzt werden sollte. Zudem existieren diverse Optimierungspotentiale des OSSD-Modells, die im folgenden Kapitel thematisiert werden.

6 Verbesserungspotentiale und –ansätze des OSSD-Modells

In diesem Kapitel werden zu Beginn einige Implikationen und Tendenzen identifiziert, die basierend auf der vorangegangenen Analyse des OSSD-Modells festgestellt werden können. Anschließend werden verschiedene Ansätze zur Erweiterung und Verbesserung des Modells vorgestellt und die damit verbundenen Zielstellungen und Anforderungen erläutert, die bei der Verbesserung des OSSD-Modells berücksichtigt werden sollten.

6.1 Implikationen und Erkenntnisse aus der empirischen Analyse

In diesem Abschnitt werden verschiedene Implikationen und Beobachtungen aufgeführt, die bei der Durchführung der Fallstudien der OSS-Projekte, während des anschließenden Generalisierungsprozesses und im Kontext der Validierung des Modells projektübergreifend festgestellt wurden und daher zu den folgenden Aussagen über OSS-Projekte verallgemeinert wurden:

- Tendenz zu redundanten Aktivitäten aufgrund mangelnder Prozesstransparenz
- Projektdiversifikation – Segmentierung in Subprojekte
- Notwendigkeit von Prozessen zur Entscheidungsfindung bzw. Konfliktbewältigung
- Entwicklerfokus auf Implementation – Tendenz zur Vernachlässigung peripherer Prozesse (Support, Dokumentation)
- Häufige Releasezyklen
- OSSD-spezifische Anforderungen an die Softwarequalität und –architektur
- Tendenz zum *Boot Strapping* (Verwendung von OSS als Infrastruktur)

Diese Implikationen und Tendenzen konnten im Rahmen der Analyse der OSS-Projekte projektübergreifend beobachtet werden und werden im Anhang *A Implikationen und Tendenzen des OSSD-Ansatzes* detaillierter erläutert. Diese Erkenntnisse dienen im Folgenden der Herleitung verschiedener Optimierungspotentiale des OSSD-Ansatzes und der Entwicklung einer geeigneten Softwareunterstützung. Dies kann neben der allgemeinen Verbesserung des OSSD-Modells auch zu einer Verwendbarkeit dieses Ansatzes in kommerziellen Softwareentwicklungsszenarien führen.

6.2 Potentiale und Risiken des OSSD-Modells

Die in *6.1* und Anhang *A* dargestellten Erkenntnisse und Implikationen repräsentieren die Basis für die Herleitung von Vor- und Nachteilen bzw. Risiken und Chancen des OSSD-Ansatzes. Diese werden in diesem Abschnitt einander gegenübergestellt.

6.2.1 Chancen und Vorteile

Die primären Vorteile des OSSD-Modells ergeben sich vor allem aus der Möglichkeit, eine größtmögliche Anwender- und Entwicklergemeinde zu erreichen und sind daher i.d.R. an die Erreichung einer kritischen Mindestanzahl aktiver Communitymitglieder gebunden:

- Darwinistisches Selektionsprinzip bei der Software-Evolution (vgl. „*Self-Correcting Code"* [Pavlicek00])
- Weitgehend selbstorganisierte Prozesse (vgl. „*Self-Correcting Community"* [Pavlicek00])
- Anwendergerechte Definition der Anforderungen und des Funktionsumfangs
- Schnelle Implementation und Veröffentlichung von Softwareanforderungen

- Objektiver, unabhängiger Peer Review
- Umfangreiche Anwendertests

Die bereits dargestellte Tendenz zu redundanten Aktivitäten umfasst auch die redundante Erstellung von neuen Softwareartefakten und stellt somit auch eine wichtige Voraussetzung für einen Softwareevolutionsansatz im Sinne einer darwinistischen Auswahl des hochwertigsten Quellcodes dar. Indem z.B. verschiedene Patches bezüglich eines Change Requests implementiert und an die Community kommuniziert werden, wird es ermöglicht, dass durch die Community der qualitativ hochwertigere und funktionalere Quellcode selektiert und in den zentralen Entwicklungsquellcode integriert wird (vgl. [Weber00][77]). Dies ermöglicht somit einen Prozess der systematischen Auslese entsprechend den Anforderungen der involvierten Akteure (vgl. *„Self-Correcting Code“* in [Pavlicek00]), der neben der Unterstützung funktionaler und syntaktischer Aspekte auch eine qualitativ hochwertige Architektur unterstützen und hervorbringen kann (vgl. Anhang *A.6 Softwarequalität und -evolution im OSS-Kontext*).

Der hohe Grad der Parallelität und Autonomie aller Prozesse ermöglicht die weitgehende Selbstorganisation der Community bzw. der Entwicklungsprozesse. Von großer Bedeutung sind in diesem Zusammenhang auch die starke Dezentralisierung, die demokratisierten Entscheidungsfindungsprozesse und der meritokratische Charakter der Community.

Indem die Anwender einer Software von passiven Nutzern zu Akteuren im OSSD-Prozessmodell und aktiv in die Entwicklung einbezogen werden, wird es ermöglicht, eine Software zu entwickeln, die in hohem Maß den Anforderungen der Anwender entspricht. In der proprietären Softwareentwicklung herrscht demgegenüber auf Seiten der Entwickler häufig ein Mangel an Kenntnis der Anwendungsdomäne, wodurch eine Diskrepanz zwischen dem entwickelten Softwareprodukt und den tatsächlichen Anforderungen der Anwender entsteht.

Zudem ermöglicht der OSSD-Ansatz die vergleichsweise schnelle Implementation und Veröffentlichung von Änderungen am Quellcode [MoFiHe00][78], sofern eine hinreichende Anzahl an Projektakteuren besteht. Während Patches im OSSD-Modell direkt in den Quellcode integriert werden können und somit sofort öffentlich verfügbar sind, werden in der kommerziellen Softwareentwicklung Verbesserungen am Quellcode i.d.R. nur nach einem vordefinierten Zeitplan bzw. nach einer zeitaufwendigen Testing-Prozedur veröffentlicht [MoFiHe00][79]. Zudem unterstützen die häufigen Software-Releases und der Fokus der involvierten Akteure auf den Prozess der (Patch)-Implementation die schnellstmögliche Entwicklung und Publikation von Softwareänderungen.

Als großer Vorteil der parallelen Test- und Reviewprozesse kann die Tatsache betrachtet werden, dass in diesem Kontext unabhängige Peer Reviews praktiziert werden, was in kommerziellen Projekten nur schwer realisierbar ist und neben den Anwendertests auch den Review des Quellcodes aus Entwicklerperspektive einbezieht [Vixie99][80]. Dies unterstützt auch eine hohe Softwarequalität und steht im Kontrast zu den eingeschränkten Reviewprozessen proprietären Quellcodes, die lediglich durch dessen Entwickler ausgeführt werden [VoCo03][81].

[77] „They will likely produce a number of different potential solutions. It is then possible […] to incorporate the best solution and refine it further.“ [Weber00]

[78] „OSS developments exhibit very rapid responses to customer problems.“ [MoFiHe00]

[79] „In commercial developments, by contrast, patches are generally bundled into new releases, and made available according to some predetermined schedule.“ [MoFiHe00]

[80] „An additional advantage enjoyed by open-source projects is the *‚peer review'* of dozens or hundreds of other programmers looking for bugs by reading the source code rather than just by executing packaged executables.“ [Vixie99]

[81] „In every engineering field, peer review is the cornerstone to furthering ideas, yet software source code is rarely reviewed, objectively, on the level OS code is reviewed.“ [VoCo03]

Neben den Reviewern des Quellcodes stellt eine große Anwendergemeinde aber auch ein großes Potential an Software-Testern dar [FeFi02][82]. Die umfangreichen Anwendertests werden nach [KiLe00][83] den Anforderungen des späteren Einsatzes zudem besser gerecht als statische Regressionstest, die z.T. in proprietären Softwareprojekten angewendet werden.

6.2.2 Risiken

Das OSSD-Prozessmodell birgt verschiedene inhärente Risiken, die bei der zentralisierten Softwareentwicklung in einem proprietären Kontext nicht oder nicht im selben Maße existieren. Diese resultieren z.T. aus den in 6.1 bzw. Anhang A dargestellten Implikationen und Erkenntnissen über den OSSD-Ansatz:

- Ungenügend ausgeführte Teilprozesse (Support, Dokumentation, Code etc.)
- Redundante Aktivitäten (Ineffizienter Ressourceneinsatz)
- Verzögerte Entscheidungsfindungsprozesse
- Hohe Fehlerdichte in neuen Software-Releases
- Forking
- Updates und Services nicht terminierbar
- Keine formelle Qualitätssicherung
- Mangelhafter Softwaresupport (Workflows, Automatisierung)

Mit dem Begriff Forking wird die unerwünschte Aufspaltung des Quellcodes in verschiedene Quellcodepfade bezeichnet. Dies tritt ein, wenn Entwickler eines OSS-Projekts den veröffentlichten Quellcode im Rahmen eines neuen OSS-Projekts verwenden und somit verschiedene Versionen der Software entstehen, die in verschiedenen Projekten verwaltet und weiterentwickelt werden. Dieses Risiko hat sich im Rahmen der Fallstudien nicht bestätigt und kann i.d.R. durch eine probate Maintenance des Projekts und der Community verhindert werden, stellt aber eine in Betracht zu ziehende Unwägbarkeit im OSSD-Ansatz dar [GaLaAr00][84].

Weitere Risiken ergeben sich aus der Tatsache, dass im OSSD-Modell i.d.R. alle Aktivitäten freiwillig und ungeplant durch Akteure ausgeführt werden und somit über die Ausführung von Entwicklungsaktivitäten keinerlei zuverlässige Planung möglich ist. Updates, Patches und neue Releases der Software sind daher nicht terminierbar und Services, wie z.B. die Bereitstellung von Support oder die Entwicklung von Updates können nie in garantierter Form zur Verfügung gestellt werden.

Nach [Koch00] ist einer der Hauptkritikpunkte des OSS-Ansatzes das Fehlen einer formalen Qualitätssicherung (QS). In eingeschränkter Form wird die QS durch den Review der Committer gewährleistet und es gibt auch im OSS-Bereich verschiedene Ansätze zur Etablierung von Methoden zur QS (vgl. [Dietze03]). Trotzdem weist OSS eine vergleichsweise hohe Fehlerdichte bei neuen Softwareversionen auf, die mit der Releasepraxis und den mangelhaften Pre-Release-Tests erklärbar ist und im Rahmen des sukzessiven Softwareverbesserungsprozesses behoben wird.

Als sehr präsentes Manko des OSSD-Ansatzes sei an dieser Stelle erneut auf die mangelhafte Softwareunterstützung des OSSD-Modells hingewiesen, die keine umfassende softwaretechnische Implementierung und Automatisierung der Prozesse ermöglicht.

[82] „Also, the software engineering principles of independent […] testing are very highly evolved to an extremely advanced level within OSS." [FeFi02]

[83] „A large, distributed, heterogeneous test ‚organization' stresses a system much better than a static regression test suite." [KiLe00]

[84] „This fear prevents some individuals and many companies from active participation in open source developments." [GaLaAr00]

6.3 Spezifische Anforderungen kommerzieller Systementwicklung

Die Entwicklung kommerzieller Software basierend auf spezifischen Kundenbedürfnissen unterliegt verschiedenen Anforderungen, die im OSS-Kontext keine oder nur eine untergeordnete Rolle spielen und durch das klassische, nicht optimierte OSSD-Modell nicht oder nur ungenügend gewährleistet werden [WaAb03][85]. Diese Anforderungen umfassen vor allem die folgenden Aspekte, die im Rahmen einer Optimierung des OSSD-Ansatzes berücksichtigt werden sollten:

- Terminierbarkeit des Entwicklungsprozesses
- Planbarkeit von Zeitpunkt und Umfang von Software-Releases
- Entwicklung basierend auf spezifischen Kundenanforderungen
- Garantierte Services (Support, Patches, Dokumentation)
- Umfassendes Pre-Release Testing (Geringe Fehlerdichte von Releases)

Entwicklungsprozesse, welche das Ziel verfolgen, ein kommerziell verwertbares Softwareprodukt zu entwickeln, sollten in einen möglichst planbaren Prozess eingebettet sein, unabhängig davon, ob es sich bei der Software um Individual- oder Standardsoftware handelt. Diese Planbarkeit beinhaltet die Planung von Ressourcen und die Terminierung von Meilensteinen entsprechend einem definierten Projektbudget [DeSeVo03][86]. Der Zeitpunkt und der Umfang von neuen Software-Releases muss entsprechend den Vertragsvorgaben (Individualsoftware) bzw. den Marktbedürfnissen (Standardsoftware) terminiert und geplant werden können. Dies wird in OSSD-Projekten nicht gewährleistet [DeSeVo03][87]. Zudem erwartet ein Kunde eines kommerziellen Softwareunternehmens die Bereitstellung von Dienstleistungen, die auch vertraglich garantiert werden, wie z.B. den Anwendersupport oder das Bereitstellen von Updates und Dokumentation. Da der Anwender zudem i.d.R. eine möglichst fehlerfreie Software erwartet, die genau seine, während des Entwicklungsprozesses definierten, Anforderungen erfüllt, ist ein umfangreiches Pre-Release-Testing vor der Übergabe der Software notwendig [Weber00][88]. Dies steht in direktem Kontrast zu der Veröffentlichungspraxis in OSSD-Projekten, welche das Ziel verfolgt, die Anwender bereits in einem frühstmöglichen Entwicklungsstadium in den Testprozess einzubeziehen.

6.4 Optimierungsansatz

Basierend auf den bisher identifizierten Implikationen des OSSD-Prozessmodells können verschiedene Potentiale identifiziert werden, welche die modifizierbaren Parameter des Prozessmodells und somit die Basis für die Optimierung des Prozessmodells darstellen. Diese identifizierten Optimierungspotentiale können vor allem zu einer gesteigerten Qualitätssicherung und Automatisierung, also zur expliziten Unterstützung der identifizierten Entwicklungsprozesse beitragen. Dies kann vor allem durch die Optimierung bzw. Erweiterung der folgenden Aspekte unterstützt werden:

- Prozesse und Rollen
- Artefakte
- Software-Infrastruktur (unterstützende Softwarefunktionalitäten)

Die Modifikation und Erweiterung des OSSD-Ansatezs zur Kompensation der dargestellten Schwächen und Risiken sollte die präskriptive Anwendbarkeit des Modells und dessen Anwendbarkeit evtl. auch in kommerziellen SE-Szenarien ermöglichen.

---

[85] „The central question regarding Open Source Software is how to stabilize and manage the entire development process and how to commercialize the code to an application level that satisfies the business community." [WaAb03]

[86] „In commercial projects the time is limited by budget constraints and contracts." [DeSeVo03]

[87] „Release planning for OS projects is driven by different forces: often technical maturity has higher priority than time-to-market." [DeSeVo03]

[88] „[…] customers who are paying a great deal of money for software may not like buggy Beta-Releases, and may like even less the process of updating or installing new releases on a frequent basis." [Weber00]

6.4.1 Prozesserweiterung

Da die dezentralen Entwicklungsprozesse, welche im deskriptiven Prozessmodell identifiziert und modelliert wurden, das Ergebnis eines weitgehend uneingeschränkten Evolutionsprozesses sind und daher den Anforderungen und Erwartungen einer verteilten Community weitgehend entsprechen, sollten diese Entwicklungsprozesse durch die Optimierung nicht eingeschränkt oder reglementiert, sondern lediglich unterstützt und erweitert werden. Dies kann vor allem dadurch geschehen, dass implizit existierende Rollen, wie z.B. die des Reviewers, durch die Definition geeigneter Infrastrukturen oder Artefakte explizit unterstützt werden oder indem verschiedene ergänzende Prozesse und Rollen definiert werden, welche die identizierten Schwachstellen und negativen Tendenzen kompensieren. Diese Vorgehensweise der Etablierung dedizierter Rollen und Prozesse wird auch in [DeSeVo03][89] als probate Methode identifiziert. Da diese ergänzenden Prozesse möglichst plan- und terminierbar und mit einer garantierbaren Intensität ausgeführt werden sollten, ist in diesem Kontext die Gruppe der Core Developer von besonderer Bedeutung, deren Existenz somit eine wichtige Voraussetzung für die Etablierung zusätzlicher Prozesse darstellt.

6.4.2 Spezifikation ergänzender Artefakte und Artefakteigenschaften

Das im deskriptiven Prozessmodell dargestellte Artefaktkonzept besteht nur aus wenigen formalisierten Artefakten, die bisher projektübergreifend in OSSD-Projekten etabliert sind. Es besteht daher Potential, die identifizierten Prozesse durch ein erweitertes, *optimiertes* Artefaktkonzept zu unterstützen. Daher kommt als weiterer Verbesserungsansatz die Beschreibung zusätzlicher Artefakte und die Definition ergänzender Artefakteigenschaften (Metadaten) in Frage. Diese sollten das erweiterte Prozess- und Rollenmodell unterstützen und durch eine geeignete Software-Infrastruktur implementiert und unterstützt werden.

Besonders im Bereich der technologischen Artefakte der Entwicklungsprozesse können softwaretechnisch realisierte Metadaten zur strukturierten Verwaltung des Informationsaufkommens in einem OSS-Projekt und zur adäquaten Unterstützung und Koordinierung der optimierten Entwicklungsprozesse dienen. Über entsprechende Metadaten kann z.B. die Definition geeigneter Artefaktlebenszyklen ermöglicht werden, welche durch eine artefaktbezogene Rechtevergabe zusätzlich unterstützt werden und weitgehend selbstregulierende Entwicklungsprozesse ermöglichen können. Primär wird mit der Definition geeigneter Artefakte die Erhöhung der Transparenz über alle Ressourcen und die Unterstützung autonomer und paralleler Prozesse verfolgt.

6.4.3 Software-Infrastruktur - Unterstützende Softwarefunktionalitäten

Im Anschluss an die Beschreibung verbesserter Prozesse, Rollen und Artefakte können ergänzende Softwarefunktionalitäten identifiziert werden, welche die modellierten Strukturen und Prozesse effizienter unterstützen und durch die im deskriptiven Prozessmodell identifizierte Infrastruktur nicht oder nur ungenügend bereitgestellt werden. Diese Softwarefunktionalitäten dienen somit der softwaretechnischen Realisierung des Prozessmodells, beinhalten idealerweise die Abbildung und Unterstützung der Prozesse, Rollen und Artefakte und stellen die Basis für die weitere Implementierung einer geeigneten Infrastruktur dar. Eine adäquate Software-Infrastuktur sollte möglichst parallele und autonome Prozesse unterstützen, indem sie transparenten, zentralen Zugang zu allen Artefakten und Werkzeugen zur Verfügung stellt und Transparenz über alle Prozesse schafft.

6.5 Zielstellungen für die Erweiterung und Unterstützung des OSSD-Modells

In diesem Abschnitt werden zu Beginn Zielsetzungen und Anforderungen der verteilten Softwareentwicklung im Allgemeinen dargestellt, die auch auf das OSSD-Modell anwendbar sind. Anschließend werden verschiedene OSSD-spezifische Zielstellungen für die weitere Verbesserung des Ansatzes erläutert.

[89] „One possible way is to introduce a role concept and assign some responsibilities to each role […].” [DeSeVo03]

6.5.1 Allgemeine Anforderungen verteilter Softwareentwicklung

Die Anforderungen der OSS-Entwicklung entsprechen zu weiten Teilen den allgemeinen Anforderungen der verteilten Softwareentwicklung durch dezentral agierende Akteure in virtuellen Projektteams, die auch für kommerzielle Softwareentwicklung relevant sind und sollten daher auch im Rahmen des OSSD berücksichtigt werden. [NoSc99] definiert u.a. die folgenden Anforderungen an einen Entwicklungsprozess für die verteilte Softwareentwicklung:

- Transparenter Zugang zu Artefakten
- Standardisierung von Prozessen und Artefakten
- Spezifikation bzw. Formalisierung aller Entwicklungsprozesse für die involvierten Rollen inkl. der verwendeten Werkzeuge, Artefakte und Entwicklungsumgebungen
- Koordination und Unterstützung der autonomen Aktivitäten der verteilten Akteure anhand von Prozess-Spezifikationen
- Zugangskontrolle der gemeinsamen Ressourcen zur Konsistenzsicherung
- Unterstützung von weitgehend internetbasierter Kollaboration
- Kommunikationsinfrastrukturen und –richtlinien
- Protokollierung der projektweiten Aktivitäten und Kommunikation zur Erhöhung der Transparenz

Eine zentrale Anforderung der verteilten Software-Entwicklung wird durch eine größtmögliche Standardisierung und Vereinheitlichung der individuell ausgeführten Entwicklungsprozesse und der dabei produzierten Artefakte dargestellt, um einen größtmöglichen Konsens zwischen den dezentralen Akteuren zu gewährleisten [KiLe00][90]. Dies ermöglicht die Minimierung zentraler Koordinierungsfunktionen und die einfache Integration neuer Akteure. Die dezentrale Verteilung der Akteure erfordert die zentrale Verfügbarkeit aller Artefakte und Ressourcen, wodurch eine größtmögliche Autonomie der einzelnen Akteure gewährleistet werden soll [NoSc99][91]. Die gemeinsame Verwendung von zentralen Ressourcen erfordert zudem die Reglementierung und Formalisierung der Prozesse, um Konflikte bei der Verwendung gemeinsamer Ressourcen zu vermeiden [NoSc99][92].

6.5.2 OSSD-spezifische Zielsetzungen

In den folgenden Abschnitten werden verschiedene Zielsetzungen erläutert, die bei der Modifikation des OSSD-Prozessmodells und dessen unterstützender Software-Infrastruktur berücksichtigt werden sollten und neben einer allgemeinen Verbesserung des OSSD-Ansatzes auch dessen Anwendbarkeit in proprietären SW-Entwicklungsszenarien unterstützen. Dabei wurde versucht, eine Unterscheidung dieser Zielstellungen nach dem Kriterium vorzunehmen, ob die jeweilige Anforderung durch die Erweiterung der Prozesse bzw. Rollen oder durch die Optimierung der Infrastruktur und der Artefakte realisiert werden kann. Da aber keine erschöpfende Abgrenzung einzelner Zielsetzungen möglich war, erfolgt die detaillierte Beschreibung der einzelnen Zielsetzungen im Anschluss an die Auflistung beider Zielstellungskategorien.

Zielsetzungen für die Erweiterung der Prozesse und Rollen
In diesem Abschnitt werden verschiedene Anforderungen und Zielsetzungen definiert, die mit der Erweiterung des Prozessmodells basierend auf den involvierten Teilprozessen und Rollen verfolgt werden sollten und anhand des deskriptiven Prozessmodells und der bereits dargestellten Erkenntnisse und Implikationen hergeleitet wurden:

- Qualitätssicherung (Software und Prozesse)

[90] „Consistency between developers is even more important given their sometimes vastly different experiences and goals.“ [KiLe00]

[91] „Participants in virtual enterprises retain a high degree of autonomy over their own development activities, product data, tools and environments. These conditions will increase concerns for how to accommodate heterogeneity while maintaining administrative autonomy and transparent access to shared online resources.” [NoSc99]

[92] „[...] participating teams will need to follow well-defined processes to coordinate their work and share objects, resources, intermediate work products - or more simply, artifacts - with other members.” [NoSc99]

- Minimierung der Eintrittsbarrieren
- Minimierung redundanter Aktivitäten
- Gewährleistung redundanzfreier und konsistenter Artefakte
- Steigerung der Transparenz über alle Prozesse
- Kompensation und Unterstützung ungenügend ausgeführter Prozesse
- Minimierung der Abhängigkeiten von Core Developers
- Konsensbildung durch Definition gemeinsam verwendeter Standards und Richtlinien

Zielsetzungen für eine prozessunterstützende Infrastruktur (Artefakte, Tools)
In diesem Abschnitt werden Anforderungen und Zielsetzungen definiert, welche durch Modifikation und Erweiterung des Artefaktmodells und der Softwareinfrastruktur realisiert werden können. Die bereits im vorangegangenen Abschnitt definierten Zielsetzungen, müssen durch eine adäquate Infrastruktur und die involvierten Artefakte natürlich ebenso unterstützt werden, werden aber durch die im folgenden angeführten Zielsetzungen noch ergänzt:

- Unterstützung paralleler, kollaborativer Prozesse
- Redundanzfreie und konsistente Artefaktbereitstellung
- Steigerung der Transparenz über alle Prozesse
- Softwaretechnische Unterstützung des Prozess- und Rollenmodells (Prozessautomatisierung, Workflows)
- Dezidierte Unterstützung der Core Developer
- Minimierung der Abhängigkeiten zu den Core Developers
- Unterstützung demokratischer Entscheidungsfindungsprozesse

Im folgenden werden einige der zentralen Bestandteile der angeführten Zielsetzungen detailliert erläutert.

6.5.2.1 Qualitätssicherung

In OSS-Prozessen wird zwar im Zuge des schrittweisen Softwareverbesserungsprozesse i.d.R. ein mit der proprietären Softwareentwicklung vergleichbares Level bezüglich der Softwarequalität erreicht, dies kann aber aufgrund der verteilten und nicht planbaren Prozesseigenschaften nicht permanent garantiert oder planbar gesteigert werden. Zwar werden die anwendungsbezogenen Post-Release-Tests mit sehr starker Intensität, die Pre-Release-Tests aber nur ungenügend ausgeführt. Vor allem die Qualität frühzeitig veröffentlichter Softwareversionen ist daher typischerweise weitaus geringer, als dies in proprietären Projekten mithilfe einer formalisierten Qualitätssicherung erreicht wird. Daher sollten verschiedene Prozesse zur Qualitätssicherung in das Prozessmodell integriert werden, welche das generalisierte Prozessmodell ergänzen. Vor allem im Bereich besonders kritischer, respektive besonders sicherer, zuverlässiger oder zeitkritischer Softwaresysteme ist die Integration von Qualitätssicherungsmassnahmen in den Entwicklungsprozess unabdingbar (vgl. [Massey03][93]). Beispielsweise könnte die Erweiterung des Prozessmodells um Prozesse zur koordinierten Ausführung von Pre-Release-Tests auf einem definierten Niveau dazu beitragen, die Fehlerdichte in veröffentlichten Software-Releases zu minimieren.

Natürlich trägt nach Linus' Law (vgl. *3.1.2 Prozessparallelisierung und Prozessautonomie*) auch eine allgemeine Vergrößerung der Anwender- und Entwicklergemeinde zu einer erhöhten Qualitätssicherung bei, was daher auch

[93] „In domains such as safety-critical and mission-critical systems, informal approaches are not enough. Finally, as the userbase for OSS becomes larger and the concentration of sophisticated users and developers is diluted, it is important that the OSS community evolve software methodologies that are compatible both with the tenets of the open source revolution and the needs of the user community." [Massey03]

explizit durch Erweiterungen des OSSD-Modells unterstützt werden sollte [CaLeCh02][94], was auch im folgenden Abschnitt dargestellt wird.

6.5.2.2 Minimierung der Eintrittsbarrieren

Wie bereits in *3.1.2 Prozessparallelisierung und Prozessautonomie* dargestellt wurde, ermöglicht die weitgehende Prozessparallelisierung und Prozessautonomie im Kontext der kollaborativen Entwicklungsaktivitäten, die fortwährende Integration neuer Akteure in den Entwicklungsprozess ohne einen Mehraufwand für die Koordinierung oder Organisation nach sich zu ziehen [Halloran01][95]. Dies steht im direkten Kontrast zum Anstieg des Management- und Organisationsaufwands, den die Integration neuer Akteure in proprietäre Softwareentwicklungsprozesse nach sich zieht. Eine möglichst große Anwender- und Entwicklergemeinde repräsentiert einen wesentlichen Erfolgsfaktor eines OSS-Projekts, da die Community alle produktiven Akteure des Entwicklungsprozesses repräsentiert und eine Voraussetzung für den kollaborativen OSS-Entwicklungsprozess darstellt. Zur optimalen Entwicklung einer Anwender- und Entwicklergemeinde ist es daher sinnvoll, die Eintrittsbarrieren für die Anwendung und Entwicklung der OSS zu reduzieren und den notwendigen Aufwand zum Projekteinstieg zu minimieren.
Als negatives Fallbeispiel und Beleg für die Notwendigkeit der Minimierung der Eintrittsbarrieren kann das Mozilla-Projekt betrachtet werden, welches in der Projektphase nach der initialen Veröffentlichung nur relativ langsam eine Projekt-Community etablieren konnte, da die Eintrittsbarrieren durch z.T. proprietären, unvollständigen und nicht modularen bzw. unverständlichen Quellcode vergleichsweise hoch waren und auch eine nicht OSD-konforme Softwarelizenz verwendet wurde (vgl. [Dietze03]). Eine erfolgreiche Etablierung umfangreicher OSS-Entwicklungsprozesse wurde erst durch die Definition einer geeeigneten Lizenz und durch das Re-Engineering des Quellcodes ermöglicht.

6.5.2.3 Minimierung redundanter Aktivitäten

Die möglichst redundanzfreie und konsistente Verwaltung des gesamten Informationsaufkommens im Rahmen eines OSS-Projekts und die damit verbundene Steigerung der Transparenz ermöglicht neben der Reduzierung der Eintrittsbarrieren auch die Vermeidung redundanter Aktivitäten. Da im Rahmen der Prozessbetrachtung der parallelisierten Prozesse des OSSD-Modells eine Tendenz zur redundanten Ausführung verschiedener Aktivitäten festgestellt werden konnte, stellt deren frühzeitige Vermeidung eine weitere sinnvolle Zielstellung für die Modifikation des OSSD-Modells dar, um dadurch einen effizienteren Ressourceneinsatz zu ermöglichen.

6.5.2.4 Kompensation ungenügend ausgeführter Prozesse

Wie in *A.4 Entwicklerfokus auf Implementation* dargestellt wurde, werden nicht alle Prozesse und Aktivitäten im Rahmen der OSS-Entwicklung mit der gleichen Intensität ausgeführt, was mit den verschiedenen Interessensschwerpunkten und Motivationsfaktoren der verteilten Akteure und dem fehlenden zentralen Management der Entwicklungsprozesse begründet werden kann. Dies führt zu einer Tendenz der Konzentration auf die Implementationsaktivitäten und der Vernachlässigung peripherer Aktivitäten wie z.B. der Dokumentation oder des Supports. Daher sollten diese weniger intensiv ausgeführten Prozesse sowohl durch die Definition dedizierter Prozesse als auch durch eine adäquate Infrastruktur unterstützt und erweitert werden. Dies könnte z.B. auch durch die Gruppe der Core Developer zentral unterstützt und gesteuert werden.

[94] „The goals of the QA processes, which are to have a large population of volunteers performing tests, creating test cases, and helping developers triage bugs, need to be supported by the community processes.“ [CaLeCh02]

[95] „It enables certain categories of participants to gracefully engage and disengage in a project without adverse consequences to the project.“ [Halloran01]

6.5.2.5 Softwaretechnische Unterstützung des Rollenmodells - Prozessautomatisierung

Alle identifizierten Rollen und die von ihnen ausgeführten Prozesse sollten softwaretechnisch durch eine geeignete Infrastruktur abgebildet und unterstützt werden. Dies kann z.B. die Realisierung automatisierter Workflows entsprechend der identifizierten Aktivitäten und die Implementation eines rollenbezogenen Rechtekonzepts umfassen. Neben dieser allgemeinen Anforderung zur softwaretechnischen Prozessimplementierung sind die folgenden Aspekte von besonderer Bedeutung.

Dedizierte Unterstützung der Core Developer

Wie bereits dargelegt wurde, tendieren OSS-Projekte dazu, einen Kreis von dedizierten Core Developers zu etablieren, die über erweiterte Privilegien verfügen und für eine Vielzahl von Quellcodemodifikationen und dedizierte Aktivitäten verantwortlich zeichnen. Aufgrund der großen Bedeutung der Beiträge dieser Akteure sollten die Core Developer und die von ihnen ausgeführten Prozesse explizit in einem präskriptiven Prozessmodell berücksichtigt und unterstützt werden. Zudem sind die Prozesse der Core Developer zum Teil reglementierter und formalisierter (vgl. *5.1.3.3 Review und Commit* und *5.2.1 Software-Release*), als die verteilten Aktivitäten der restlichen Projektmitglieder und sollten daher bei der softwaretechnischen Prozessunterstützung und –automatisierung explizit berücksichtigt werden.

Automatisierung und Unterstützung der Releaseprozesse

Wie bereits umfassend dargestellt wurde, ist ein zentraler Erfolgsfaktor des OSS-Modells die frühzeitige und häufige Veröffentlichung von Quellcode, um diesen einer größtmöglichen Anwendergemeinde zugänglich zu machen und dadurch den Prozess der graduellen Softwareverbesserung zu ermöglichen. Daher sollten die analysierten Releaseprozesse soweit wie möglich automatisiert, optimiert und durch die instrumentalisierte Software-Infrastruktur unterstützt werden. Dies kann z.B. auch die Steigerung der Releasefrequenz ermöglichen, um eine größtmögliche Zahl von Akteuren und somit potentiellen Testern bereits zu einem frühen Zeitpunkt zu erreichen.

6.5.2.6 Minimierung der Abhängigkeiten

Da die starke Dezentralisierung und Parallelisierung der Prozesse zugleich eine wichtige Voraussetzung für das Gelingen eines OSS-Projekts darstellt, sollten die Abhängigkeiten der Community von den Maintainern und Core Developers soweit wie möglich minimiert werden, um eine möglichst große Unabhängigkeit der kollaborativen und dezentralen Prozesse zu erreichen. Diese Anforderung kollidiert z.T. mit anderen Zielsetzungen, wie z.B. der Unterstützung der Core Developer und kann daher nur bis zu einem bestimmten Grad realisiert werden, da einige Aktivitäten, wie z.B. der Commit des Quellcodes oder der Release einer neuen Version der OSS, ausschließlich durch enstprechend qualifizierte Rollen, z.B. die der Core Developer, ausgeführt werden sollten. Die kollaborativen Prozesse durch die dezentral verteilte Community sollten aber dabei nicht unnötigerweise durch zu viele Interdependenzen zu den zentralisierteren Akteuren eingeschränkt werden, was beispielsweise durch die Reduktion der Schnittstellen zwischen den Aktivitäten der Core Developer und der verteilten Akteure realisiert werden könnte.

6.5.2.7 Konsensbildung bzw. Entscheidungsfindung

Da im OSS-Entwicklungsmodell sehr heterogene Interessen, Einstellungen und Qualifikationen der verteilten Akteure aufeinandertreffen und zum Teil miteinander konkurrieren, sollte die Erreichung eines gemeinsamen minimalen Konsens über die Modalitäten der Durchführung der Prozesse durch die Definition von gemeinsam verwendeten Standards, Konventionen und Verhaltensrichtlinien unterstützt werden. Zudem erfordert die demokratische und offene Struktur der OSS-Entwicklung und die Abwesenheit von autoritärer Kontrolle und Management die Definition demokratischer Prozesse zur Entscheidungsfindung und Konfliktlösung.

7 Erweiterung der Entitäten des OSSD-Modells

In diesem Kapitel werden die verschiedenen Modifikationen und Erweiterungen dargestellt, welche das dargestellte OSSD-Modell verbessern und dessen präskriptive Verwendung ermöglichen. Diese Erweiterungen basieren auf den in *6 Verbesserungspotentiale und –ansätze des OSSD-Modells* dargestellten Ansätzen und Zielstellungen. Die Entwicklung der verschiedenen Erweiterungen des Prozessmodells orientieren sich dabei sowohl an den Erfahrungen aus der Durchführung der Fallstudien in [Dietze03] als auch an Methoden, die bereits in einigen OSSD-Projekten praktiziert werden.

7.1 Erweiterung existierender Prozesse und Rollen

Die in diesem Kapitel thematisierten Prozesse repräsentieren dezidierte Prozesse, welche die im deskriptiven Prozessmodell dargestellten Prozesse entsprechend den im vorangegangenen Abschnitt hergeleiteten Anforderungen unterstützen und somit das deskriptive Prozessmodell um verschiedene Prozesse und Rollen erweitern. Die hier dargestellten Prozessbeschreibungen dienen zu einem großen Teil der Realisierung von qualitätssichernden Elementen in den kollaborativen Entwicklungsprozessen und der Kompensation von durch die Community in ungenügendem Masse ausgeführten Prozessen.

Einschränkungen
Ein Kernproblem der Realisierung derartiger Prozessmodifikationen stellt die Integration dieser Prozesserweiterungen in die kollaborativen und weitgehend unreglementierten Entwicklungsprozesse und die damit verbundene Adaption durch die Community (vgl. [Halloran01]) dar. Daher wird ein wichtiges Kriterium bei der Entwicklung und Spezifikation dieser Prozesse durch die Bedingung repräsentiert, dass die modifizierten Prozesse die evolutionär herausgebildeten Entwicklungsprozesse des deskriptiven Modells nicht einengen, beschränken oder reglementieren, sondern lediglich unterstützen, um die freie Evolution der Prozesse nicht zu behindern. Da die neu spezifizierten Prozesse fast aussschließlich koordinierende und supplementäre Aufgaben repräsentieren, welche zentral durch entsprechend qualifizierte und privilegierte Akteure ausgeführt werden sollten, sollten die damit verbundenen Rollen ausschließlich Akteuren der bereits dargestellten Core Developers oder besonders qualifizierten und den Maintainern eines Projekts nahestehenden Entwicklern zugewiesen werden. Die hier dargestellten Prozesse repräsentieren somit lediglich Erweiterungen des identifizierten Prozessmodells, z.B. der impliziten Review- und Testprozesse des deskriptiven Modells.

Prämisse: Erweitertes Request Tracking
Einige der im folgenden dargestellten erweiterten Prozesse basieren auf der Voraussetzung eines gesamtheitlicheren Request Tracking, was daher in diesem Abschnitt einführend als Prämisse für die Ausarbeitung der folgenden Prozesse dargestellt wird.
In einem OSS-Projekt entstehen neben den klassischen Change Requests, also Änderungsanforderungen an den Quellcode der OSS, permanent auch verschiedene Anforderungen bezüglich peripherer Artefakte, an die Projektinfrastruktur, die Projektorganisation oder Supportanfragen. Da auch diese Anforderungen direkte Aktivitäten nach sich ziehen, die für die gesamte Community von Relevanz sind bzw. sein können, sollten auch diese Anforderungen analog zur Verwaltung von Change Requests zentral und für jeden Akteur transparent verwaltet werden und der Status ihrer Bearbeitung stets transparent sein (vgl. [Cubranic01][96]). Daher werden in *7.2.2 Request-Artefakte* allgemeine Request-Artefakte (Issues, Task Requests) eingeführt, welche somit nicht nur jegliche Anforderungen bezüglich dedizierter Artefakte der OSS bzw. der Peripherie, sondern auch

[96] „[...] there is still no support for higher-level coordination group decision-making, knowledge management, task scheduling and progress tracking, etc. - for any of the projects examined here.“ [Cubranic01]

organisatorische bzw. infrastrukturelle Aufgaben oder Supportanfragen umfassen und somit ein gesamtheitliches Task Management ermöglichen.

7.1.1 Überblick: Prozesserweiterungen und die realisierten Zielsetzungen

Die folgende Tabelle visualisiert die verschiedenen entwickelten Prozesserweiterungen des OSSD-Prozessmodells und stellt ihnen die mit ihrer Einführung verfolgten Zielsetzungen aus *6.5 Zielstellungen für die Erweiterung und Unterstützung des OSSD-Modells* gegenüber. Der jeweilige Prozess sollte idealtypisch stets durch eine explizite neu eingeführte Rolle ausgeführt, die aus Platzgründen nicht in der Tabelle dargestellt wurden aber im Rahmen der detaillierten Prozessdarstellungen in den folgenden Abschnitten bzw. Anhang *B Unterstützende und erweiterte Prozesse* erläutert werden.

Tab. 7.1: Erweiterte Prozesse im Kontext der Optimierungsziele

		Optimierungszielstellungen						
		Qualitätssicherung	Minimierung der Eintrittsbarrieren	Minimierung redundanter Aktivitäten	Redundanzfreie und konsistente Artefakte	Transparenz über Prozessfortschritt	Kompensation ungenügend ausgeführter Prozesse	Minimierung der Abhängigkeiten von Core Developers
Erweiterte Prozesse	Source Code Review	x	x		x		x	
	Software Test	x	x				x	
	Test Case Development	x	x				x	x
	Content Artifact Management	x	x	x	x	x	x	x
	Communication Review		x	x	x		x	
	Request/Issue Review	x	x	x	x	x	x	x
	Release Process (incl. periodical build develoment)		x				x	x
	Infrastructure Maintenance		x		x		x	x

In den folgenden Abschnitten werden exemplarisch die folgenden, erweiterten Prozesse und die im Zuge der Prozessdefinition neu eingeführten Rollen dargestellt:

- Quellcodereview (*Source Code Review*)
- Review der Anforderungsartefakte (*Request Review*)
- Erweiterter Releaseprozess inkl. periodischer Erstellung und Veröffentlichung von Builds des Entwicklungsquellcodes (*Release Process incl. periodical build development*)

Die verbleibenden Prozesserweiterungen werden im Anhang *B Unterstützende und erweiterte Prozesse* eingehender beschrieben.

7.1.2 Source Code Review

Der dezentrale Review jeglicher Quellcodeartefakte stellt gerade im OSS-Bereich eine elementare Aktivität dar, welche durch alle Akteure der Community unabhängig ausgeführt wird und somit die typische Basis für die Qualitätssicherung von OSS darstellt. Diese qualitätssichernden Aktivitäten werden, wie bereits veranschaulicht

wurde, nicht notwendigerweise zu jeder Phase des Lebenszyklus eines Quellcodeartefakts in ausreichendem Umfang ausgeführt. Dies resultiert aus dem nicht planbaren Charakter des Entwicklungsmodells, der eine ausreichende Softwarequalität somit nicht zu jedem Zeitpunkt gewährleisten kann. Daher erscheint es sinnvoll, diesen Unsicherheitsfaktor durch eine Erweiterung der dezentralen Reviewprozesse um zentral ausgeführte Reviewaktivitäten zu minimieren und dadurch eine kontinuierliche Kontrolle der Softwarequalität zu realisieren. Somit ergänzen die hier dargestellten Reviewprozesse lediglich die dezentralen Reviewprozesse, die im Kontext des Patchentwicklungszyklus bereits als integraler Bestandteil des deskriptiven Prozessmodells modelliert wurden. Ähnliche Qualitätssicherungselemente werden bereits ansatzweise zum Beispiel im Kontext des Mozilla-Projekts bereits (vgl. [ReFo02][97] und [Dietze03]).

Die folgenden Aspekte sollten im Rahmen dieses Reviewprozesses untersucht werden:

- Lesbarkeit des Quellcodes
- Konformität zur Gesamtarchitektur (Modularität, Schnittstellen etc.)
- Einhaltung projektspezifischen Programmierkonventionen
- Softwarefunktionalität

Die Evaluierung der Softwarefunktionalität liegt nicht im primären Fokus dieser Reviewaktivitäten, da Kriterien wie Funktionalität, Usability oder Performance noch durch die expliziten Software-Tests (Anwendungstests) geprüft werden. Zur Erreichung eines Konsens über die Durchführung dieses Prozesses sollte bezüglich der Kriterien und Methoden des Reviewprozesses ein Review-Guide definiert werden, der die elementaren Vorgaben für den Prozess des Review definiert.

Im Rahmen des ergänzenden Source Code Review kann entsprechend des Betrachtungsgegenstandes und des Auslöseereignisses zwischen den folgenden Varianten unterschieden werden:

- Patch Review
- Periodischer Review des Entwicklungsquellcodes

Patch Review
Der *Patch Review* wird als Bestandteil eines Patchentwicklungszyklus durch den Source Code Reviewer ausgeführt und ergänzt den Review durch die gesamte Community. Die erfolgreiche Durchführung des Patch Review sollte die notwendige Voraussetzung für den Commit eines Patches in den zentralen Entwicklungsquellcode darstellen. In dieser Phase eines Patchlebenszyklus sollten lediglich die bereits genannten Kriterien evaluiert und noch kein dedizierter Software-Test durchgeführt werden, um den Entwicklungsquellcode nicht zu reglementieren oder in seiner Evolution zu beschränken, da dies eine elementare Bedingung für die Realisierung von OSS-Projekten darstellt. Falls das Patch nicht den definierten Kriterien entspricht, wird durch den Reviewer eine Überarbeitung des Patches durch Entwickler eingeleitet und dieser Prozess iterativ wiederholt, bis eine erfolgreiche Evaluierung durchgeführt werden kann. Bei erfolgreich durchgeführtem Review wird das Patch direkt an entsprechend priviligierte Committer weitergeleitet, welche sinnvollerweise ebenfalls durch die Source Code Reviewer repräsentiert werden können.

Periodischer Review des Entwicklungsquellcodes
Der periodisch ausgeführte Review des gesamten Entwicklungsquellcodes stellt eine weitere wichtige Aufgabe im Rahmen der gesamtheitlichen Qualitätssicherung dar. Diese Aktivitäten dienen der präventiven Software-Maintenance, die in OSS-Projekten häufig nur ungenügend ausgeführt wird (vgl. [TrGoLeHo00][98]) und haben vor

[97] „Mozilla has taken this idea and implemented a formalized system which solves problems without becoming an overly burdensome bureaucracy.“ [ReFo02]

[98] „[…] the active lifespan is devoted to new development and corrective maintenance; relatively little time is spent performing preventive maintenance […], such as restructuring and redesign.” [TrGoLeHo00]

allem die Qualitätssicherung bezüglich der sich permanent verändernden Softwarearchitektur und des –designs zum Ziel. Da hierfür ähnliche Qualifikationen wie für den Patch Review erforderlich sind, sollte diese Aufgabe ebenfalls durch die Source Code Reviewer ausgeführt werden. Die Aktivitäten des periodischen Source Code Review können dazu führen, dass die Überarbeitung eines bestimmten Quellcodeartefakts direkt mit dem verantwortlichen Entwickler koordiniert oder ein allgemein verfügbarer Change Request generiert wird. Die Frequenz für die Ausführung dieser Prozesse sollte dabei an den Umfang des Quellcodes und der verfügbaren Ressourcen angepasst werden.

Die folgende Grafik visualisiert den gesamten Prozess des *Source Code Review*:

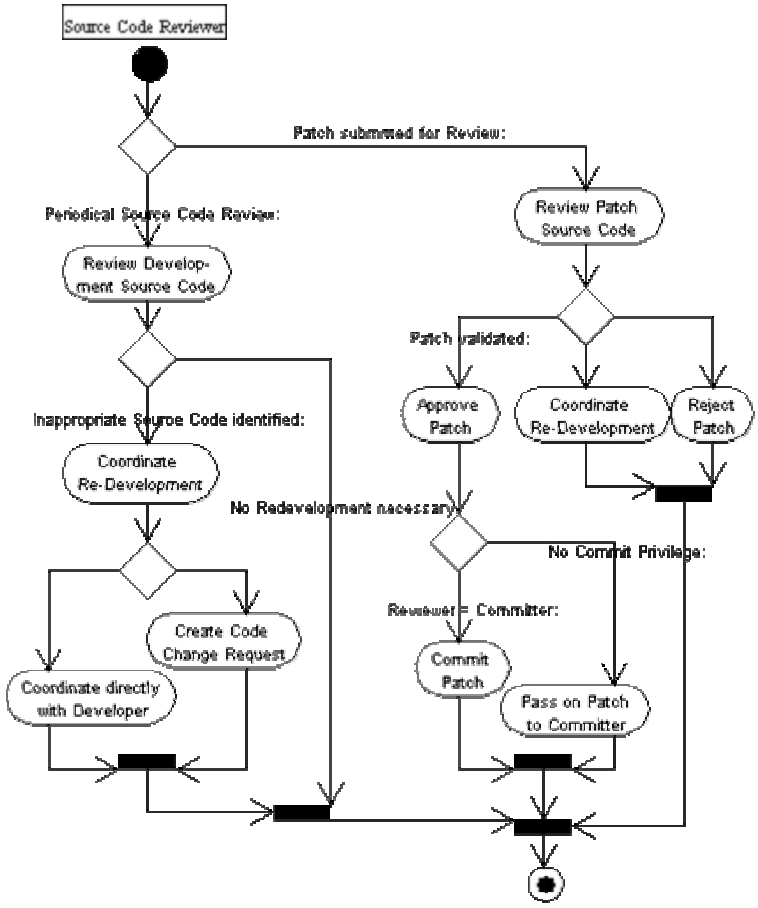

Abb. 7.1: Aktivitätssicht auf den Prozess des *Source Code Review*

Rolle: Source Code Reviewer

Der zentrale erweiterte Source Code Review sollte durch eine dedizierte Rolle, die des Source Code Reviewers ausgeführt werden, der über entsprechende Qualifikationen verfügen und daher den Core Developers des Projekts entstammen sollte. Die Rolle des Source Code Reviewers ist somit für die folgenden Prozesse verantwortlich, die in der Anwendungsfallsicht dargestellt wurden. Diese dedizierte und explizit zu ernennende Rolle ergänzt somit lediglich die implizit identifizierten Reviewer des kollaborativen Patchentwicklungszyklus und sollte durch erweiterte Softwarefunktionalitäten unterstützt werden.

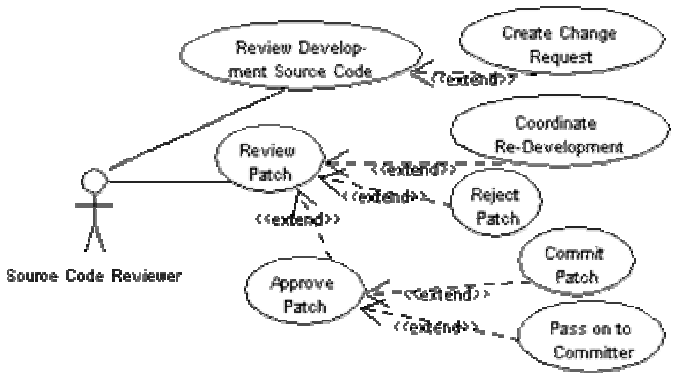

Abb. 7.2: Anwendungsfälle des Source Code Reviewers

7.1.3 Request Review

Im deskriptiven Modell wurde der Prozess des Review der Requirements (Bug Triage) als impliziter Prozess identifiziert und modelliert, welcher die kontinuierliche Evaluierung und Validierung der verwalteten Change Requests durch die Akteure der verteilten Community beinhaltet. Da es sich beim Requirements Review um einen impliziten Prozess handelt, der nicht explizit definiert bzw. gesteuert wird, ist somit nicht zu jedem Zeitpunkt gewährleistet, dass er mit hinreichende Intensität ausgeführt wird.

Es ist für die dezentral agierenden Akteure aber von großer Wichtigkeit, dass die zentral verwalteten Request-Artefakte (vgl. *7.2.2 Request-Artefakte*) stets in einem aktuellen, konsistenten und möglichst redundanzfreien Zustand sind. Dies ermöglicht, vereinfacht und beschleunigt die dezentralen Entwicklungsprozesse für die verteilten Akteure, da so eine individuelle Verifikation und Validierung der Requests obsolet wird. Daher erscheint es sinnvoll, diese Aktvitäten durch einen dezidierten Akteur permanent ausführen zu lassen und zudem auf den Review aller in *7.2.2 Request-Artefakte* dargestellten Request-Artefakte zu erweitern. Diese Aktivität sollte die bestehenden kollaborativen Entwicklungsprozesse und die Erstellung der verschiedenen Requests durch die verteilten Akteure in keiner Weise einschränken oder limitieren, sondern lediglich ergänzend einen bestimmten Grad der Qualitätssicherung bezüglich der verwalteten Anforderungsartefakte realisieren. Dieser Prozess wird nicht in Form einer neuen Aktivitätssicht modelliert, da er analog zu dem im deskriptiven Modell dargestellten und implizit identifizierten Prozess des Requirements Review ausgeführt wird und der Unterschied primär in der kontinuierlichen und garantierten Ausübung durch eine dezidierte Rolle bzw. in der gesamtheitlichen Überwachung aller Request-Artefakte besteht.

Rolle: Request Reviewer

Der Request Reviewer sollte über erweiterte Privilegien bezüglich der verwalteten Anforderungsartefakte verfügen, um beispielsweise den Lebenszyklus einzelner Artefakte verändern oder obsolete Artefakte deaktivieren bzw. löschen zu können. Die Instanziierung einer dedizierten Rolle für die beschriebenen Aufgaben wird auch in [Behlendorf99][99] als notwendig dargestellt. Diese Privilegien sollten in verschiedenen Phasen der Artefaktlebenszyklen ausschließlich ausgewählten Akteuren vorbehalten sein, wobei die Erstellung der Change Requests natürlich jedem Akteur ermöglicht sein muss. Die Rolle des Request Reviewers sollte nur durch einen sehr qualifizierten Akteur ausgeübt werden, der über langfristige Erfahrungen im Rahmen des Projekts verfügt und ebenfalls der Gruppe der Core Developers angehört.

[99] „The bug database maintainer would be the first line of support, someone who goes through the submisssions on a regular basis and weeds out the simple questions, tosses the clueless ones, and forwards the real issues on to the developers." [Behlendorf99]

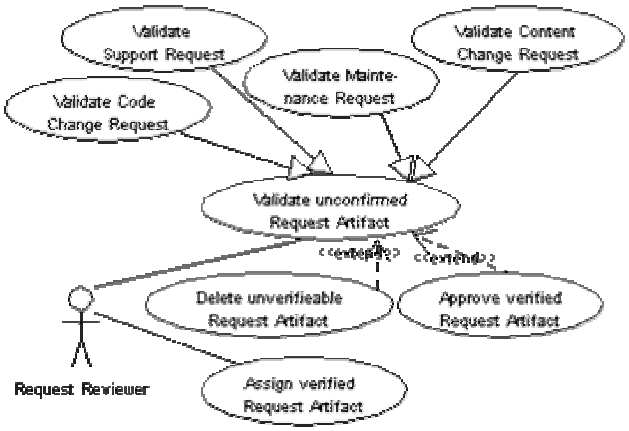

Abb. 7.3: Use Cases des Request Reviewers

7.1.3.1 Koordination des Support

Wie in [Dietze03c] dargestellt wird, wird der Support der Anwender und Entwickler ebenfalls primär durch die dezentralen Akteure in einem kollaborativen Prozess zur Verfügung gestellt. Dies ermöglicht vor allem in hochfrequentierten Projekten die Evolution eines sehr funktionalen kollaborativen Support-Ansatzes, welcher dem klassischen Support kommerzieller Dienstleister durchaus überlegen sein kann. Trotzdem kann nicht zu jedem Zeitpunkt des Projekts und bezüglich jeder Supportanfrage eine ausreichend schnelle und erschöpfende Bewältigung gewährleistet werden, da die notwendigen Aktivitäten nicht zentral koordiniert oder terminiert werden und auf freiwilliger Mitwirkung beruhen. Daher sollte dieser Aspekt des OSS-Prozessmodells durch die Etablierung dedizierter Supportprozesse unterstützt werden, welche die dezentralen Aktivitäten ergänzen und koordinierend bzw. ausgleichend wirken, falls ein Bedarf identifiziert werden kann. Diese Support-Koordination kann, basierend auf den in *7.2.2 Request-Artefakte* dargestellten Artefakten, als spezielle Ausprägung des dargestellten Request Review-Prozesses betrachtet werden.

Ein weiteres Problem in einigen OSS-Projekten ist die unzureichende Gewährleistung der Nachhaltigkeit der individuellen Supportaktivitäten da keine gemeinsame und konsistent und projektweit verwaltete Informationsquelle aller Supportinformationen existiert und somit die langfristige, redundanzfreie Archivierung aller relevanten Informationen nicht gewährleistet wird. Dies führt dazu, dass häufig supportrelevante Themen individuell bzw. bilateral zwischen einzelnen Akteuren diskutiert werden, woraus große Redundanzen bei der individuellen Bearbeitung ähnlicher Problemstellungen resultieren. Dies wird zudem durch die inkonsistente Verwendung der verschiedensten Kommunikationskanäle zur Diskussion supportrelevanter Aspekte begünstigt. Durch die Steigerung der Transparenz über alle supportrelevanten Informationen und Aktivitäten können die Einstiegsbarrieren für neue Akteure zur Mitwirkung am Supportprozess reduziert und dadurch die Menge der aktiv involvierten Akteure gesteigert werden. Es erscheint daher sinnvoll, jegliche supportrelevante Informationen und Anfragen in einer einzigen und zentralen Datenbasis analog zu den Change Requests zu verwalten und zentral zur Verfügung zu stellen, um die Transparenz zu erhöhen, die Recherchemöglichkeiten zu verbessern, Redundanzen zu verringern und die Nachhaltigkeit zu gewährleisten.

Review der Support Requests

Die bereits angesprochene zentrale Koordination der Supportprozesse im Rahmen eines kontinuierlichen Support-Request-Review unterstützt die zeitnahe und erschöpfende Bearbeitung der Support Requests zu jedem Zeitpunkt. Der hier beschriebene Prozess wird natürlich auch kollaborativ durch die gesamte Community ausgeführt und stellt in dieser zentralisierten Form lediglich eine Ergänzung dieser Aktivitäten und des gesamtheitlichen Request Reviews dar. Eine dezidierte Rolle, der Support Reviewer, überwacht im Rahmen dieses Prozesses kontinuierlich den Support-Prozess. Sobald unbewältigte Anfragen identifiziert werden können, versucht er, diese selbständig zu bearbeiten, indem er auf evtl. vorhandene Dokumentationsartefakte verweist, welche zur Problemlösung beitragen.

Falls dies nicht möglich ist, versucht der Support Reviewer, etwaige qualifizierte Entwickler zu identifizieren und mit diesen die Bearbeitung der Anfrage zu koordinieren.

Falls häufig wiederkehrende Supportanfragen identifiziert werden, die noch nicht durch ein explizites Dokumentationsartefakt thematisiert wurden, koordiniert der Support Reviewer mit dem Dokumentationsmanager die Erstellung oder Modifikation eines entsprechenden Artefakts bzw. generiert einen diesbezüglichen Content Change Request (*7.2.2.2 Content Change Request*). Dies gewährleistet die nachhaltige Dokumentation aller im Supportprozess generierten Informationen und die Vermeidung von redundanten Supportaktivitäten. Aufgrund der kontinuierlichen, iterativen Ausführung dieser Aktivitäten, wurde kein finaler Zustand dieses Prozesses in das Modell integriert.

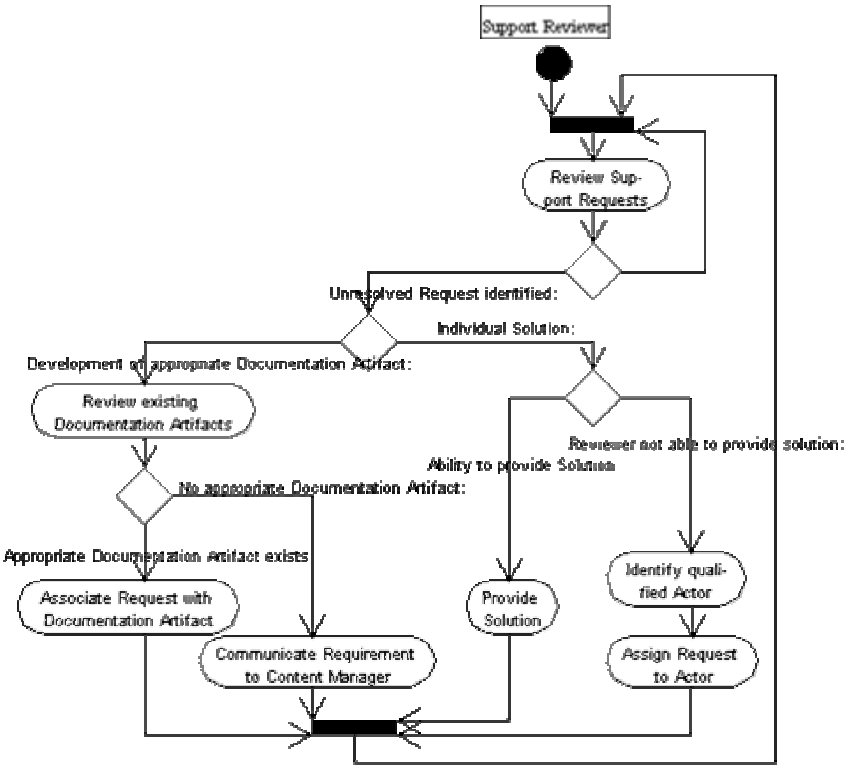

Abb. 7.4: Aktivitätssicht *Review Support Requests*

Rolle: Support Reviewer
Der Support Reviewer muss über sehr gute Kenntnisse der OSS und vor allem der OSS-Dokumentation verfügen. Zur Identifikation geeigneter Dokumentationsartefakte bzw. zur Koordination der Entwicklung neuer Artefakte sollte der Support Reviewer eng mit dem Content Manager kooperieren. Daher sollte auch dieser Akteur den Core Developers zugeordnet werden können.

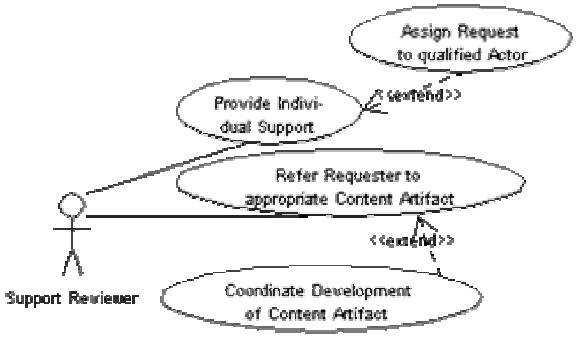

Abb. 7.5: Anwendungsfallsicht des Support Reviewers

7.1.4 Erweiterter Releaseprozess

Wie bereits dargestellt wurde, stellt die häufige Veröffentlichung aller aktuellen Quellcodemodifikationen eine elementare Voraussetzung für den Erfolg eines OSS-Projekts dar. Die dynamische und häufige Publikation von Releases umfasst sowohl die Publikation von offiziellen und getesteten neuen Softwareversionen als auch von Test-Releases, welche vor allem Testzwecken in Vorbereitung der Veröffentlichung einer neuen Softwareversion dienen. Durch eine zentrale Veröffentlichung stehen die Quellcodeänderungen des aktuellen Entwicklungsquellcodes allen interessierten Anwendern zur Verfügung und können somit von einer größtmöglichen Anwendergemeinde eingesetzt werden, welche die Software dabei implizit testet und essentiell zu deren Verbesserung beiträgt. Die so realisierten Testprozesse sind somit weitgehend parallelisiert und ermöglichen die Integration einer nahezu beliebigen Anzahl von Akteuren ohne einen Mehraufwand zu erzeugen.

Periodische Generierung von Builds

Es kann daher als sehr sinnvoll betrachtet werden, möglichst häufig sogenannte *Snapshots* des aktuellen Entwicklungsquellcodes zu veröffentlichen und diese auch in kompilierter Form einer breiteren Zielgruppe zur Verfügung zu stellen. Außerdem wird durch die kontinuierliche Kompilation des Entwicklungsquellcodes dessen Kompilierbarkeit sichergestellt und dadurch ein weiterer Beitrag zu einer umfassenderen Qualitätssicherung geleistet. Daher stellt die dynamische und permanente Kompilation des Entwicklungsquellcodes und die periodische Veröffentlichung der so erstellten Test-Releases eine sinnvolle Erweiterung der Releaseprozesse dar und wird auch schon im Rahmen der Nightly Build-Prozesse des Mozilla-Projekts erfolgreich praktiziert (vgl. [Dietze03]).

Die Frequenz, in welcher der Quellcode kompiliert und die Test-Builds veröffentlicht werden, kann dabei je nach Projekt variieren, wobei eine tägliche Ausführung analog zum Mozilla-Projekt als geeignete Vorgehensweise erscheint. Zudem bestehen verschiedene Optionen in Bezug auf den Quellcodeumfang, der periodisch kompiliert wird. Es erscheint denkbar, lediglich elementare Module, welche die Kernfunktionalitäten der jeweiligen OSS zur Verfügung stellen oder alle Quellcodeelemente, die in einem bestimmten Zeitraum modifiziert wurden, zu kompilieren, während periphere Quellcodesegmente nicht oder nur in einer geringeren Frequenz kompiliert und veröffentlicht werden.

Die periodische Kompilation sollte für die verschiedenen Plattformen durchgeführt werden, die von der OSS unterstützt werden sollen, wobei zumindest die verbreitetsten einbezogen werden sollten. In jedem Fall wird das Ergebnis der Kompilation auf der Website des Projekts kommuniziert, um diesbezüglich die Transparenz zu erhöhen. Falls die Kompilation des Quellcodes für eine oder mehrere Plattformen fehlschlägt, wird entweder der verantwortliche Quellcode entfernt oder die direkte Überarbeitung des entsprechenden Quellcodes koordiniert. Dieser Prozess wird solange iterativ wiederholt, bis der Quellcode erfolgreich kompiliert werden konnte. Anschließend erfolgt die Veröffentlichung der so erstellten Builds. Der noch ungeteste Zustand der Builds sollte klar signalisiert und evtl. sogar die Publikation in einem, von anderen Releases der OSS klar getrennten Bereich

durchgeführt werden, um alle Akteure über deren instabilen Zustand in Kenntnis zu setzen und explizit nur Anwender zu adressieren, welche die Absicht haben, aktiv am Testprozess teilzuhaben. Die kollaborativen Software-Tests, welchen die erstellten Builds unterzogen werden, sollten dabei durch verschiedene, zentral ausgeführten Software-Tests (vgl. *B.1 Software-Test*) ergänzt werden.

Die folgende Grafik repräsentiert den erweiterten Prozess der periodischen Erstellung von Test-Builds, der den Tätigkeitsbereich des im deskriptiven Prozessmodell dargestellten Release Managers erweitert oder durch einen dedizierten Build Coordinator ausgeführt werden kann.

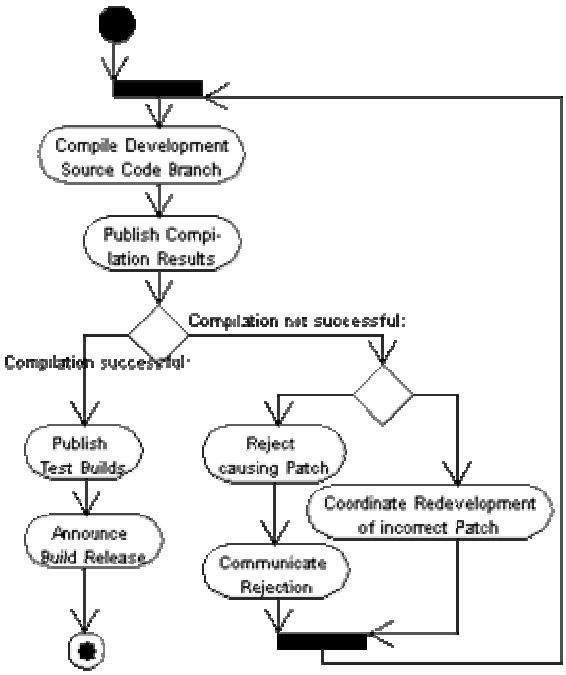

Abb. 7.6: Aktivitätssicht der periodischen Erstellung und Publikation von Test-Builds

Rolle: Dedizierter Release Manager bzw. Build Coordinator
Der im deskriptiven Prozessmodell dargestellte, allgemeine Release Prozess wird diesen Anforderungen bereits weitgehend gerecht und wurde daher in diesem Abschnitt nicht explizit dargestellt. Die operative Ausführung dieses Prozesses wird durch die Rolle des Release Managers ausgeübt, wobei festgestellt werden kann, dass diese Funktion häufig durch die Person des Maintainers eines Projekts ausgeübt wird (vgl. [FoBa02][100]). Um eine hohe Veröffentlichungsfrequenz zu ermöglichen und alle Phasen des Releaseprozesses probat und effizient ausführen zu können, wird es aber als sinnvoll betrachtet, dass ein dedizierter Akteur die operative Durchführung des Software-Release verantwortet und bezüglich der Release Planung mit den Maintainern kommuniziert.

Die folgende Anwendungsfallsicht stellt die eben beschriebenen Aktivitäten zur periodischen Erstellung von Test Builds dar, welche von einem dedizierten Build Coordinator ausgeführt werden können oder gegebenenfalls die Aktivitäten des Release Managers erweitern.

[100] „In kleinen Projekten sind häufig der Release Manager und der Betreuer ein und dieselbe Person." [FoBa02]

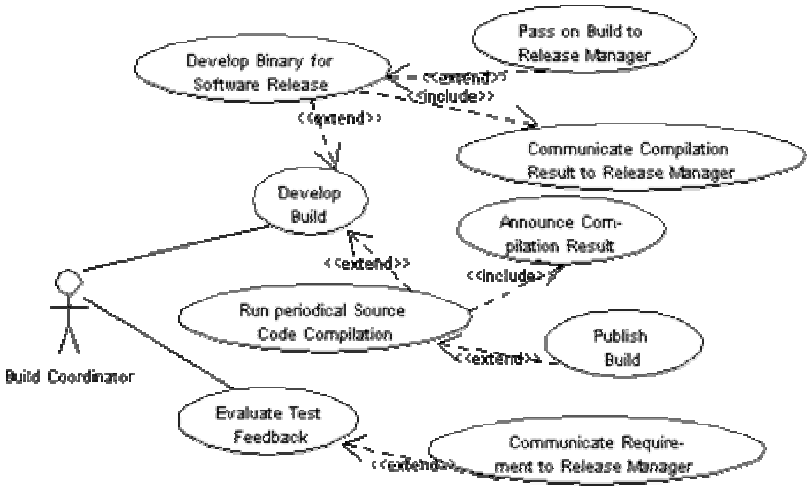

Abb. 7.7: Use Cases eines ergänzenden Build Coordinators

7.1.5 Erweitertes Rollenmodell

Die folgenden Rollen sollten gemäß der dargestellten Prozesserweiterungen (vgl. auch Anhang *B Unterstützende und erweiterte Prozesse*) im Rahmen eines kollaborativen Softwareentwicklungsprojekts realisiert werden, um identifizierte Schwachstellen des evolutionär herausgebildeten OSS-Entwicklungsmodells zu kompensieren:

- Source Code Reviewer
- Software-Tester
- Test Case Developer
- Content Manager
- Communication Reviewer
- Requirements Reviewer
- Support Reviewer
- Environment Maintainer

Diese dedizierten Rollen stehen den Maintainern eines Projekts nahe, gehören im Idealfall dem Kreis der Core Developer an und sollten durch die Maintainer eines Projekts oder ähnlich qualifizierte Personen ernannt werden. Sie führen die dargestellten, ergänzenden Prozesse aus und können bereits während der Initialisierung eines Projekts vorgesehen werden. Es erscheint ebenfalls denkbar, dass diese Rollen und Prozesse durch ein geschlossenes Entwicklerteam einer kommerziellen Unternehmung bereitgestellt werden, wobei verschiedene entsprechende Szenarien in *8.1.2 Kommerzielle Software-Entwicklung basierend auf OSSD-Methoden* diskutiert werden.
Die Aufgabenbereiche dieser Rollen sind nicht statisch einem Akteur zuweisbar, sondern können verschiedenen Akteuren zugewiesen werden. Diesbezüglich ist es denkbar und z.T. auch notwendig, dass ein Akteur mehrere Rollen ausübt. Andererseits ist es auch möglich, nach Erreichen einer kritischen Projektgröße die Aufgabenbereiche einer Rolle auf mehrere Akteure zu verteilen.

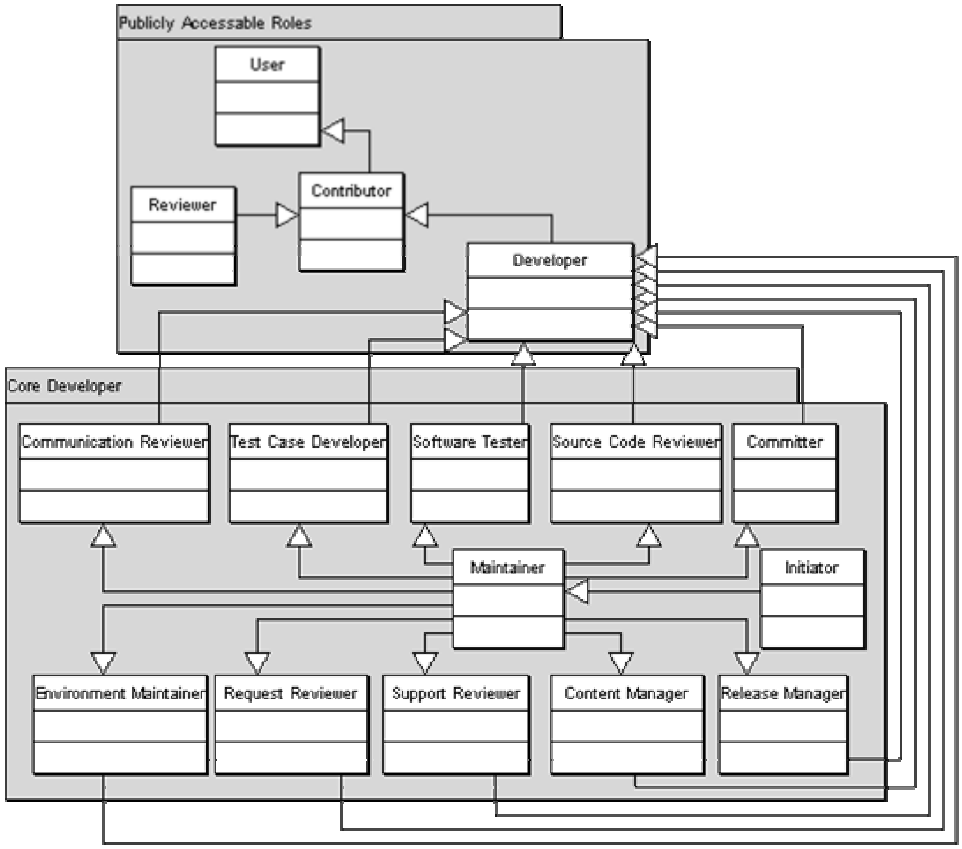

Abb. 7.8: Rollenstruktur im erweiterten Prozessmodell

7.2 Erweiterung und Modifikation des Artefaktmodells

Dieser Abschnitt thematisiert die Erweiterung des identifizierten Artefaktmodells. Es werden dabei sowohl neue Artefakte vorgestellt, als auch Eigenschaften von bereits beschriebenen Artefakten erweitert, mit deren Etablierung primär die in *6.5 Zielstellungen für die Erweiterung und Unterstützung des OSSD-Modells* dargestellten Zielstellungen verfolgt werden. Gerade im OSSD-Kontext kommt den Artefakten eine besondere Bedeutung zu, da die Entwicklungsprozesse kaum direkt reglementier- und formalisierbar sind, sondern vor allem durch die Eigenschaften der entwickelten und verwendeten Artefakte determiniert und gesteuert werden. Die Definition von expliziten Artefakteigenschaften kann z.B die adäquate Unterstützung der dargestellten Prozesse durch die Artefakte und die Automatisierung zentraler Prozesse ermöglichen.

Daher ist ein wichtiges Ziel, welches mit der Optimierung des identifizierten Artefaktmodells verfolgt wird, die Unterstützung und Ermöglichung einer weitgehend selbstregulierenden Projektkoordination durch die Definition neuer Artefakte oder die Spezifikation geeigneter Metadaten und darauf basierender Lebenszyklen und Workflows. Die Spezifikation der jeweiligen Artefaktcharakteristika stellt zudem die Basis für die anschließende softwaretechnische Realisierung und Verwaltung dieser Artefakte dar.

7.2.1 Überblick über Artefakte und deren Verwendung

In der folgenden Matrix wurden sowohl die im Rahmen der Fallstudien identifizierten Artefake des deskriptiven Prozessmodells als auch die modifizierten bzw. zusätzlich eingeführten Artefakte aufgeführt, welche als

Erweiterung des deskriptiven Prozessmodells betrachtet werden können. Diesen Artefakten wurden die verschiedenen Prozesse, sowohl die erweiterten als auch die bereits im deskriptiven Modell identifizierten, gegenübergestellt, um für jedes Artefakt definieren zu können, in welchen Prozessen dieses Artefakt erstellt, modifiziert, gelöscht oder verwendet wird.

Tab. 7.2: Artefakte und ihre Prozessbeziehungen

		Erweiterte Prozesse [Core Developers]									Prozesse des deskriptiven Prozessmodells								
		Source Code Review	Software Test	Test Case Development	Content Artifact Management	Communication Review	Request/Issue Review	Extended Release Process	Erweitere Managementprozesse	Erweiterte infrastructure Prozesse	Software Deployment	Software Test	Anwender- und Entwicklersupport	Beitrag von Request Artifacts	Kollaborativer Request Review	Patchentwicklung	Entwicklung von Dokumentationsartefakten	Managementprozesse	Infrastrukturelle Prozesse
Management-artefakte	Softwarelizenz (OSD)								**m**		**u**					**u**		**m**	
	Prozessguidelines	**u**	**u**	**u**	**u**	**u**	**u**	**u**	**m**		**u**	**u**	**u**	**u**	**u**	**u**	**u**	**m**	
	Programmierkonventionen	**u**							**m**							**u**		**m**	
	Documentation Styleguide				**m**		**u**		**m**								**u**	**m**	
	Review- und Test Guide	**u**	**u**						**m**					**u**				**m**	
	Status File	**m**	**m**		**m**				**m**										
Technologische Artefakte	Patch	**u**	**u**					**u**				**u**				**m**	**u**		
	Allgemeines Content Artefakt				**u**				**u**		**u**	**u**			**u**		**m**	**m**	
	Dokumentationsartefakt	**u**			**u**				**u**		**u**	**u**					**m**	**m**	
	[Request Artefakte]																		
	Code Change Request	**u**	**u**			**m**	**m**							**m**	**m**	**u**	**u**		
	Support Request					**m**	**m**						**u**	**m**	**m**				
	Content Change Request				**u**	**m**	**m**							**m**	**m**		**u**		
	Maintenance Request					**m**	**m**		**u**	**u**				**m**	**m**			**u**	**u**
	[Software Releases]																		
	Nightly Build		**u**						**m**		**u**	**u**	**u**	**u**	**u**	**u**	**u**		
	Alpha Release	**u**	**u**	**u**	**u**		**u**		**m**		**u**	**u**	**u**	**u**	**u**	**u**	**u**	**m**	
	Beta Release	**u**	**u**	**u**	**u**		**u**		**m**		**u**	**u**	**u**	**u**	**u**	**u**	**u**	**m**	
	GA Release	**u**	**u**	**u**	**u**		**u**		**m**		**u**	**u**	**u**	**u**	**u**	**u**	**u**	**m**	

u: usage (Prozess verwendet Artefakt)
m: modification (Prozess verwendet Artefakt und kann auch Modifikation, Erstellung oder Löschung beinhalten)

In der Matrix wurde ein Artefakt, sofern es in einem Prozess lediglich verwendet, nicht aber modifiziert wird, mit einem *u* (usage) markiert. Im Gegensatz dazu, wurden Artefakte, die in einem bestimmten Prozess auch modifiziert werden können, mit einem *m* (modification) markiert. Eine Modifikation kann die Erstellung, Löschung oder Veränderung eines Artefakts oder einer spezifischen Eigenschaft eines Artefakts beinhalten.

In diesem Kapitel werden lediglich die verschiedenen Ausprägungen der neu eingeführten Request-Artefakte und die Eigenschaften des erweiterten, dezidierten Content-Artefakts dargestellt, während weiterführende Informationen zu anderen Artefakten, die neu eingeführt bzw. deren Eigenschaften modifiziert wurden, im Anhang *C Erweiterte bzw. optimierte Artefakte* enthalten sind.

7.2.2 Request-Artefakte

Die sogenannten Change Requests stellen die elementaren Anforderungsartefakte der Entwicklungsprozesse dar und wurden im deskriptiven Prozessmodell bereits umfassend beschrieben. In vielen OSSD-Projekten hat sich die die indirekte, artefaktzentrierte Koordination und Steuerung der Entwicklungsprozesse durch die Change Requests und die Definition geeigneter Metadaten bewährt, was dem OSSD-Entwicklungsprozess auch die Charakterisierung als *bug driven* eingebracht hat (vgl. [Dietze03]). Daher könnte neben den Change Requests bezüglich des Quellcodes prinzipiell jegliche erforderliche Aktivität in Form eines ähnlichen, metadatenbasierten Artefakts verwaltet werden, um die zentrale Dokumentation und Koordination aller Anforderungen und Aktivitäten eines OSSD-Projekts zu ermöglichen. Dadurch wird die Definition verschiedener Ausprägungen eines Requests erforderlich, welche eine Beschreibung der Kern-Entwicklungsaktivitäten eines OSSD-Projekts ermöglichen:

- Code Change Request
- Content Change Request
- Maintenance Request
- Support Request

Diese einzelnen Ausprägungen dienen der Beschreibung der verschiedenen Anforderungen, welche im Lauf des OSS-Entwicklungsprozesses anfallen und werden im folgenden noch explizit dargestellt. Analog zum deskriptiven Prozessmodell sollte die Verwaltung dieser Artefakte über ein entsprechendes Bug Tracking System erfolgen. Dies ermöglicht die strukturierte Definition und Verwaltung aller möglichen Aktivitäten und Aufgaben und erweitert die Funktionalitäten des Bug Tracking Systems zur gesamtheitlichen Verwaltung aller Tasks und Aktivitäten zu einem allgemeinen Request Tracking System mit zentralen Projektkoordinationsfunktionalitäten. Eine derartige softwaretechnische Unterstützung wird bereits als Prämisse für die weitere Definition der Request-Artefakte und der assoziierten Metadaten angenommen, da vor allem für die Nutzung durch große Communities eine strukturierte, SW-unterstützte und metadatenbasierte Verwaltung aller Informationen notwendig ist (vgl. [KiLe00][101]).

Durch die Standardisierung dieser Artefakte und deren strukturierter Beschreibung mit Hilfe von zentral verwalteten Metadaten kann die Transparenz über die dezentralen Prozesse verbessert werden. Dies trägt dazu bei, die redundante und inkonsistente Verwaltung und Definition von entsprechenden Informationen zu reduzieren, wodurch die typischerweise unstrukturierte und inkosistente Informationslogistik in OSSD-Projekten verbessert werden kann.

Generische Metadaten
Die verschiedenen Spezialisierungen des Request-Artefakts verfügen zum einen über spezifische Metadaten. Gleichzeitig besitzen sie aber auch gemeinsame Metadaten, die von allgemeiner Gültigkeit für alle Ausprägungen sind. Diese sollten die folgenden Metainformationen umfassen:

- *ID:* Eindeutiger Identifikator (durch das System vergeben)
- *Name:* Bezeichner des Requests
- *Summary:* Kurzbeschreibung
- *Owner:* Initiator des Request
- *Owner Mail:* E-Mail Adresse des Initiators
- *Date:* Datum der Erstellung
- *Priority:* Priorität der Bearbeitung
- *Assignee:* Bearbeiter des Requests
- *Assignee Mail:* E-Mail Adresse des Bearbeiters

[101] „With a large user community, typical for open-source products, structured feedback is essential." [KiLe00]

- *Category:* Art des Requests (Code- oder Content Change-, Support-, Maintenance Request)
- *Comments:* Kommentare primär zur Diskussion des Request durch die Community
- *Keywords:* Schlüsselwörter zur Suche
- *Attachments:* Container für ergänzende Dokumente

Sofern nicht zwingend eine Freitext-Eingabemöglichkeit für den Anwender ermöglicht werden muss, sollten die möglichen Ausprägungen der Attribute bereits implizit als Auswahloption definiert und somit deren Vergabe reglementiert werden. Auf die genauen Ausprägungen und Definitionen der einzelnen Attribute wird im folgenden nur dann eingegangen, wenn diese Merkmale grundsätzlich von den im deskriptiven Prozessmodell definierten abweichen.

Alle Metadaten werden, abgesehen von der *ID* des Requests, durch die Akteure vergeben, wobei verschiedene Attribute zu bestimmten Zeitpunkten nur durch bestimmte Akteure modifiziert werden sollten (vgl. *7.2.2.1 Code Change Request*). Durch die Definition von Zugangsbeschränkungen und die dedizierte Rechtevergabe bezüglich der jeweiligen Artefakte in Abhängigkeit von ihrem Zustand oder Status ist es möglich, rollenbasierte Workflows zu realisieren. Das Attribut *Status* repräsentiert das zentrale Attribut zur Abbildung des Lebenszyklus und wurde nicht im Rahmen der gemeinsamen Metadaten definiert, da es im Kontext der diversen Artefaktausprägungen verschiedene Werte annehmen kann und somit in den folgenden Abschnitten explizit dargestellt wird. Durch die Möglichkeit der Abbildung der Request-Lebenszyklen mit Hilfe des *Status* Attributes übernimmt das System zur Verwaltung und Speicherung der Request-Artefakte indirekt auch Prozesskoordinierungsfunktionalitäten im Rahmen der Entwicklungsprozesse, da dadurch ermöglicht wird, den Zustand der auf den Request-Artefakten basierenden Entwicklungsaktivitäten zu dokumentieren. Somit steuert die Definition von komplexen Lebenszyklen bezüglich der Anforderungsartefakte indirekt auch die gesamten Erstellungs- bzw. Bearbeitungsprozesse des Requests und auch die der darauf basierenden (Software-)Artefakte (z.B. Patches).

7.2.2.1 Code Change Request

Im OSS-Kontext stellt es nicht notwendigerweise einen Nachteil dar, dass Softwareanforderungen nicht anhand formaler, modellbasierter Spezifikationsmethoden, wie z.B. durch UML-Modelle, definiert werden, da in diesem Zusammenhang Entwickler typischweise auch Anwender der Software sind und somit über ein besseres Verständnis der Anwendungsdomäne verfügen (vgl. [Scacchi01]). Zudem dienen die im sukzessiven Verbesserungsprozess eines OSSD-Projekts erfassten Software-Anforderungen nicht der Entwicklung und erschöpfenden Beschreibung der Anforderungen eines Softwareprototyps, sondern werden erst im Kontext einer schrittweisen, prototypbasierten Softwareverbesserung eingesetzt und müssen daher auch nicht den Prozess einer vollständigen Anforderungsanalyse unterstützen. Daher wurde von einer umfassenden Erweiterung der typischen Change Requests von OSSD-Projekten abgesehen und mit dem in diesem Abschnitt beschriebenen Code Change Request lediglich eine erweiterte und optimierte Variante des traditionellen Change Request des deskriptiven Modells entwickelt. Dieser beschreibt Systemverbesserungsanforderungen bezüglich des Entwicklungsquellcodes oder eines spezifischen Software-Release. Die strukturierte (metadatenbasierte) Definition der Anforderungen stellt dabei eine wichtige Anforderung dar, die noch nicht in allen OSS-Projekten erfüllt wird, da dadurch die umfassende Recherche ermöglicht und die Bereitstellung aller notwendigen Informationen sichergestellt wird (vgl. [KiLe00][102]).

Ergänzende Metadaten

Für Code Change Requests sollten zusätzlich zu den generischen Metadaten aller Request-Artefakte die folgenden ergänzenden Metadaten definiert werden, welche z.T. denen des Change Requests des deskriptiven Modells entsprechen (vgl. [Dietze03], [Dietze03c]:

[102] „In addition, it is much easier for developers to rapidly find what they are looking for in a bug report when it is structured." [KiLe00]

- *Status:* Status der Bearbeitung zur Abbildung eines Lebenszyklus
- *Software Version:* Verweis auf das zugrundeliegende Software Artefakt (mögl. Ausprägungen: Datum des Checkout des Entwicklungsquellcodes bzw. Bezeichner des jeweiligen Software-Release)
- *Modul:* Betroffenes Software Modul
- *Platform:* Verwendete Hardwareplattform des Owners
- *OS:* Betriebssystem des Owners
- *Severity:* Auswirkungen eines Software-Fehlers bzw. Enhancement
- *Patch:* Container für das entwickelte Patch

Das zentrale *Status*-Attribut wird in diesem Kontext durch verschiedene Metadaten ergänzt, welche der Beschreibung der Software dienen, die der definierten Änderungsanforderung zugrunde liegt. Außerdem wurde ein spezifischer Attachment-Container vorgesehen, in welchem ein entwickeltes Patch direkt mit dem Anforderungsartefakt verknüpft werden kann.

Lebenszyklus
Das Attribut *Status* repräsentiert das zentrale Attribut zur Dokumentation und Koordinierung der Entwicklungsprozesse und kann verschiedene, definierte Ausprägungen annehmen, wodurch es den Lebenszyklus des Anforderungsartefakts und indirekt auch den des darauf basierenden Patches beschreibt. Die folgenden möglichen Ausprägungen beschreiben die verschiedenen, im Rahmen der Entwicklungsprozesse identifizierten Entwicklungsphasen und sollten für dieses Attribut zur Abbildung und Unterstützung der realisierten Prozesse vorgesehen werden:

- *New:* Initialer Status nach der Generierung
- *Verified:* Verifikation des Requests durch dezidierten Akteur durchgeführt
- *Not Verified:* Verifikation nicht möglich
- *Assigned:* Individueller Entwickler implementiert eine Lösung
- *Patch Review:* Patch wurde entwickelt und dem Reviewprozess zugeführt
- *Patch Approved:* Patch wurde durch Reviewer validiert
- *Patch Committed:* Patch wurde durch Committer in den Entwicklungsquellcode integriert
- *Patch Released:* Patch wurde als Bestandteil eines Software-Release oder separat publiziert

Zur konkreten Realisierung des Patchentwicklungsprozesses und zur Qualitätssicherung, sollten einige der dargestellten Ausprägungen nur durch entsprechend privilegierte Akteure vergeben werden können, was durch eine explizite Rechtevergabe des Request Tracking Systems unterstützt werden muss. Im folgenden Modell wurde ein erweiterter Lebenszyklus eines Code Change Requests in der Zustandssicht des Metamodells modelliert, in dem die Abhängigkeit zwischen der Rollenzugehörigkeit eines Akteurs und der Möglichkeit zur Zustandsänderung des *Status*-Attributs dargestellt wurde.

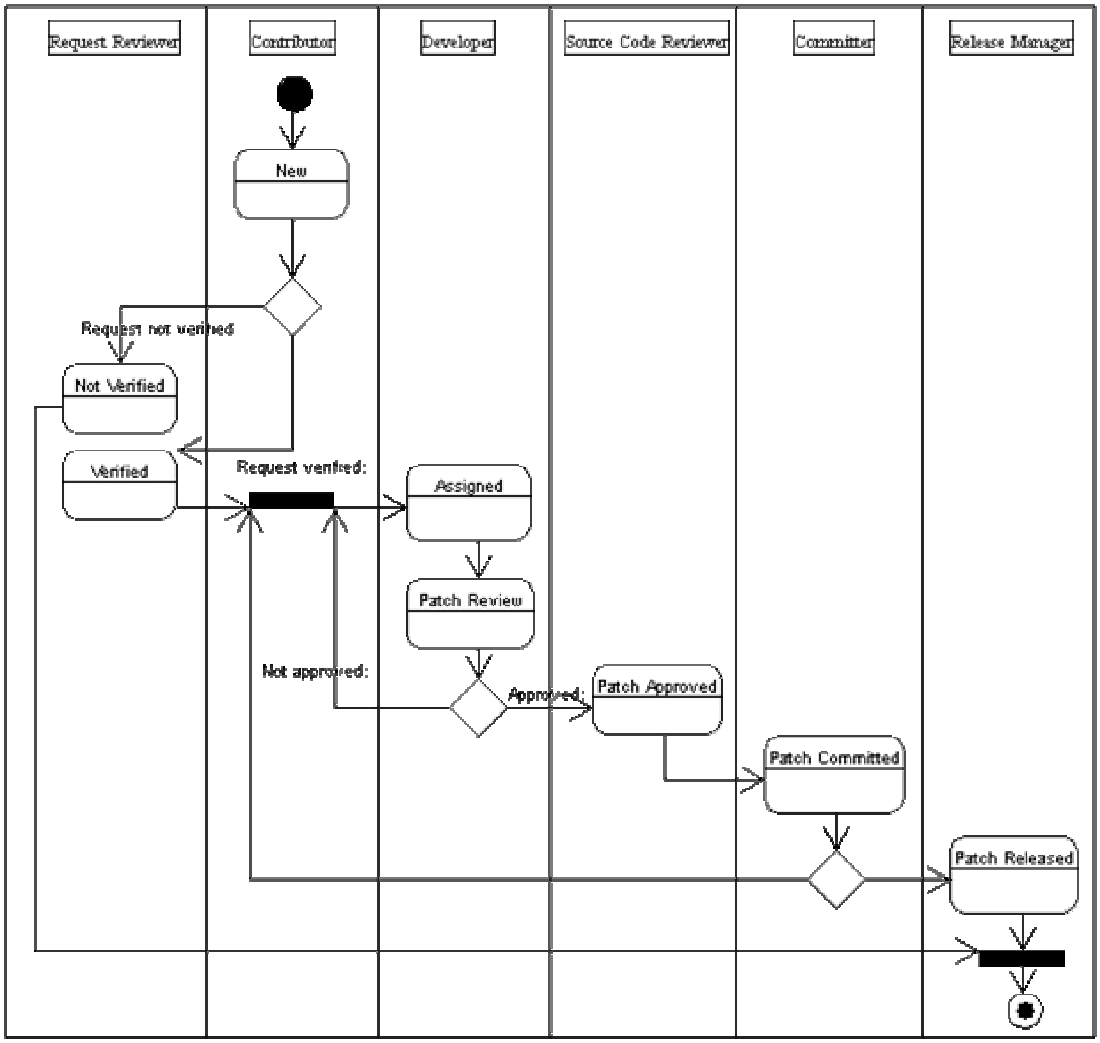

Abb. 7.9: Modifizierter Lebenszyklus eines Code Change Request

Rollenbasierter Lebenszyklus

Die minimale Rolle, welcher ein Akteur angehören muss, um den Zustandsübergang eines Artefakts in dessen Lebenszyklus durch die Zuweisung einer neuen Ausprägung des Status-Attributs durchzuführen, wurden in diesem Modell durch Swimlanes der UML dargestellt. Mit der initialen Definition eines Code Change Requests wird automatisch der Status *New* vergeben. Nach dem bereits dargestellten Prozess des Requirements Review können durch den Requirements Reviewer entsprechend des Resultats die Stati *Verified* bzw. *Not Verified* vergeben werden. Bei der anschließenden Selektion des Anforderungsartefakts durch einen Entwickler dokumentiert er dies durch die Transition in den Status *Assigned* und vergibt nach erfolgter Entwicklung und Kommunikation des Patches den Status *Patch Review*, wodurch alle Akteure und speziell die Source Code Reviewer zur Evaluierung des Patches aufgefordert werden. Nach erfolgreichem Review werden die Committer über die Transition in den Zustand *Patch Approved* zur Integration in den zentralen Entwicklungsquellcode aufgefordert. Der erfolgte Commit wird durch die Zuweisung der Ausprägung *Patch Committed* dokumentiert. Der finale Status wird durch den Zustand *Patch Released* repräsentiert, der ausschließlich durch den Release Manager definiert werden kann. Nachdem ein Patch durch den Release Manager als Bestandteil eines Software-Release oder als explizites Update-Patch veröffentlicht wurde, können Änderungen des Patch-Quellcodes nur noch

durch die Definition eines neuen Code Change Request angeregt werden. Daher ist nach dem Release die Zurückstufung des Status eines Anforderungsartefakts nicht mehr möglich.

Desweiteren sollten auch die Modifikationsrechte verschiedener anderer Attribute in Abhängigkeit von den verschiedenen Ausprägung des *Status*-Attributs betrachtet werden. Das Attribut *Comments* sollte beispielsweise durch jeden Akteur und unabhängig vom jeweils aktuellen Status des Artefakts modifiziert oder zumindest erweitert werden können. Im Gegensatz dazu sollten alle anderen Attribute nur durch Akteure modifiziert werden können, die auch das Privileg zur Transition in den nächsten Zustand besitzen. Desweiteren sollten einige Attribute nach ihrer initialen Definition über den gesamten Lebenszyklus hinweg unverändert bleiben, was z.B. auf die Attribute *Name*, *Owner*, *Date* und *Owner Mail* zutrifft.
Durch die Implementierung eines derartigen rollenbasierten Lebenszyklus können Rollen, Prozesse und Workflows indirekt und artefaktbasiert abgebildet und somit softwaretechnisch unterstützt werden.

7.2.2.2 Content Change Request

Analog zu den Code Change Requests sollten dezidierte Artefakte zur Definition von Anforderungen bezüglich der Modifikation oder Erstellung eines Dokumentationsartefakts definiert werden. Dieses Anforderungsartefakt sollte ebenfalls über verschiedene ergänzende Metadaten verfügen:

- *Status:* Status der Bearbeitung; dient zur Abbildung eines Lebenszyklus
- *Software Version:* Verweis auf das zugrundeliegende Software Artefakt (Ausprägungen: Datum des Checkout des Entwicklungsquellcodes bzw. Bezeichner des jeweiligen Software-Release)
- *Modul:* Betreffendes Software Modul
- *Plattform:* Verwendete Hardwareplattform des Owners
- *OS:* Betriebssystem des Owners
- *Content Artifact:* Verweis auf das entsprechende Content Artifact

Vom letzten Attribut abgesehen, orientieren sich diese Metadaten an denen eines Code Change Request und dienen der Beschreibung der Software-Artefakte auf welches sich das Content-Artefakt bezieht. Falls diesbezüglich keine direkten Abhängigkeiten bestehen, ist eine Definition dieser Attribute obsolet. Das Attribut *Content Artifact* kann als Container für das erstellte Dokumentationsartefakt verwendet werden oder auf ein entsprechendes Artefakt verweisen.

Analog zum Code Change Request sollte ein rollenbasierter Lebenszyklus durch die Definition geeigneter Ausprägungen für das Attribut *Status* definiert werden und die spezifizierten Prozesse durch eine dedizierte artefaktbezogene Rechtevergabe unterstützt werden. Aus Platzgründen wird von einer Modellierung aller Lebenszyklen abgesehen und auf dessen exemplarische Darstellung im Kontext der Code Change Requests verwiesen.

7.2.2.3 Support Request

Um eine konsistente und strukturierte Verwaltung von Supportanfragen zu ermöglichen und den redundanten und intransparenten Informationsfluss zu vermeiden, der in diesem Kontext im deskriptiven Prozessmodell dargestellt wurde, sollten Supportanfragen analog zu anderen Anforderungen als strukturiertes, metadatenbasiertes Artefakt im Request Tracking System verwaltet werden. Dies sichert die Nachhaltigkeit der Supportaktivitäten, vereinfacht die Recherche in bereits bearbeiteten Supportfällen und ermöglicht die konsequente Kontrolle über den Status der Bearbeitung einzelner Anfragen. Damit stellt es die Voraussetzung für den erweiterten Prozess des Review von Support Requests dar (vgl. *7.1.3.1 Koordination des Support*). Daher sollte ein entsprechendes Artefakt über die folgenden ergänzenden Metadaten verfügen.

- *State:* Status der Bearbeitung; dient zur Abbildung eines Lebenszyklus

- *Software Version:* Verweis auf das zugrundeliegende Software Artefakt (Ausprägungen: Datum des Checkout des Entwicklungsquellcodes bzw. Bezeichner des jeweiligen Software-Release)
- *Modul:* Betreffendes Software Modul
- *Plattform:* Verwendete Hardwareplattform des Owners
- *OS:* Betriebssystem des Owners
- *Solution:* Container für die Informationen zur Problemlösung (enthält u.U. Verweis auf Content Artifact)

Abgesehen vom letzten Attribut orientieren sich diese Metadaten wieder an den Metadaten der Code Change Requests. Das Attribut *Solution* kann dazu verwendet werden, die Lösung zu dem definierten Problem beizufügen oder auf ein entsprechendes Dokumentationsartefakt zu verweisen, welches eine Lösung für das Problem enthält.

7.2.2.4 Maintenance Request

Analog zu den beschriebenen Anforderungsartefakten sollten auch allgemeine Anfragen bezüglich der Maintenance des Projekts in einem entsprechendem System strukturiert und konsistent verwaltet werden, um die Bearbeitung durch die Maintainer und Core Developer zu vereinfachen und die Transparenz über die zentralen Aktivitäten der Core Developer und Maintainer zu verbessern. Bezüglich dieses Artefakts sind vor allem zwei spezifische Attribute notwendig.

- *Status:* Status der Bearbeitung; dient zur Abbildung eines Lebenszyklus
- *Type:* Kategorie der Anfrage (Ausprägungen: *Infrastructure*, *Project Organisation*, *Process Coordination*)

Das Attribut *Type* dient der Spezifikation einer Kategorie, welcher das Artefakt zugeordnet werden soll. Ein Maintenance Request kann somit eine infrastrukturelle Anforderung, eine organisatorische Aktivität oder eine koordinierende Aktivität beschreiben und adressiert direkt die Maintainer oder die Core Developer des Projekts.

7.2.3 Content-Artefakte

In diesem Abschnitt werden die Content-Artefakte beschrieben, welche alle Informationsobjekte beinhalten sollten, die im Kontext eines OSS-Projekts von Bedeutung sind und projektrelevante Informationen beinhalten. Primär sind dies die Inhalte, welche über die Website eines Projekts publiziert werden (Web-Content) und die Dokumentationsartefakte der OSS. Die Verwaltung jeglicher Inhalte in einem dezidierten Software-System, respektive einem Document bzw. Content Management System, erscheint sinnvoll und ermöglicht die artefaktbezogene Definition von Metadaten und die Realisierung von Lebenszyklen und Workflows. Dies ermöglicht die kollaborative Erstellung und Modifikation jeglicher Informationsobjekte und erscheint gerade im OSS-Kontext von besonderer Bedeutung (vgl. [ErHaSc03]).

Metadaten

Die Definition von Metadaten zur Verwaltung zentraler und kollaborativ erstellter und genutzter Artefakte ermöglicht die umfassende Recherche nach verschiedenen Attributausprägungen und damit eine bessere Kontrolle über den Entwicklungsprozess. Durch die strukturierte Beschreibung aller Inhalte der Website und aller Informationsobjekte des Projekts ist es möglich, kollaborative Entwicklungsprozesse für diese Informationsobjekte analog zu den dargestellten Patchentwicklungsprozessen zu etablieren und ähnliche Lebenszyklen zu implementieren. Die folgenden Metadaten werden als adäquate Ergänzung für alle Informationsobjekte, inklusive des Website Content und der Dokumentationsartefakte betrachtet:

- *ID:* Eindeutiger Identifikator (durch das System zugewiesen)
- *Name:* Bezeichner des Informationsobjekts
- *Owner:* Initialer Autor des Artefakts

- *Status:* Status des Artefakts
- *Description:* Kurzbeschreibung des Inhalts
- *Keywords:* Schlüsselwörter zur Recherche
- *Date:* Datum der initialen Erstellung
- *Co-Authors:* Liste aller an der Aktualisierung des Artefakts beteiligter Autoren
- *Last Update:* Datum der letzten Aktualisierung
- *Type:* Typ des Artefakts (Ausprägungen z.B.: *User Documentation*, *Developer Documentation*, *Source Code Documentation*)
- *Format:* Format des Documents

Lebenszyklus
Auch in diesem Zusammenhang erscheint es sinnvoll, einen Lebenszyklus basierend auf den verschiedenen Ausprägung des *Status*-Attributs zu definieren. Dabei erscheint die Definition mindestens der folgenden Stati sinnvoll:

- *New:* Artefakt wurde kommuniziert
- *Approved:* Artefakt wurde verifiziert
- *Published:* Artefakt wurde publiziert
- *Update:* Aktualisierung erforderlich

Diese Stati dienen der Dokumentation des Zustands aller Content-Artefakte und stehen in direktem Zusammenhang zu den Stati des in *7.2.2.2 Content Change Request* definierten Lebenszyklus eines Content Change Request. Auch hier sollten einzelne Stati idealtypischerweise nur durch entsprechend autorisierte Akteure in Abhängigkeit von ihrer Rollenzugehörigkeit definiert werden können. Beispielsweise sollte der Status *Approved* sollte beispielsweise nur durch die bereits beschriebene Rolle des Content Manager zugewiesen werden.

Die Dokumentationsartefakte, die zur Dokumentation von Quellcode-Artefakten dienen, werden als spezifische Ausprägung der Content-Artefakte betrachtet und in Anhang *C.2.2 Dokumentationsartefakte* dargestellt.

7.2.4 Beziehungen der technologischen Artefakte

In der folgenden Grafik wurden das erweiterte Artefaktkonzept der technologischen Artefakte basierend auf der Artefaktsicht des Metamodells (vgl. [Dietze03b]) dargestellt. Aus Gründen der Komplexitätsverringerung wurde auf die Integration der Managementartefakte in das folgende Modell verzichtet, da zwischen den Managementartefakten i.d.R. kaum bzw. weniger bedeutende Beziehungen existieren.

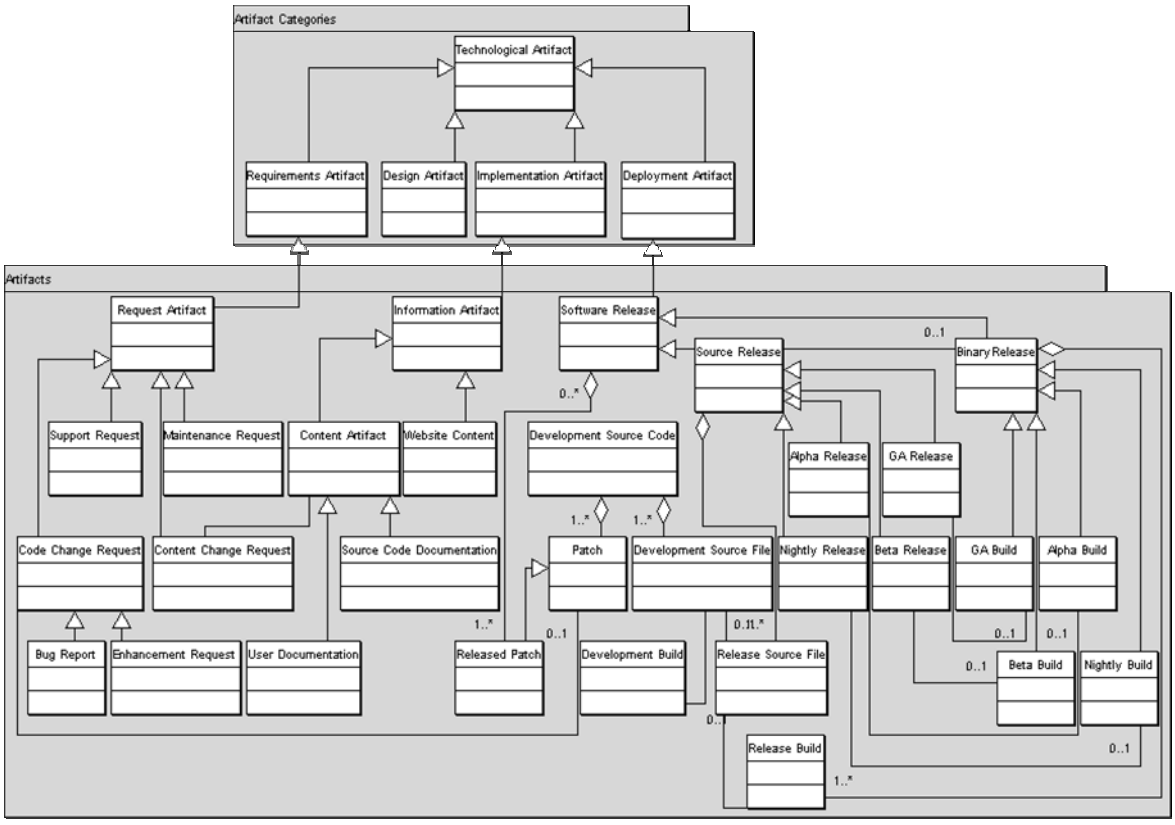

Abb. 7.10: Artefaktkonzept der technologischen Artefakte

Auffällig an obiger Grafik ist die vollständige Abwesenheit von expliziten Design-Artefakten. Diese konnten im Rahmen des deskriptiven OSSD-Modells nicht identifiziert werden und wurden aufgrund der Charakterisierung des OSSD-Ansatzes als ausschließlicher Softwareverbesserungsprozess auch nicht als expliziter Bestandteil eines erweiterten Artefaktkonzepts vorgesehen. Trotzdem erscheint eine Verwendung von Entwurfsartefakten vor allem durch die Core Developer im Rahmen von umfangreicheren Implementationsvorhaben sinnvoll.

7.3 Unterstützende Software-Infrastruktur

Die bisher identifizierten Softwarefunktionalitäten, die in vielen OSS-Projekten in vergleichbarer Weise realisiert werden, haben zu einem projektübergreifenden Defacto-Konsens über im OSS-Kontext eingesetzte Infrastrukturen geführt. Trotzdem existiert ein erheblicher Optimierungsbedarf der typischerweise im OSSD instrumentalisierten Software-Werkzeuge, da diese lediglich eine rudimentäre Unterstützung der verteilten Entwicklungsprozesse ermöglichen und Potential zur erweiterten softwarebasierten Automatisierung und Unterstützung der Prozesse beobachtet werden kann (vgl. [AnHeSy03][103]). Dies muss insbesondere eine möglichst ganzheitliche, transparente, redundanzfreie und konsistente Verwaltung der Artefakte und eine gesamtheitliche Integration aller notwendigen

[103] „[...] there are a number of ways in which open source development environments could be improved, particularly with respect to supporting collaboration among developers, supporting new and potential contributors, and in bringing users and developers together.“ [AnHeSy03]

Softwarefunktionalitäten ermöglichen (vgl. [Weber92][104]). Die dargestellten Charakteristika des generalisierten OSSD-Prozessmodells und die in *6.5 Zielstellungen für die Erweiterung und Unterstützung des OSSD-Modells* dargestellten Optimierungszielstellungen, ermöglichen daher die Definition verschiedener Softwarefunktionalitäten, welche die dargestellten Prozesse besser abbilden und unterstützen.

In diesem Abschnitt werden daher unterstützende Softwarefunktionalitäten thematisiert, ein möglicher Ansatz zur Implementierung einer derartigen Software-Infrastruktur vorgestellt und dessen Potentiale zur Abbildung und Unterstützung des OSSD-Modells erläutert.

7.3.1 Überblick: Softwarefunktionalitäten und unterstützte Prozesse bzw. Anforderungen

Die folgenden Softwarefunktionalitäten wurden basierend auf den in *6.5 Zielstellungen für die Erweiterung und Unterstützung des OSSD-Modells* vorgestellten Zielstellungen hergeleitet und sollten durch eine Software-Infrastruktur möglichst integriert und gesamtheitlich zur Verfügung gestellt werden:

1. Dedizierte Kommunikationskanäle
2. Gesamtheitliches, metadatenbasiertes Artefaktmanagement
    - 2.1.1 Konfigurationsmanagement (Quellcodeverwaltung)
    - 2.1.2 Gesamtheitliches Request Tracking
    - 2.1.3 Verknüpfung verwandter Artefakte
    - 2.1.4 Content- und Document Management
3. Einheitliche, webbasierte Benutzerschnittstelle
4. Workflow Management
    - 4.1.1 Softwaretechnische Rollenimplementierung (Rechte)
    - 4.1.2 E-Mail-Integration
    - 4.1.3 Artefakt-Routing (Core Developers)
    - 4.1.4 Release-Automatisierung

In der folgenden Grafik werden die bedeutendsten Elemente bzw. Funktionalitäten einer gesamtheitlichen Software-Infrastruktur im Kontext der erweiterten Prozesse, der infrastrukturellen Anforderungen und der unterstützten Artefakte betrachtet, um darzustellen, inwiefern eine derartige Software-Infrastruktur das gesamtheitliche Prozessmodell unterstützen kann.

[104] „Wird dennoch angestrebt, den gesamten Software-Entwicklungsvorgang durch Werkzeuge zu unterstützen, sind Integrationen dieser Werkzeuge notwendig, [...]. Die Gesamtheit der Werkzeuge wird damit über eine einheitliche Benutzerschnittstellen zugänglich und kann über eine gemeinsame Datenhaltungskomponente die Interaktion zwischen Softwareentwicklern [...] unterstützen.“ [Weber92]

Tab. 7.3: Softwarefunktionalitäten und jeweilig unterstützte Anforderungen, Prozesse und Artefakte

		Erweiterte Softwarefunktionalitäten											
		1. Dezidierte Kommunikationskanäle	2. Gesamtheitl. Artefaktmanagement	2.1 Konfigurationsmanagement (Quellcode)	2.2 Gesamtheitl. Request-Tracking	2.3 Artefaktverknüpfung	2.4 Content- und Document Management	3. Einheitl., webbasiertes UI	4. Workflowmanagement	4.1 Softwaretechn. Rollenimplementierung	4.2 E-Mail-Integration	4.3 Artefakt-Routing	4.4 Release-Automatisierung
Erweiterte Prozesse (Core Developers)	Source Code Review	x		x	x			x	x	x	x	x	
	Software Test	x		x	x			x	x	x	x	x	x
	Test Case Development	x			x			x	x	x	x	x	
	Content Artifact Management	x			x		x	x	x	x	x	x	
	Communication Review	x						x	x	x	x	x	
	Request/Issue Review	x			x			x	x	x	x	x	
	Extended Release Process			x				x	x	x	x	x	x
	Infrastructure Maintenance	x			x			x	x	x	x	x	
Infrastrukturelle Zielstellungen	Unterstützung paralleler Prozesse	x		x	x		x	x	x	x	x	x	x
	Redundanzfreie, konsistente Artefakte	x		x	x		x	x	x			x	
	Steigerung der Prozess-Transparenz	x	x	x	x	x	x	x			x		
	Unterstützung Rollen- und Prozessmodell	x		x	x		x	x	x	x	x	x	x
	Dedizierte Unterstützung der Core Developer	x			x		x	x	x	x	x	x	x
	Minimierung Abhängigkeiten (Core Developer)	x		x	x		x	x	x	x	x		x
	Unterstützung Entscheidungsfindungsprozesse				x		x	x	x			x	
Prozesse des deskriptiven Prozessmodells	Software Deployment	x		x					x	x	x		x
	Software Test	x		x	x			x	x	x	x		x
	Anwender- und Entwicklersupport	x			x			x	x	x	x		
	Beitrag von Request Artifacts	x			x			x	x	x	x		x
	Kollaborativer Request Review	x			x			x	x	x	x		
	Patchentwicklung	x		x	x			x	x	x	x		
	Entwicklung von Dokumentationsartefakten	x			x		x	x	x	x	x		
	Managementprozesse	x		x	x		x		x	x	x		
	Infrastrukturelle Prozesse	x			x				x	x	x		
Technologische Artefakte	Patch			x				x			x	x	
	Allgemeines Content Artefakt						x	x				x	
	Dokumentationsartefakt						x	x				x	
	Code Change Request				x			x	x	x	x	x	
	Support Request				x			x	x	x	x	x	
	Content Change Request				x			x	x	x	x	x	
	Maintenance Request				x			x	x	x	x	x	
	Nightly Build			x				x				x	x
	Alpha Release			x				x				x	x
	Beta Release			x				x				x	x
	GA Release			x				x				x	x

In der Tabelle wurden die zentralen Softwarefunktionalitäten aufgeführt, die durch eine optimierte Software-Infrastruktur zur Unterstützung von OSSD-Projekten implementiert werden sollten. Im Anhang *D Unterstützende Softwarefunktionalitäten* werden die verschiedenen Softwarefunktionalitäten und ihre Verwendung eingehender dargestellt, sofern die Charakteristika von denen der bereits im deskriptiven Modell [Dietze03c] dargestellten und bereits identifizierten Softwarefunktionalitäten abweichen.

7.3.2 Implementationsansatz - Implementierung des OSSD-Modells mit GENESIS

„Introducing the components of the GENESIS platform into an open-source project will, we claim, increase the productivity of the project by decreasing the management overheads of the project leaders, and by allowing the creation of a richer representation of the artefacts created and used by the project members." [BoLaNuRa03]

Die in Anhang *E.2 GENESIS* dargestellten Charakteristika der Software GENESIS ermöglichen den Einsatz der GENESIS-Software zur softwaretechnischen Implementierung und Abbildung des dargestellten OSSD Prozessmodells, was auch in [BoLaNuRa03][105] dokumentiert wird.

Status
Das GENESIS-Projekt befindet sich zum aktuellen Zeitpunkt noch in der Entwicklungsphase, wobei die Entwicklung über Sourceforge[106] durchgeführt wird. Momentan ist bereits ein Prototyp der Software (GENESIS Platform Version 1.0.1 Alpha) verfügbar (Stand: 04.11.2003), der in Java entwickelt wurde und einige zentrale Funktionalitäten zur Verfügung stellt. Dieser wurde unter der OSD-konformen Lizenz MPL 1.1 veröffentlicht und wird sukzessive im Rahmen eines OSSD-Projekts verbessert und erweitert. Da verschiedene Funktinalitäten, u.a. das Artefaktmanagement-System, im aktuellen Prototyp noch nicht verfügbar sind und somit nicht im Rahmen der momentan durchgeführten Test-Installation evaluiert werden können, wird im folgenden auch auf begleitende Dokumente zurückgegriffen, die den geplanten Leistungsumfang und die konzeptuelle Architektur der GENESIS-Software dokumentieren.

7.3.2.1 Softwaretechnische Implementierung der zentralen OSSD-Entitäten

In dem folgenden konzeptuellen Klassendiagramm wurde das Datenmodell der GENESIS-Software in einer konzeptuellen Sicht dargestellt:

[105] „Although it is initially targeted at development within commercial organisations, it can be useful to open-source software development." [BoLaNuRa03]

[106] URL: http://sourceforge.net/projects/genesis-ist

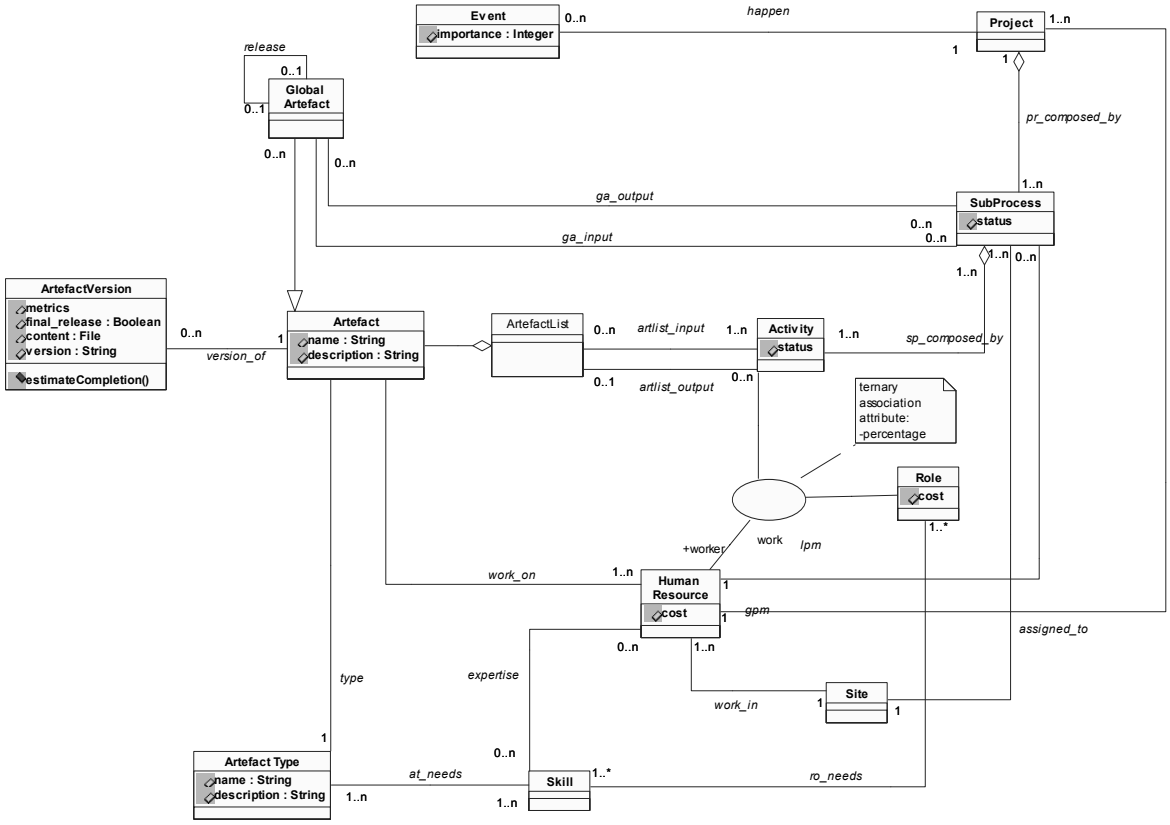

Abb. 7.11: Konzeptuelle Sicht des GENESIS-Datenmodells aus [GENESIS03][248]

Im dargestellten Datenmodell werden alle zentralen Entitäten des OSSD-Modells, respektive Rollen, Akteure (*Human Resource*), Artefakte, Artefakttypen, Teilprozesse und Aktivitäten als eigenständige, konzeptuelle Klassen definiert, die durch das GENESIS-System metadatenbasiert beschrieben und unterstützt werden. Die konzeptuelle Architektur von GENESIS unterstützt somit die Implementierung von Teilprozessen und Aktivitäten und die softwaretechnische Abbildung der im OSSD-Modell definierten Entitäten.

Durch die Möglichkeit der freien Definition von Rollen und Artfakttypen wird es ermöglicht, das in *7.1.5 Erweitertes Rollenmodell* dargestellte Rollenmodell und das in *7.2.4 Beziehungen der technologischen Artefakte* beschriebene Artefaktmodell softwaretechnisch zu realisieren und die einzelnen Typen durch Metadaten zu beschreiben. Die Definition konkreter *Artefact*- und *Human Resource*-Instanzen ermöglicht die anschließende Zuordnung des jeweiligen Artefakttyps bzw. der entsprechenden Rolle.

Verknüpfung von Artefakten
Wie in *E.2.2 Gesamtheitliches Artefaktmanagement - OSCAR* dargestellt wurde, können Artefakte basierend auf ihren Metadaten beliebig miteinander verknüpft werden und unterstützen dadurch einen gesamtheitlicheren Ansatz des Artefakt-Managements.

7.3.2.2 Modellierung der Prozesse

Über eine explizite Prozessmodellierungskomponente sind alle zu unterstützenden Prozesse beschreibbar und können nach ihrer initialen Definition und der anschließenden Instanziierung durch die Workflowkomponente dediziert unterstützt werden [BoLaNuRa03][249], wobei die genauen Möglichkeiten zur Workflowunterstützung mit

dem aktuellen GENESIS-Prototyp noch nicht getestet wurden, aber in [AvCiLuStVi03] und [GENESIS03a] erläutert werden. Im Kontext von GENESIS wird zwischen abstrakten Modellen (*abstract models*) und konkreten Modellen (*concrete models*) unterschieden. Während das abstrakte Modell dazu dient, Prozesse abstrakt auf modellebene und projektunabhängig zu modellieren, indem Aktivitäten und involvierte Rollen modelliert und erzeugte bzw. modifizierte Artefakttypen definiert werden, dient das konkrete Modell dazu, einen abstrakten Prozess zu instanziieren, also auch projektspezifisch zu konfigurieren und anzupassen (vgl. [AvCiLuStVi03][107]). Dies ermöglicht es beispielsweise, einmalig ein abstraktes Modell des OSSD in GENESIS zu modellieren, welches anschließend für den projektspezifischen Einsatz nur noch zu einem konkreten Modell instanziiert werden muss. Ein derartiges konkretes Prozessmodell kann anschließend direkt durch die Workflow Engine von GENESIS unterstützt werden [AvCiLuStVi03][108].

Die Dekomposition der Prozesse des OSSD-Modells wird in GENESIS durch die sogenannten *Super-Activities* unterstützt, welche eine hierarchische Dekomposition analog zur Prozessdekomposition des Metamodells (vgl. *2.2.1 Prozesse*) ermöglichen [AvCiLuStVi03][109]. Bei der Modellierung einzelner Aktivitäten wird im GENESIS-Kontext nicht konkretisiert, wie die Aktivitäten ausgeführt werden sollen, sondern nur, welche Artefakte erzeugt, modifiziert oder verwendet werden, um den Akteuren den vor allem im OSSD-Zusammenhang notwendigen Freiraum über die Art und Weise der Ausführung einzuräumen [AvCiLuStVi03][110].

7.3.2.3 Projektspezifische Implementierung

Die Implementierung der spezifischen Prozesse des OSSD-Modells wird direkt durch die Prozessdefinitionskomponente und die Möglichkeit zur freien Artefakt- und Rollendefinition unterstützt. Das abstrakte OSSD-Prozessmodell kann anschließend im Rahmen der Instanziierung zu einem konkreten Prozessmodell instanziiert und an die Anforderungen des jeweiligen Einsatzszenarios angepasst werden [BoSmWe03][111]. Die Instanziierung erfolgt über ein dediziertes Project Management Tool, welches es ermöglicht, z.B. Akteuren die Rollen des abstrakten Modells zuzuweisen oder Artefakt-Typen zu instanziieren [AvCiLuStVi03][112]. Im Rahmen der Konkretisierung des abstrakten Modells ist es auch möglich, Aktivitäten und Teilprozesse des abstrakten Modells zu modifizieren, zu löschen oder um neue zu ergänzen.

Schrittweise Implementierung

Durch die dargestellte Flexibilität ist es auch möglich, das in dieser Arbeit dargestellte Prozessmodell im Sinne einer *sanften Migration* schrittweise oder nur partiell einzuführen, um beispielsweise bestehende Prozesse und Aktivitäten durch die Implementierung der Software-Infrastruktur nicht zu tangieren [BoLaNuRa03][113]. Zudem ermöglicht der modulare Aufbau des Gesamtsystems die schrittweise Einführung einzelner Module des Gesamtsystems aufbauend auf dem Artefaktverwaltungs- und Konfigurationsmanagementsystem OSCAR [BoLaNuRa03][114]. Auch der konträre Ansatz, also die Implementierung der prozessunterstützenden Module im

[107] „[...] an abstract process model consists of an abstract description, temporal succession, iteration or parallelism of activities and artefacts production, independently of the particular project. This model is then refined for each project separately, thus achieving a concrete (or refined) process model.“ [AvCiLuStVi03]

[108] „The concrete process model is suitable for enactment by the workflow engine.“ [AvCiLuStVi03]

[109] „Super-activities allow a hierarchical decomposition of a process model.“ [AvCiLuStVi03]

[110] „[...] each activity of the process model is essentially described by the artefacts that will produce, and freedom is left to the worker(s) to decide how to actually perform the activity.“ [AvCiLuStVi03]

[111] „[...] it has been conceived following a service oriented approach facilitating extensibility and simplifying its tailoring to any specific organisation’s software process needs.” [BoSmWe03]

[112] „It implements the facilities for refining the abstract process model into a concrete process model with the information specific for the project (actual workers, actual artefact names, etc.).“ [AvCiLuStVi03]

[113] „As the GENESIS platform is process-aware, it can be adapted to support whatever process is in use by a project, rather than enforcing a process model in place of an already existing method. This allows for low-impact introduction of the system into a project.” [BoLaNuRa03]

[114] „The component nature of the platform allows progressive introduction of the platform: OSCAR can be introduced initially, in place of the configuration management system (such as CVS, from which data can be extracted) which is already in use.” [BoLaNuRa03]

Rahmen von Entwicklungsprojekten, die bereits ein Artefaktmanagementsystem verwenden, ist nach [BoLaNuRa03] möglich, wobei aber in diesem Fall nicht die Möglichkeiten eines gesamtheitlichen Artefaktmanagements und der semantischen Verknüpfung der Artefakte bestehen [BoLaNuRa03][115].

Die unterstützten Software-Werkzeuge zur dezentralen Artefakterstellung und -manipulation sind zudem durch GENESIS keinen Einschränkungen unterworfen, da GENESIS lediglich das zentrale Management der Artefakte übernimmt, nicht aber direkt deren Bearbeitungsprozesse beeinflusst. Somit sind auch alle identifizierten Entwicklungswerkzeuge weiterhin einsetzbar [BoSmWe03][116]. Zudem können prinzpiell alle Artefaktformate und –typen verwaltet werden.

7.3.2.4 Workflow Management

Die Workflow Engine von GENESIS interpretiert die intern verwendete Prozessbeschreibungssprache, die zur Modellierung der abstrakten und konkreten Modelle verwendet wird, und ist dadurch in der Lage, die definierten konkreten Prozessinstanzen zu interpretieren und zu unterstützen. Dazu erzeugt es Prozess- und Aktivitätsinstanzen basierend auf den Vorgaben und Restriktionen, die im konkreten Prozessmodell definiert wurden und weist diese den Akteuren entsprechend dem konreten Prozessmodell zu [AvCiLuStVi03][117]. Dabei ist es auch möglich, verschiedene Instanzen eines konkreten Prozessmodells, z.B. eines bestimmten Teilprozesses, parallel zu unterstützen [AvCiLuStVi03][118], was in OSSD-Projekten aufgrund der Prozessparallelisierung unabdinglich ist.

E-Mail Benachrichtigungen

Die auotmatisierte Benachrichtigung von Akteuren erfolgt über die sogenannte *Event Engine*, welche es den Akteuren und vor allem dem Maintainer ermöglicht, die gewünschten Benachrichtigungen selbständig zu konfigurieren [AvCiLuStVi03][119].

Bedingte, rollenbasierte Lebenszyklen

Durch die Verwendung von Variablen zur Definition von bedingten Aktititätsübergängen ist es möglich, auch bedingte Artefaktlebenszyklen (vgl. *7.2.2.1 Code Change Request*) zu implementieren, die eine Transition zwischen verschiedenen Zuständen nur basierend auf vorher definierten Bedingungen, z.B. entsprechenden Rollenzugehörigkeiten zulassen [AvCiLuStVi03][120].

7.3.2.5 Konfigurationsmanagement

Die Artefaktmanagementkomponente, welche zum aktuellen Zeitpunkt noch nicht vollständig implementiert wurde, verfügt nach [AvCiLuStVi03][121] auch über Synchronisationsmechanismen, wodurch eine kollaborative Arbeit an einer gemeinsam genutzten Artefaktbasis unterstützt wird. Dies umfasst z.B. auch Branch/Merge-Funktionalitäten analog zum Commit im Kontext von CVS.

[115] „[...] it will be suitable for adoption by projects which already have artefact repositories but no process support." [BoLaNuRa03]

[116] „Each site is free to choose whatever tools are appropriate to their processes, varying from generic tools such as word processors, to more specific software engineering tools such as design tools or compilers." [BoSmWe03]

[117] „[...] it is in charge of creating new activity instances and assigning them to the workers specified in the concrete process model." [AvCiLuStVi03]

[118] „Different process instances may be created and enacted according to the same concrete process model." [AvCiLuStVi03]

[119] „[...] the event engine supports sending notifications of particular events to users according to a subscription policy." [AvCiLuStVi03]

[120] „Conditions are boolean expressions on the workflow control data to permit or inhibit the transition." [AvCiLuStVi03]

[121] „Also synchronization facilities to collaboratively work on the same artefact are provided by the artefact management system. Examples [...] are locking/unlocking or branching/merging of artefacts." [AvCiLuStVi03]

7.3.2.6 Etablierung von Kommunikationskanälen

Für die Etablierung geeigneter, sowohl synchroner als auch asynchroner Kommunikationskanäle, deren Maintenance und Verwaltung/Archivierung steht nach [BoSmWe03] und [AvCiLuStVi03][122] die sogenannte *Communication Engine* zur Verfügung, wobei aber noch keine detaillierten Informationen über deren Funktionsumfang vorliegen und daher auch keine konkreteren Aussagen über deren Einsatzmöglichkeiten zur Bereitstellung geeigneter Kommunikationsmöglichkeiten getroffen werden können.

7.3.2.7 Implementierter Beispielprozess des OSSD-Modells

Die folgende Grafik (*Abb. 7.12*) enthält den, mit der Prozessmodellierungskomponente von GENESIS modellierten, Prozess *Release Software* aus dem deskriptiven OSSD-Modell (vgl. *5.2.1 Software-Release*) in Form eines sogenannten abstrakten Modells:

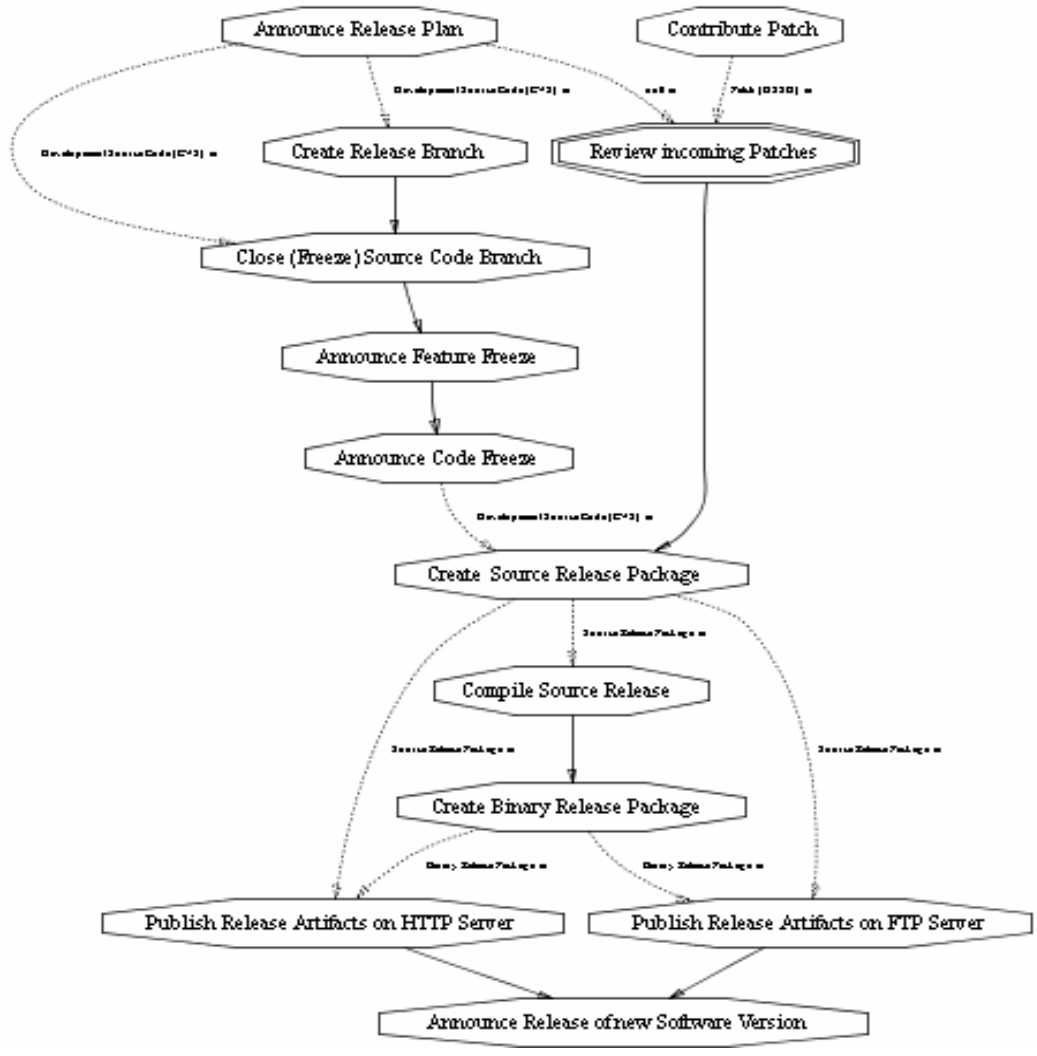

Abb. 7.12: In GENESIS implementierter OSSD-Teilprozess *Release Software*

[122] „[...] synchronous and asynchronous communication facilities are provided by the platform to help during collaborative work." [AvCiLuStVi03]

Im Rahmen der Prozessimplementierung wurden die verschiedenen Aktivitäten des Release-Prozesses in GENESIS mit der Prozessmodellierungskomponente abgebildet, indem für jede Aktivität eine Beschreibung, die ausführende Rolle und die In- und Outputartefakte definiert wurden. Die involvierten Entitäten (Artefakte, Rollen) wurden zuvor ebenfalls in GENESIS abgebildet. Anschließend wurden die verschiedenen Aktivitätsübergänge (*transitions*) definiert und die dabei zu übergebenden Artefakte zugewiesen. Basierend auf diesen Definitionen wurde anschließend von GENESIS automatisch die unten abgebildete Prozessdarstellung generiert.
Die Aktivität *Review incoming Patches* wurde dabei in einer von den restlichen Aktivitäten verschiedenen Darstellungsweise abgebildet, da es sich dabei um eine sogenannten *Super-Activity* handelt, welche zur Strukturierung von Prozessen (Prozessdekomposition) verwendet wird. Im Rahmen der Modellierung der OSSD-Prozesse des deskriptiven Modells dieser Arbeit wurden diese *Super-Activities* zur Implementierung von Teilprozessen verwendet, die in Form eines eigenen, abstrakten Prozessmodells weiter konkretisiert werden.

7.3.2.8 Status und Fazit

Die im Rahmen des GENESIS-Projekts entwickelte Software bietet Potentiale zur Implementierung und Unterstützung der OSSD-Prozesse und Prozessentitäten und zur Bereitstellung der identifizierten Softwarefunktionalitäten (*7.3.1 Überblick: Softwarefunktionalitäten und unterstützte Prozesse bzw. Anforderungen*). Der existierende Software-Prototyp realisiert momentan die geplante (konzeptuelle) Architektur und den geplanten Funktionsumfang der Software noch nicht vollständig und offenbart noch verschiedene Mängel (z.B. Usability, unvollständiges Artefaktmanagement). Zum aktuellen Zeitpunkt werden durch den Autor mithilfe der Prozessmodellierungskomponente von GENESIS im Rahmen einer Testinstallation die zentralen Prozesse dieser Arbeit als *abstract models* implementiert, um die Möglichkeiten zur Instanziierung und zur softwaretechnischen Unterstützung der instanziierten Prozesse durch GENESIS untersuchen zu können.

Momentan sind noch erheblich Mängel bezüglich der Usability und des implementierten Funktionsumfangs identifizierbar, wobei aber eine sukzessive Verbesserung dieser Aspekte von seitens der Core Developer des Projekts geplant ist [AvCiLuStVi03][123]. Der Zeitpunkt des Release eines Prototyps der Software, der alle dargestellten Funktionalitäten implementiert, ist momentan aber noch offen [BoLaNuRa03][124]. Eine erste fallstudienbasierte Evaluierung der Software in einem realen Entwicklungsprojekt wird momentan direkt im Kontext der GENESIS-Entwicklung durchgeführt, da die Prozesse der internen Weiterentwicklung der GENESIS-Software (Core Developer) durch den existierenden GENESIS-Prototyp unterstützt werden [BoLaNuRa03][125]. Zudem sind bereits interessierte Partner aus der Softwareentwicklungsindustrie an der Entwicklung der Software beteiligt, welche die Pilotversion der Software in ersten Fallstudien einsetzen und testen werden, wobei eine Verwendung durch die OSS-Community auch avisiert ist [BoSmWe03][126].

[123] „Besides the distributed process management, in the future we plan to add a number of new features to enhance the system both in ist functionalities and in ist usability.“ [AvCiLuStVi03]

[124] „Currently, the GENESIS platform is under development, nearing the release of the first stable version.” [BoLaNuRa03]

[125] „An initial case-study will be provided by hosting the development of the GENESIS project using the GENESIS platform.” [BoLaNuRa03]

[126] „[...] they will also be trialled by the repective projects’ industrial partners and possibly within the open source development community.” [BoSmWe03]

8 Potentiale und Anwendungsszenarien des Open Source Ansatzes

In diesem Kapitel werden Potentiale und mögliche Anwendungsszenarien des OSSD-Ansatzes identifiziert, um mögliche Interessenten an einem (verbesserten) OSSD-Modell aufzuzeigen. Dabei wird sowohl auf die Übertragbarkeit auf klassische Softwareentwicklungsprozesse eingegangen, als auch die Adaptierkeit typischer OSSD-Methoden als allgemeines Modell für verteilte Kooperation thematisiert. Da eine erschöpfende Auseinandersetzung mit dieser komplexen Thematik nicht direkt im Fokus dieser Arbeit liegt und rechtliche, soziologische und ökonomische Aspekte umfasst, dient dieses Kapitel primär dazu, die Potentiale und den bereits identifizier- oder prognostizierbaren Einfluss des OSSD-Ansatzes zu verdeutlichen. Die Grenzen der Realisierbarkeit der im folgenden skizzierten Szenarien wurden aber noch nicht umfassend untersucht und bedürfen daher einer eigenständigen wissenschaftlichen Untersuchung.

Anwendungsdomänen von OSS
OSS ist mittlerweile in vielen verschiedenen Anwendungsdomänen vertreten, wobei die kontinuierliche Etablierung von OSS-Projekten in neuen Domänen beobachtet werden kann (vgl. [Behlendorf99][127]). Die Bandbreite etablierter OSS-Produkte umfasst neben dem bekannten Betriebssystem GNU/Linux und dem Apache Webserver auch Software-Domänen wie Datenbanken, Content Management Systeme, Office Suites oder Application Server (vgl. [VoCo03][128]). Auch im professionellen Anwendungsbereich konnten sich verschiedene OSS-Produkte gegenüber kommerziellen Softwareprodukten etablieren [RuZuSu03][129]. Dies kann auch als Bestätigung der Relevanz und Praktikabilität des OSS-Entwicklungsmodells betrachtet werden. Auch im öffentlichen Sektor und in sicherheitskritischen Anwendungsdomänen spielt OSS inzwischen eine bedeutende Rolle [Grassmuck02][130], was durch die große Transparenz quelloffener Software begründet werden kann. Auch im Rahmen der EU existieren Förderprogramme zur Unterstützung quelloffener Software, wie zum Beispiel das Information Society Technology (IST) Programm[131] (vgl. [Grassmuck02]), in dessen Rahmen auch die in Anhang *E.2 GENESIS* dargestellte Software GENESIS [GENESIS03] unterstützt wird.

Trotzdem haben überdurchschnittlich viele OSS-Produkte bisher serverseitigen oder infrastrukturellen Charakter (Servertechnologien, Software-Entwicklungswerkzeug, Betriebssystemnahe Anwendungen) während der Bereich der Endanwendersoftware noch vergleichsweise unterrepräsentiert ist [Behlendorf99][132]. Dies resultiert aus der geschichtlichen Entwicklung des OSSD-Ansatzes, in dessen Rahmen frühe OSSD-Projekte typischerweise durch IT-Spezialisten aus dem Softwareentwicklungskontext initiiert wurden und somit in der Frühphase der Evolution des OSSD-Modells i.d.R. nur entsprechende Anwendungsdomänen und Zielgruppen adressiert wurden.

[127] „[…] there are probably very few commercial niches, that don't have at least beginnings of a decent open source alternativ available." [Behlendorf99]

[128] „The open source movement has produced robust office suites, operating systems, commercial grade database systems, not too mention the most popular web server, Apache. " [VoCo03]

[129] „Linux and the Apache web server are found in respectively 30% and 66% of the Internet's public servers, according to Netcraft's survey […]." [RuZuSu03]

[130] „In Deutschland setzen sich u.a. das Bundeswirtschaftsministerium, das Bundesamt für Sicherheit in der Informationstechnologie (BSI) sowie die Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik (KBSt) für den Einsatz freier Software ein." [Grassmuck02]

[131] IST: URL: http://www.cordis.lu/ist/home.html

[132] „Open-source software has tended to be slanted towards the infrastructural/back-end side of the software spectrum […]." [Behlendorf99]

8.1 Anwendungsszenarien des OSSD-Ansatzes in der Software-Entwicklung

„[…] open-source development model might not be the right choice for every software project, but it is still suitable for a wide variety of systems." [Cubranic01]

Der Einfluss, den die Erforschung des OSS-Entwicklungsmodells auf die klassische Wissenschaftsdisziplin des Software Engineering und die in diesem Kontext vertretenen Vorgehensmodelle ausüben wird, ist momentan noch nicht absehbar und auch mögliche Szenarien der Integration in die real praktizierten Softwareentwicklungsprozesse im proprietären Bereich befinden sich gerade erst in einem Frühstadium ihrer Entwicklung [Jordan01][133].

Im Rahmen dieser Arbeit wurden aber bereits verschiedene Parallelen des OSS-Entwicklungsmodells zu real praktizierten Softwareentwicklungsprozessen identifiziert, die vermuten lassen, dass verschiedene Elemente des OSSD-Ansatzes auch in kommerzielle, proprietäre (verteilte) Softwareentwicklungsprojekte integriert werden können. Die folgenden Einsatzszenarien des (erweiterten) OSSD-Modells und einer entsprechenden Entwicklungsinfrastruktur können bereits auch in der realen SE-Praxis identifiziert werden:

1. Optimierte Entwicklungsprozesse in klassischen OSSD-Projekten
2. Kommerzielle Software-Entwicklung basierend auf OSSD-Methoden
    a. Nutzung von OSSD-Methoden in verteilten Software-Entwicklungsprojekten (Closed Source)
    b. Offenlegung des Quellcodes von (Elementen) proprietärer Software
    c. Unterstützung bestehender OSSD-Projekte

8.1.1 Optimierte Entwicklungsprozesse in klassischen OSSD-Projekten

Die Realisierung von optimierten Entwicklungsprozessen in klassischen OSSD-Projekten, in denen Software unter einer OSD-konformen Lizenz entwickelt wird, erscheint als das naheliegendste Szenario für das erweiterte OSSD-Modell. Durch die frühzeitige Etablierung der Prozesse des (deskriptiven) OSSD-Modells und der notwendigen Infrastrukturen können der typische, langwierige Prozess der Evolution der Entwicklungsprozesse wesentlich verkürzt und typische Prozesse, Rollen, Artefakte und Infrastrukturen bereits während der Initialisierung eines OSSD-Projekts etabliert und unterstützt werden. Dadurch könnte die Produktivität eines OSSD-Projekts bereits in einer frühen Phase der Projektentwicklung gesteigert und der typische Evolutionsprozess in OSSD-Projekten verkürzt werden. Durch die zusätzliche Realisierung der in dieser Arbeit definierten, präskriptiven Prozesserweiterungen und einer verbesserten Software-Infrastruktur könnte zudem sowohl die Prozess- als auch die Produktqualität verbessert werden.

8.1.2 Kommerzielle Software-Entwicklung basierend auf OSSD-Methoden

„The recent explosion in popularity of open-source software development has attracted increasing attention in the developer's community. What was previously dismissed as too ad-hoc and chaotic for serious projects is now being reconsidered as a viable development model even for commercial software." [Cubranic01]

Die kommerzielle Verwendung von OSS in professionellen Anwendungsdomänen und die damit verbundene Verwertung eröffnet auch kommerziellen Software-Unternehmen große Potentiale (vgl. [DeSeVo03][134]). Neben der reinen Verwendung von OSS in kommerziell orientierten Softwareentwicklungsunternehmen eröffnet auch die

[133] „What is not clear is the impact open source software development and distribution methods will have on the software industry as a whole." [Jordan01]

[134] „There are many reasons for commercial organizations to be interested in using OS software (OSS), e. g. usually the quality or the transparency of OSS." [DeSeVo03]

Etablierung des dargestellten OSS-Entwicklungsmodells, einzelner Bestandteile dessen oder der beschriebenen Software-Infrastruktur Möglichkeiten für kommerzielle Unternehmen.

Nutzung von OSSD-Methoden in verteilten SW-Entwicklungsprojekten

Aktuelle kommerzielle Softwareentwicklungsprojekte unterliegen immer mehr einer starken, z.T. auch unternehmensübergreifenden Dezentralisierung und Verteilung der involvierten Akteure (vgl. [AvCiLuStVi03][135], [BoSmWe03][136]). Auch erscheinen die Planungs- und Entwicklungsprozesse weit weniger hierarchisch, formalisiert und phasenorientiert, als dies durch verschiedene propagierte Vorgehensmodelle suggeriert wird. Da sich somit diverse Merkmale des OSSD-Modells mit den Rahmenbedingungen real praktizierter SE-Prozesse überschneiden, stellen sich an das jeweilige Vorgehensmodell und die verwendete Software-Infrastruktur zum Teil ähnliche Anforderungen. Daher erfahren die Methoden des OSSD eine stetig wachsende Praxisrelevanz auch für kommerzielle Softwareentwicklung [WaAb03][137]. Dabei ist es vorstellbar, verschiedene OSSD-Methoden für eine limitierte Zahl von Entwicklern, also ein abgrenzbares Team, zu etablieren oder den Quellcode der gesamten Zahl der Mitarbeiter eines Unternehmens zur Verfügung zu stellen, was z.B. mit dem in [DiGa03] dargestellten Ansatz des sogenannten *Corporate Source* praktiziert wird. Indem der Quellcode somit lediglich für ein abgrenzbares Team von Akteuren offengelegt wird, kann die komplexe Lizenzproblematik umgangen werden, die mit einer vollständigen oder partiellen Offenlegung des Quellcodes für die gesamte Öffentlichkeit verbunden ist.

Offenlegung (von Elementen) des Quellcodes kommerzieller Software

Die Veröffentlichung von ehemals kommerziell verwertetem Quellcode bietet verschiedene Vorteile, da Softwareentwicklungsprojekte dadurch von einem theoretisch unbegrenzten Anwender- und Entwicklerpotential profitieren können:

- Unabhängiger Peer Review
- Erweiterung der Entwicklerressourcen
- Eigenwerbung
- Steigerung der Marktanteile
- Schnellere Implementation anwendungsorientierter Änderungen

Die komplexe Lizenzierungsproblematik in diesem Kontext kann durch die lediglich partielle Veröffentlichung des Quellcodes einer Software eingeschränkt werden. Eine derartige Strategie könnte z.B. die Offenlegung von peripheren Softwareapplikationen oder lediglich der API eines Softwareprodukts beinhalten, während der Quellcode des eigentlichen Produkts weiterhin geschlossen bleibt (vgl. [Pavlicek00][138]). Dadurch könnte die Entwicklung peripherer Anwendungen in einem OSSD-Prozess forciert werden, wobei durch die Zunahme des Funktionsumfangs supplementärer Applikationen einer Software auch der Wert des eigentlichen, kommerziellen Softwareprodukts gesteigert werden könnte. Das diametral entgegengesetzte Szenario ist ebenfalls in Betracht zu ziehen, bei dem z.B. die eigentliche Kernapplikation offengelegt und einem OSS-Entwicklungsmodell zugeführt wird, während der Verkauf und Vertrieb von proprietären, auf der veröffentlichten Software basierenden Applikationen (Peripherie) als Geschäftsmodell betrachtet wird (vgl. [Rosenberg00][139]).

---

[135] „[...] companies are moving from traditional software factory models towards virtual organization models, where people make substantially more use of network based communication than physical presence to interact and cooperate within a project.“ [AvCiLuStVi03]

[136] „With increasing globalisation of the software industry, cross organisational multi-company projects are becoming commonplace. Most large software projects are undertaken by teams of software staff working across a number of organisations.” [BoSmWe03]

[137] „Geographically and culturally dispersed organizations could benefit from analyzing the pros and cons of the different OSS paradigms, and adapt the most prominent solutions into use in their specific context of software development.” [WaAb03]

[138] „If you cannot open the main code, then publish all the Application Programming Interface (API) specifications to other programs can call your code.“ [Pavlicek00]

[139] „Ajuba’s approach is to continue Tcl/Tk as an Open Source product, while adding proprietary extensions to it [...] and selling those.” [Rosenberg00]

8.1.3 Unterstützung bestehender OSS-Entwicklungsprojekte

„Many companies not only use OSS, but also take advantage of the available source code and work on the code base to improve it, fix bugs, or add features or interfaces in order to make it even more useful for their specific purposes." [DeSeVo03]

Das aktive Mitwirken an der Weiterentwicklung bereits existierender OSS-Projekte kann für ein kommerziell ausgerichtetes Unternehmen ebenfalls von Vorteil sein [DeSeVo03][140]. Dies bedeutet, dass ein kommerzielles Unternehmen als Akteur im OSS-Prozess auftritt, wobei es verschiedene Rollen, typischerweise die eines Entwicklers, einnehmen kann. Dies erscheint z.B. sinnvoll, wenn eine proprietäre Software direkt auf einer OSS basiert, was z.B. im Falle von MySQL, Apache HTTPD oder dem weit verbreiteten Linux Kernel sehr häufig beobachtet werden kann [GaLaAr00][141]. Dies kann z.B. auch die Bereitstellung von bezahlten Entwicklern zur ausschließlichen Mitwirkung an einem OSS-Projekt beinhalten, wodurch der Prozess der OSS-Entwicklung planbarer und zuverlässiger gesteuert werden könnte und beispielsweise die ergänzenden Prozesse der Core Developer durch diese Akteure ausgeübt werden können [DeSeVo03][142].

Dies kann u.a. die folgenden Vorteile für ein Softwareunternehmen bieten:

- Wertsteigerung der eigenen (auf der OSS basierenden) Software
- Einflussnahme auf den Leistungsumfang der OSS
- Herstellerunabhängigkeit
- Eigenwerbung und Imageverbesserung

8.1.4 OSSD als übertragbares Softwareentwicklungsmodell

Wie bereits in den vorangegangenen Abschnitten dargestellt wurde, repräsentiert das Modell der OSS-Entwicklung eine alternative Vorgehensweise der Software-Entwicklung mit ambivalenten Einsatzmöglichkeiten. Dies wird inzwischen auch von der proprietären Softwareentwicklungsindustrie wahrgenommen und zunehmend realisiert [WaAb03][143].
[Massey03] dokumentiert zudem auch eine bestehende Diskrepanz zwischen den durch die Wissenschaftsdisziplin des SE postulierten Modellen und den real existierenden Vorgehensweisen der Softwareentwicklung. Das dargestellte OSSD-Modell repräsentiert in diesem Zusammenhang eine evolutionär entwickelte und daher auch praxisrelevante Alternative zu anderen präskriptiven Vorgehensmodellen und könnte dadurch bei der Annäherung des wissenschaftlichen Verständnisses der Softwareentwicklung an real praktizierte Softwareentwicklungsmethoden behilflich sein [Massey03][144].

8.2 OSSD als verallgemeinerbares Modell für verteilte Kooperation

„The open source model may be replicated in many fields, as it's a natural consequence of the distributed nature of the web." [Sandred00]

---

[140] „The participation of companies in OS communities offers chances for both sides." [DeSeVo03]

[141] „Another way of making money out of open source is by using the relevant open source as a platform, upon which commercial (often proprietary) application software can be built." [GaLaAr00]

[142] „[...] commercial organizations can offer a lot for OS projects and contribute to the OS community, such as: developers with defined budget are committed to their assigned tasks and make planning more reliable so that projects can evolve faster, reliable service, thorough documentation." [DeSeVo03]

[143] „Software companies are showing more and more interest in finding out the possibilities of harnessing the OSS method to support their daily work as the method itself embodies several interesting aspects that could benefit the software industry." [WaAb03]

[144] „Reactions in response to the perceived mismatch between 'real world' commercial and OSS development and SE 'best practice' have run a gamut, from the development of customized and lightweight software development methodologies such as Extreme Programming … to the simple business-as-usual plan of ignoring the SE community altogether." [Massey03]

Mit der Evolution der industriell geprägten zu einer Informationsgesellschaft geht eine Unabhängigkeit von materiellen Ressourcen einher, die vor allem durch die technologischen Möglichkeiten zur digitalen Speicherung, Reproduktion und Distribution von Wissen ermöglicht wird [Grassmuck02][145]. Dies hat auch eine Dezentralisierung der Arbeitsmethoden und die Etablierung neuer Organisationsformen hervorgebracht, die ebenfalls von Methoden und Werkzeugen des OSSD-Ansatzes profitieren könnten.

Mit zunehmenden Möglichkeiten zur Etablierung technologischer Infrastrukturen zur vernetzten Kollaboration erscheint es möglich, verschiedene Ansätze des OSSD-Modells als allgemeines soziotechnologisches Kooperationsmodell zu verstehen [FeFi02][146]. Das OSSD-Modell wird im wissenschaftlichen Kontext z.B. auch häufig mit der *Scientific Method*, also der konsequenten Veröffentlichung und kollaborativen Verbesserung wissenschaftlicher Informationen verglichen. Parallelen existieren dabei z.B. in ähnlichen Motivationsfaktoren und dem gemeinsamen Ansatz der Publikation von Wissen. Es erscheint prinzipiell möglich, den Ansatz der Offenlegung von Informationen und ein darauf basierendes Modell der kollaborativen Informationsentwicklung auf verschiedene Lebensbereiche anzuwenden [Sandred00][147]. Primär scheinen in diesem Zusammenhang die Bereiche der kollaborativen Wissensproduktion und das Verständnis des OSSD-Modells als allgemeines Kooperations- und Organisationsmodell von Bedeutung.

8.2.1 Kollaborative Wissensproduktion

Es existieren bereits verschiedene Ansätze, die sich mit der Veröffentlichung und kollaborativen Weiterentwicklung von Inhalten und Informationen befassen. Beispielsweise dient das Projekt Open Theory[148] als Projekt-Host für die kooperative Entwicklung von Theorien und Texten und ist in dieser Funktion mit dem Projekt Sourceforge vergleichbar, welches Infrastrukturen für OSS-Projekte bereitstellt [OT03][149]. Als weiterer Ansatz können kollaborativ erstellte Webkataloge betrachtet werden. Beispielsweise wird im Open Directory Project[150] (ODP) durch ein global verteiltes, offenes Projektteam ein Web-Verzeichnis entwickelt, welches die Inhalte des WWW nach Themenbereichen kategorisiert und strukturiert. Zur Informationsverwertung wurde auch in diesem Kontext ein spezielles Lizenzmodell, die Open Directory License (ODL) [ODP03][151] entwickelt. Auch die sogenannte WWW Virtual Library[152] stellt eine frühe Form eines kollaborativ erstellten Webverzeichnisses dar.

Als weitere bedeutende Ausprägungen der kollaborativen Informationsproduktion können die internetbasierenden, unabhängigen Medien betrachtet werden, die in [Grassmuck02] auch unter der Bezeichnung Peer-to-Peer-Journalismus (P2P) zusammengefasst werden. Plattformen wie Telepolis[153], Slashdot[154] oder Indymedia[155] stellen lediglich ein Forum bereit, welches durch Interessierte zur kollaborativen Entwicklung und Veröffentlichung von aktuellen Beiträgen genutzt wird und beseitigen bzw. reduzieren die Autorität einer zentralen Redaktion. Derartige Weblogs stoßen vor allem auch in der OSS-Community auf weitreichende Akzeptanz, was vor allem auch aus

[145] „Information wird anstelle von Materie und Energie zur zentralen industriellen Ressource und Ware.“ [Grassmuck02]

[146] „OSS may be regarded as a software engineering paradigm shift, or a software business strategy revolution, but we believe that at its heart, it is a sociological phenomenon, which has potentially massive implications for the future of work and society.” [FeFi02]

[147] „The idea of open source can be applied to any kind of digital information, not just computer software.” [Sandred00]

[148] URL: http://www.opentheory.org

[149] „Offene Theorie - open theory, kurz ot, ist der Versuch, das Modell freier Softwareentwicklung auf die Entwicklung von Theorie zu übertragen.“; URL: http://www.opentheory.org/open_theory/text.phtml; Abfrage: 23.05.2003

[150] URL: http:// www.dmoz.org

[151] URL: http://dmoz.org/license.html; Abfrage: 20.06.2003

[152] URL: http://vlib.org

[153] URL: http://www.telepolis.de

[154] URL: http://slashdot.org

[155] Indymedia: Independent Media Center; URL: http://www.indymedia.org bzw. http://de.indymedia.org

dem ähnlichen Ansatz resultiert, der auf der kollaborativen und weitgehend demokratischen Entwicklung im Rahmen eines sukzessiven Verbesserungsprozess analog zur OSS-Entwicklung beruht [CuHoYiMu03][156].
Als weitere Ausprägung kollaborativer Wissensentwicklung können Wiki-basierte Webseiten betrachtet werden. Mit dem Terminus *Wiki* werden zum einen Softwaretools, welche zur offenen, kollaborativen und weitgehend unreglementierten Content-Entwicklung verwendet werden und zum anderen auch die damit unterstützten Websites bezeichnet [CuHoYiMu03][157]. Als intranetweite Kommunikationsbasis werden Wiki-Systeme nach [Sixtus03] bereits bei verschiedenen Unternehmen eingesetzt. Im Rahmen des Wikipedia-Projekts wird der Wiki-Ansatz zur kollaborativen Wissensbildung und –archivierung verwendet und eine Wiki-basierte Enzyklopädie erstellt, die nach [Sixtus03] momentan über 120.000 Einträge und Übersetzungen in mehrere Sprachen verfügt und als *„größte, freie Wissensdatenbank im Netz“* [Jendro03] betrachtet werden kann.

Als einen weiteren Schritt in der Dezentralisierung und Demokratisierung der kollaborativen Wissens- bzw. Contentdistribution kann das sogenannte Freenet-Projekt[158] betrachtet werden. Im Rahmen von Freenet wurden Algorithmen und eine darauf basierende OSS entwickelt, welche ähnlich zu klassischen P2P-Projekten die dezentralisierte Generierung und Veröffentlichung von Informationen ermöglichen und speziell die zensurbedingte Einschränkung der Meinungsfreiheit durch Anonymisierung der Akteure verhindern soll (vgl. [Pawlo03][159]).

8.2.2 Virtuelle Organisationsformen

„Auch die unterschiedlichen Wissenspraktiken der Wissenschaften, des Journalismus, der Künste, der Politik usw. müssten jeweils auf ihre Eignung für offene Kooperationen hin untersucht werden.“ [Grassmuck02]

Allgemein entspricht das OSS-Entwicklungsmodell den veränderten Anforderungen moderner Organisationsformen und weist viele Parallelen zur modernen, projektbezogenen Unternehmensorganisation in dynamisch variierten, verteilten Projektteams auf (vgl. [MaMaAg03][160]). Dies umfasst vor allem die Aspekte Dezentralisierung, Hierarchieverflachung und vertikale Kooperation, die sowohl im OSSD als auch zunehmend in aktuellen Organisationsformen von Bedeutung sind. Nach [Grassmuck02][161] repräsentiert der OSSD-Ansatz bezüglich des Grades der Dezentralisierung und der Hierarchieverflachung eine extreme Ausprägung eines Organisations- und Prozessmodells.
Auch das Konzept der vertikalen Kooperation, also die aktive Einbeziehung der Anwender in Entwicklungsprozesse, stellt ein Fundament des OSSD-Modells dar und wird zunehmend auch in kommerziell orientierten Projekten praktiziert.

Auch der Ansatz der verschiedenen, weit verbreiteten Peer-To-Peer-Netzwerke (P2P), welche zum dezentralisierten Austausch von Dateien verwendet werden, weist Parallelen zum OSSD-Modell auf, da hier in einem kollaborativen Prozess durch die Nutzer ein gemeinsam erstelltes Produkt in Form der Gesamtheit der zur Verfügung gestellten Datenmenge erstellt wird. Desweiteren existieren nach [FeFi02][162] bereits weitere Ansätze, in

[156] „While weblogs are not commonly associated with opensource software engineering, they are certainly a significant and influential presence in the open-source community.” [CuHoYiMu03]

[157] „A wiki is a tool for collaborative development of documents and web pages, although the term is also used to describe such web sites themselves.“ [CuHoYiMu03]

[158] URL: http://www.freenetproject.org

[159] „Freenet wendet sich gegen jede Art von Zensur, indem es eine technische Infrastruktur zur Verfügung stellt, die unkontrollierbar ist, denn anders als das Internet distributiert Freenet dezentralsiert Information.“ [Pawlo03]

[160] „[…] there is a relatively high degree of correspondence between the open-source movement and popular depictions of the organization of the future and the virtual networked organization.” [MaMaAg03]

[161] „Die Projekte der freien Software übertreffen jede management- und organisationstheoretische Vision an Dezentralisation, lockerer Kooperation und zwangloser Koordination.“ [Grassmuck02]

[162] „A project at the Berkman Center at the Harvard Law School is investigating the transfer of the Open Source model to other areas, including governance, education, economics, and law.“ [FeFi02]

denen versucht wird, die Eignung von OSSD-Methoden zur Unterstützung von kollaborativen und verteilten Prozessen zu evaluieren.

8.3 OSS und Geistiges Eigentum – Gesetzliche Rahmenbedingungen und Lizenzmodelle

„Information and freedom are indivisible. The information revolution is unthinkable without democracy, and true democracy is unimaginable without freedom of speech." (UN Generalsekretär Kofi Annan auf der Global Knowledge Conference; 1999; aus [Sandred00])

Im allgemeinen Spannungsfeld freier Wissensentwicklung und im speziellen Kontext der OSS stellen die juristischen Rahmenbedingungen eine zentrale Einflußgröße dar, welche die Realisierbarkeit derartiger Modelle wesentlich determinieren und damit maßgeblich für den langfristigen Erfolg dieser Ansätze sind. Dies offenbart sich z.B. in den sehr komplexen Diskussionen um Lizenzmodelle und rechtliche Fragen des Softwarepatentierungs- und Urheberrechts. Diese Problematik liegt nicht im Fokus dieser Arbeit, soll aber an dieser Stelle einführend thematisiert werden, um ein Verständnis über die Komplexität und Bedeutung dieser Einflussgrößen zu ermöglichen.

Gesetzliche Rahmenbedingungen
Mit der zunehmenden Evolution technologischer Möglichkeiten zur Reproduktion und Distribution geistiger Werke und geistigen Eigentums wurden rechtliche Grundlagen zu deren Schutz entwickelt, welche die kommerziellen Interessen der Eigentümer wahren sollen, aber z.T. dem freien Informationsfluss und dem OSS-Gedanken widersprechen. Restriktiver Umgang mit Wissen, der sich z.B. in der Patentierung von Software oder restriktiven Lizensierungsmethoden manifestiert, widerspricht direkt dem OSS-Ansatz [Rosenberg00][163]. Gerade im Problemfeld der Softwarepatentierung bzw. der Patentierung von entsprechenden Algorithmen ist die Rechtslage noch weitgehend ungeklärt und differiert erheblich in den verschiedenen nationalen Gesetzgebungen (vgl. [Waldt03]). Allgemein ist ein möglichst freier Informationsfluss und die Möglichkeiten zur Veröffentlichung und Verbreitung von Informationen notwendig, um kollaborative Verbesserungs- und Entwicklungsprozesse zu motivieren und zu ermöglichen [Sandred00][164]. Daher finden sich verschiedene Parallelen zwischen der Argumentation der OSS-Community und der allgemeinen Debatte um geistiges Eigentum und Urheberrechte [Rosenberg00][165].

Offene Lizenzmodelle
Neben neueren Softwarelizenzmodellen, die sich im OSS-Kontext etablieren konnten und inzwischen z.T. auch von der Industrie adaptiert werden, haben sich inzwischen analog zum OSS-Ansatz Lizenzmodelle entwickelt, die eine möglichst offene und freie Verbreitung und Entwicklung von Inhalten im Allgemeinen ermöglichen. Beispielsweise wurde die Open Publication License (OPL)[166] entwickelt, um den Ansatz der GNU GPL in den Kontext der Produktion von Inhalten zu transportieren [Sandred00][167]. Eine ähnliche Zielsetzung wird mit der

[163] „The Open Source movement […] feels that the innovation and spread of knowledge that were originally supposed to be promoted by the limited monopolies of the patent and copyright systems are now becoming victims of broad extensions of these two IP protections." [Rosenberg00]

[164] „It is well known that a free flow of information promotes progress, growth, and production. Secrecy prevents others from helping each other to solve common problems." [Sandred00]

[165] „[…] the Open Source community finds itself trying to defend software users, readers, and music fans by providing arguments and technology to be used against IP conglomerates such as Walt Disney and Time Warner and their trade associations, such as the Motion Picture Association of America (MPAA) and the Recording Industry Association of America (RIAA)." [Rosenberg00]

[166] URL: http://www.opencontent.org

[167] „The OPL grants anybody permission to modify and redistribute the materials provided changes are marked and resulting work is also put under the license." [Sandred00]

GNU Free Documentation License (GNU FDL)[168] verfolgt, welche von der FSF als Lizensierungsmodell für Dokumentationen analog zur GNU GPL entwickelt wurde. Ein weiterer Ansatz wird mit dem an der Stanford University initiierten Projekt Creative Commons[169] verfolgt, welches den Urhebern geistiger Werke die möglichst freie Adaption eines eigenen Lizenzmodells basierend auf einem Basis-Lizenzmodell ermöglichen soll. Der Ausgangspunkt der individuell angepassten Lizenzmodelle sind allgemeine Regelungen, die sich an der GNU GPL orientieren [Malotki03][170]. Dies soll zwar eine Veröffentlichung geistiger Werke ermöglichen, dabei aber den Urhebern stets die Rechte für eine zumindest partielle, kommerzielle Verwertung einräumen, indem sie beispielsweise im adaptierten Lizenzmodell Tantiemen-Zahlungen für den kommerziellen Einsatz des geistigen Werks festlegen. Als erster bedeutender Anwender dieser Lizensierungsmethode kann der Buchverlag O'Reilly[171] betrachtet werden [Malotki03][172].

Für die weitreichende Etablierung des OSSD-Ansatzes ist es notwendig, dass die rechtlichen Rahmenbedingungen in Zukunft eine möglichst freie Entwicklung, Publikation und Zirkulation von Wissen und geistigen Werken ermöglichen und dies nicht durch restriktive Reglementierung einschränken.

[168] URL: http://www.fsf.org/copyleft/fdl.html

[169] URL: http://www.creativecommons.org

[170] „Texte, Musikstücke, Bilder und Filme, die unter dem Label der Creative Commons veröffentlicht werden, sind vornehmlich frei zitier-, kopier- oder veränderbar, dafür muss diese Regel aber auch für daraus hervorgehende Werke gelten." [Malotki03]

[171] URL: http://www.oreilly.com/openbook

[172] „Der Buchverlag und Projektpartner O'Reilly hat unter einer der CC Lizenzen kürzlich den Großteil seiner nicht mehr aufgelegten Titel zum freien Download und zur Vervielfältigung bereitgestellt." [Malotki03]

9 Fazit und Ausblick

Dieses Kapitel fasst die Ergebnisse der Arbeit zusammen und stellt weitere Forschungsaktivitäten vor, die für die weitere wissenschaftlichen Auseinandersetzung mit der OS-Thematik sinnvoll erscheinen.

9.1 Ergebnisse

Im Rahmen dieser Arbeit wurde, basierend auf empirischer Analyse [Dietze03], ein deskriptives Prozessmodell der Softwareentwicklung im Kontext von Open Source entwickelt. Durch die fallstudienbasierte Analyse möglichst repräsentativer OSS-Projekte, basierend auf einem eigens entwickelten, gemeinsamen Metamodell [Dietze03b], wurde es ermöglicht, die typischen und projektübergreifend identifizierbaren Prozesse und Prozessentitäten zu generalisieren und durch Modellelemente der UML (vgl. [Dietze03b]) zu beschreiben und zu formalisieren [Dietze03c]. Dies stellte die Basis dar, um Implikationen und Tendenzen zu identifizieren, die sich aus dem deskriptiven Prozessmodell der OSS-Entwicklung ergeben. Daraus konnten Ansätze für mögliche Erweiterungen und Modifikationen des Modells abgeleitet werden, um den beschriebenen Ansatz gezielt zu verbessern und dadurch eine präskriptive Verwendung des OSSD-Modells zu erlauben (vgl. *1.2.2.1 Präskriptive, deskriptive und transiente Modelle*). In einem weiteren Arbeitsschritt wurden mögliche Erweiterungen der bisher identifizierbaren Softwareunterstützung des OSSD-Modells entwickelt, die eine gesamtheitliche Softwareunterstützung des OSSD-Ansatzes ermöglichen. Zudem wurde ein möglicher Implementationsansatz dargestellt, der Potentiale zur Implementierung der Softwarestützung und zur softwaretechnischen Realisierung des gesamten Prozessmodells bietet. Aufbauend auf dieser Arbeit konnten zudem verschiedene Anwendungsszenarien für das gesamte OSSD-Modell oder einzelner Methoden dessen eruiert werden. Die Ergebnisse dieser Arbeit implizieren die folgenden Thesen und Beiträge.

Methode (Metamodell) zur Analyse und Modellierung von Software-Entwicklungsprozessen
Zur Durchführung von Fallstudien im Kontext des SE, respektive zur strukturierten Analyse von Prozessen im Rahmen der Software-Entwicklung ist ein allgemeines Metamodell notwendig, welches die zu analysierenden Prozesse und Prozessentitäten auf einer Metaebene beschreibt und strukturiert. Zudem erfordert auch die anschließende, formalisierte Modellierung der analysierten Prozesse und Prozessentitäten ein gemeinsames Metamodell, welches die Modellierungselemente und -sichten zur Darstellung der identifizierten Prozesse und Strukturen definiert. Im Rahmen dieser Arbeit wurde ein derartiges Modell entwickelt (vgl. Kapitel *2 Metamodell zur Prozessanalyse und –modellierung* bzw. [Dietze03b]), welches zum Teil bereits nach OSS-spezifischen Gesichtspunkten gebildet wurde, aber auch zur Analyse und Modellierung allgemeiner SE-Prozesse aus Nicht-OS-Softwaredomänen verwendet werden kann. Dieses Metamodell wurde mit UML-Modellelementen beschrieben und definiert sowohl die zu analysierenden Entitäten der Softwareentwicklungsprozesse als auch verschiedene Sichten und Modellierungselemente, welche zu deren formalisierter Beschreibung verwendet werden können. Die Wiederverwendbarkeit dieses Metamodells wird dabei durch die Orientierung an etablierten Gestaltungsmitteln der UML unterstützt.

Methoden und Prozessentitäten des OSSD z.T. zu einem deskriptiven OSSD-Prozessmodell generalisierbar
Basierend auf den durchgeführten Fallstudien wurde festgestellt, dass bestimmte Methoden und Prozesse projektübergreifend in verschiedenen OSSD-Projekten angewendet werden und diese daher generalisiert und formalisiert beschrieben werden können. Die durchgeführten Fallstudien beschränkten sich dabei auf OSSD-Projekte, deren Strukturen und Prozesse bereits einen umfangreichen evolutionären Prozess durchlaufen haben, um so die typischen Prozesse und Prozessentitäten analysieren zu können, die in OSS-Projekten im Laufe einer typischen Projektentwicklung evolutionär entstehen. Daher wird davon ausgegangen, dass das dargestellte, generalisierte und formalisierte Prozessmodell und die identifizierten Entwicklungsmethoden (vgl. Kapitel 3-5 und [Dietze03c]) einen Zustand beschreiben, den OSS-Projekte typischerweise im Verlauf eines Projekts einnehmen, sofern bestimmte Rahmenbedingungen erfüllt werden, wie z.B. die Verwendung eines OSD-konformen

Lizenzmodells oder die Überschreitung einer kritischen Anzahl von Communitymitgliedern. Diese These wurde auch im Rahmen der Validierung des deskriptiven OSSD-Modells (vgl. Abschnitt *5.5 Validierung des deskriptiven Prozessmodells* und [Dietze03c]) bestätigt.

OSSD - Prozess der sukzessiven Softwareverbesserung
Das identifizierte, deskriptive Modell der OSSD-Prozesse kennzeichnet den gesamten Entwicklungsprozess als einen Prozess der sukzessiven Softwareverbesserung, der immer auf der Veröffentlichung eines Software-Prototyps basiert, was somit als wichtige Voraussetzung für OSSD-Projekte betrachtet werden kann. Daher stellen die Methoden des OSSD keine probate Methode zur vollständigen Entwicklung von Softwaresystemen dar, sondern sind lediglich zur verteilten Pflege und Weiterentwicklung eines bereits existierenden Softwareprototyps geeignet.

Reduzierung der Projektevolutionsdauer
Ein wesentlicher Nutzen einer formalisierten Beschreibung des OSSD-Modells kann darin gesehen werden, dass durch die Analyse und allgemeinverständliche Beschreibung dieser Prozesse die Evolution von OSS-Projekten erheblich beschleunigt werden kann, indem ein Inititiator eines Projekts das deskriptive Modell dieser Arbeit bereits bei der Initialisierung eines OSS-Projekts berücksichtigt und entsprechende Prozesse und Prozessentitäten bereits zu Beginn der Projektevolution etabliert. Dies könnte z.B. durch die Bereitstellung der entsprechenden Softwarestützung und des dargestellten Artefaktkonzepts oder durch die frühzeitige Etablierung oder Unterstützung der dargestellten Rollen, insbesondere von Rollen der Core Developer (z.B. Release Manager) erfolgen.

Generalisierbare Implikationen und Tendenzen
Basierend auf der umfassenden Analyse und formalisierten Beschreibung des OSSD-Modells konnten verschiedene Tendenzen, Implikationen und Schwachstellen dieses Ansatzes identifiziert werden (vgl. Kapitel *6.1 Implikationen und Erkenntnisse aus der empirischen Analyse*), die - eine Generalisierbarkeit des identifizierten Modells vorausgesetzt - auf typische OSSD-Prozesse zutreffen. Wichtige Aspekte in diesem Zusammenhang sind die Tendenz zur Vernachlässigung verschiedener Prozesse, die nicht formalisierte und dadurch nicht garantierte Qualitätssicherung und die dediziert zu unterstützende, hohe Frequenz der Veröffentlichung neuer Software-Releases.
Desweiteren sind einige der bereits in *3 Charakteristika und gesamtheitliches Modell der OSS-Entwicklung* dargestellten Erkenntnisse von großer Bedeutung für die weitere wissenschaftliche oder pragmatische Auseinandersetzung mit dem OSSD-Ansatz. Dies betrifft vor allem die Tendenz zur Herausbildung eines Entwicklerkerns (Core Developer) und den hohen Grad der Prozessparallelisierung und Prozessautonomie.

Verbesserungspotentiale
Das evolutionär entwickelte Prozessmodell des OSSD birgt verschiedene Vorteile, die vor allem aus dem hohen Grad der Prozessparallelisierung und dem *„darwinistischen Selektionsprozess“* im Rahmen der Softwareevolution resultieren. Diese Potentiale umfassen z.B. unabhängige Peer Review-Prozesse, eine anwendergerechte Anforderungsdefinition und weitgehend selbstregulierende Prozesse (vgl. *6.2 Potentiale und Risiken des OSSD-Modells*). Trotzdem konnten im Rahmen dieser Arbeit verschiedene Risiken und Schwachstellen identifiziert werden (vgl. *6.2 Potentiale und Risiken des OSSD-Modells*), die im Rahmen klassischer Softwareentwicklungsmodelle oder im Kontext proprietärer Softwareentwicklung nicht oder in weitaus geringerem Maße existieren. Diese basieren z.T. auf einer mangelhaften Softwarestützung, den nur sehr rudimentär formalisierten Qualitätssicherungsprozessen oder dem Ansatz, die Anwender als Tester in den Entwicklungsprozess zu integrieren. Daher konnten verschiedene Zielstellungen (*6.5 Zielstellungen für die Erweiterung und Unterstützung des OSSD-Modells*) definiert werden, welche die Vorgaben für die strukturierte Verbesserung des OSSD-Ansatzes darstellen. Die Realisierung dieser Zielstellung dient der Etablierung von effizienteren Prozessen in OSSD-Projekten und soll eine Nutzung von OSSD-Methoden auch in proprietären Softwareentwicklungsprozessen ermöglichen.

Prozessverbesserung durch Erweiterung der Prozesse und Rollen
Um den OSSD-Ansatz zu verbessern und somit ein präskriptiv einsetzbares, erweitertes Prozessmodell zu erstellen, konnten verschiedene Ansätze identifiziert werden, die vor allem auf der Etablierung zusätzlicher Prozesse und Rollen basieren. Diese Erweiterungen des OSSD-Modells (*7.1 Erweiterung existierender Prozesse und Rollen*) integrieren primär zusätzliche und formalisiertere Qualitätssicherungsprozesse bzw. Prozesse zur Kompensation von tendenziell in ungenügendem Mass ausgeführten Prozessen (z.B. Support, Dokumentationsentwicklung). Derartige Funktionen können primär durch Akteure der Core Developers bereitgestellt werden, da die enge Kooperation dieses Entwicklerkerns mit dem Maintainer eines Projekts eine konkrete Rollen- und Aufgabenzuweisung und die Planbarkeit der zugeordneten Aktivitäten ermöglicht. Daher sind derartige Erweiterungen des OSSD-Ansatzes vor allem im Rahmen von OSS-Projekten denkbar, die bereits über eine umfangreiche Community und ein Team von Core Developers verfügen oder die durch Entwicklerteams eines kommerziellen Softwareentwicklungsunternehmens ergänzt werden.

Prozessverbesserung durch Erweiterung der Artefakte und Softwarestützung
Neben einer Erweiterung der Prozesse und Rollen scheint auch eine Modifikation existierender bzw. die Einführung neuer Artefakte in das OSSD-Modell sinnvoll, da die Prozesse bisher nur rudimentär durch geeignete Artefakte unterstützt werden. Gerade in der verteilten Softwareentwicklung sind möglichst transparente und konsistente Ressourcen und Artefakte von zentraler Bedeutung, um deren kollaborative Entwicklung zu ermöglichen. Daher werden in *7.2 Erweiterung und Modifikation des Artefaktmodells* verschiedene neue bzw. erweiterte Artefakttypen eingeführt, z.B. die über Metadaten beschriebene Kategorie der Request-Artefakte, welche zur umfassenden Dokumentation und softwaretechnischen Unterstützung der OSSD-Prozesse dienen oder die ebenfalls metadatenbasierte Kategorie der Content-Artefakte, die zur strukturierten Verwaltung aller begleitender Informationsobjekte verwendet werden. Desweiteren konnten im Rahmen einer geeigneten Softwarestützung der identifizierten Prozesse verschiedene zusätzliche Softwarefunktionalitäten identifiziert werden (*7.3 Unterstützende Software-Infrastruktur*), welche die im deskriptiven Modell dargestellten Software-Infrastrukturen sinnvoll ergänzen und die beschriebenen Prozesse und Artefakte gesamtheitlich unterstützen. Da Kommunikation und Transparenz über alle Ressourcen und Prozesse in verteilten Projekten von großer Bedeutung ist, wurde neben der Notwendigkeit für konsistente Kommunikationskanäle die Notwendigkeit eines gesamtheitlichen, metadatenbasierten Artefaktmanagements identifiziert, welches die Verknüpfung semantisch verwandter Artefakte ermöglicht. Weitere sinnvolle Erweiterungen der Software-Infrastruktur beinhalten z.B. ein versioniertes, metadatenbasiertes Content-Management aller Informationsobjekte (Content-Artefakte) und ein gesamtheitliches Management aller Request-Artefakte.

Implementationsansatz: GENESIS
Als möglicher Implementationsansatz einer gesamtheitlicheren Software-Infrastruktur zur Unterstützung des OSSD-Modells wurde das in *7.3.2 Implementationsansatz - Implementierung des OSSD-Modells mit GENESIS* dargestellte GENESIS-Projekt identifiziert. Die Prozessmanagementsoftware GENESIS wird momentan als OSSD-Projekt[173] entwickelt und unter der Mozilla Public License (MPL) veröffentlicht, wobei bereits ein javabasierter Prototyp zur Verfügung steht. Das öffentlich geförderte Projekt, in welches auch europäische Hochschulinstitutionen involviert sind, verfolgt das Ziel, die Möglichkeiten der Softwareunterstützung verteilter Software-Entwicklungsprozesse zu erforschen und eine geeignete Softwareinfrastruktur zu entwickeln, wobei der Fokus auf die Unterstützung von OSSD-Prozessen gelegt wurde. Aufgrund des untersuchten Datenmodells der Software, welches eine softwaretechnische Implementierung der im Rahmen der Arbeit beschriebenen Prozessentitäten (Rollen, Artefakte, Prozesse) ermöglicht und den unterstützten Funktionalitäten der Software, scheint eine umfassende Softwareunterstützung des OSSD-Modells und auch der hergeleiteten Prozesserweiterungen durch GENESIS möglich. Die Modelle dieser Arbeit können auf der Typenebene mit der Prozessmodellierungskomponente von GENESIS direkt abgebildet werden (*abstract models*) und anschließend in verschiedenen Projekten zu sogenannten *concrete models* instanziiert werden. Derartig instanziierte Prozesse

[173] URL: http://sourceforge.net/projects/genesis-ist

werden anschließend durch die Workflowmanagementkomponente der Software unterstützt und automatisiert. Da bisher nur eine begrenzte Teilmenge des geplanten Funktionsumfangs durch die Software implementiert wurde, konnte eine erschöpfende Evaluierung der Potentiale von GENESIS zur Unterstützung des OSSD-Modells noch nicht im Rahmen der Dissertation erfolgen.

Anwendungsszenarien des OSSD Modells in der Softwareentwicklung
Es erscheinen verschiedene Anwendungsszenarien denkbar, die sowohl für das gesamte, verbesserte, softwaregestützte Modell des OSSD als auch für einzelne Elemente dessen in Betracht gezogen werden können (vgl. *8 Potentiale und Anwendungsszenarien des Open Source Ansatzes*). Diese umfassen neben einer Etablierung verbesserter, softwaregestützter Prozesse in typischen OSS-Entwicklungsprojekten auch verschiedene Szenarien in der Domäne der proprietären, kommerziellen Softwareentwicklung. Beispielsweise können Methoden des OSSD ergänzend in verteilten Softwareentwicklungsprojekten genutzt werden oder es kann eine unternehmensinterne Veröffentlichung von Quellcode erfolgen, um die interne Weiterentwicklung nach OSSD-Methoden zu ermöglichen. Desweiteren kann die Offenlegung von proprietärem Quellcode von Vorteil für ein kommerzielles Softwareentwicklungsunternehmen sein, da dadurch von den Beiträgen der Projektcommunity profitiert werden kann. In diesem Zusammenhang ist die Lizenproblematik für eine weitere kommerzielle Verwertung der Software von großer Bedeutung, wobei dieses Problemfeld aber durch eine lediglich partielle Quellcodeveröffentlichung eingegrenzt werden kann. Dies könnte z.B. bedeuten, dass lediglich periphere Applikationen oder Softwaremodule einer Software nach dem OSSD-Prinzip entwickelt werden, während die Kernsoftware weiterhin proprietär und kommerziell vertrieben wird. Auch die Veröffentlichung der Kernsoftware stellt eine Option dar, wobei aber das Geschäftsmodell des Softwareentwicklungsunternehmens entsprechend angepasst werden muss. Außerdem erscheint in einigen Fällen auch das Unterstützen bestehender OSS-Produkte für kommerziele Softwareentwickler lukrativ, sofern die proprietäre Software auf der jeweiligen OSS basiert und somit durch die Verbesserung der OSS auch die Attraktivität der kommerziellen Software gesteigert werden kann.
Bei allen skizzierten Einsatzmöglichkeiten ist es für das jeweilige Unternehmen von Bedeutung, die OSSD-Prozesse adäquat zu unterstützen. Dazu können die in dieser Arbeit dargestellten Prozesserweiterungen als möglicher Ansatz dienen und beispielsweise die dargestellten ergänzenden Rollen durch Akteure eines kommerziellen Unternehmens ausgeübt werden. Zudem erscheint es denkbar, das Modell des OSSD als allgemeinen Ansatz der verteilten Kooperation zu verstehen, wobei im Rahmen der Arbeit bereits verschiedene Ansätze der kollaborativen Wissensproduktion identifiziert werden konnten, die sich auf das OSSD-Modell beziehen (vgl. *8.2 OSSD als verallgemeinerbares Modell für verteilte Kooperation*).

9.2 Ausblick

Diese Arbeit hat verschiedene Problemfelder und Fragestellungen identifiziert, die aufbauend auf den Ergebnissen dieser Arbeit im Rahmen weiterer wissenschaftlicher Auseinandersetzung mit der Thematik des OSSD bearbeitet werden sollten. So konnten einige Fragestellungen, die nicht direkt im Fokus der Zielstellungen dieser Arbeit lagen, nur rudimentär diskutiert werden:

- Validierung der eingeführten Prozesserweiterungen
- Entwicklung einer gesamtheitlichen Softwarestützung
- Anwendungsszenarien des OSSD-Modells im kommerziellen SE
- Domänenübergreifender Einfluss des OSSD auf verteilte Kollaborations- und Organisationsformen

Die verschiedenen, in *7 Erweiterung der Entitäten des OSSD-Modells* vorgestellten Erweiterungen des OSSD-Ansatzes konnten noch nicht im Praxiskontext validiert werden. Daher sollten die dargestellten Prozesserweiterungen im Rahmen realer OSSD-Projekte etabliert und evaluiert werden, um deren Praxisrelevanz nachzuweisen und evtl. notwendige Modifikationen zu ermöglichen.

Im Rahmen dieser Arbeit wurden lediglich einige mögliche Erweiterungen der Software-Infrastruktur identifiziert und implementationsunabhängig beschrieben. Bisher konnte ein vielversprechender, aber momentan nur

rudimentär implementierter Ansatz einer gesamtheitlichen Softwarstützung identifiziert werden. Daher ist eine Implementierung eines derartigen Softwareframeworks zur Unterstützung des OSSD-Ansatzes von hohem Wert, unabhängig davon, ob das klassische OSSD-Modell (vgl. [Dietze03c]) oder ein verbesserter Ansatz unterstützt werden sollen. Eine derartige Infrastruktur könnte ebenfalls in proprietären, verteilten Softwareentwicklungsprojekten zur Unterstützung der dezentralen kollaborativen Entwicklungsprozesse eingesetzt werden.

Desweiteren erscheint es sinnvoll, die Realisierbarkeit von OSSD-Methoden in verschiedenen Softwareentwicklungsdomänen, die z.T. in *8 Potentiale und Anwendungsszenarien des Open Source Ansatzes* diskutiert wurden, fundiert zu erforschen und anschließend im Praxiskontext zu validieren. Dies könnte zudem als Grundlage dienen, ein allgemeines, präskriptives Modell der verteilten Softwareentwicklung zu definieren, welches sich an Ansätzen der OSS-Entwicklung orientiert und dadurch zusätzliche Praxisrelevanz erfährt. Dies könnte die Wissenschaftsdisziplin des Software Engineering um ein Modell erweitern, welches konkret den realen Anforderungen und Praktiken verteilter Softwareentwicklung entspricht.

Zudem kann das soziotechnologische Modell des OSSD-Prozesses als Ausgangspunkt dienen, die enthaltenen Ansätze zur Kooperation verteilter Akteure auf andere Domänen als die der Softwareentwicklung zu übertragen und dessen Potential als allgemeingültiges Modell für verteilte Kooperations- und Organisationsformen zu untersuchen (vgl. *8.2 OSSD als verallgemeinerbares Modell für verteilte Kooperation*).

Anhang

A Implikationen und Tendenzen des OSSD-Ansatzes

A.1 Tendenz zu redundanten Aktivitäten

Die parallele Bearbeitung von Aufgaben führt zum Teil zu redundanten Ergebnissen bzw. zur redundanten Durchführung verschiedener Aktivitäten, wodurch es möglich ist, unter verschiedenen Resultaten das bestmögliche zu selektieren. Dies stellt zwar keine effiziente Methode zur Ausführung von Arbeitsschritten, stellt aber eine wichtige Voraussetzung für die evolutionäre Entwicklung unter dem Paradigma von Open Source dar und ist zudem ein sinnvoller Ansatz im Kontext derartig komplexer und heterogener sozialer Systeme (vgl. *6.2.1 Chancen und Vorteile*). Im Gegensatz dazu stellt ein stark zentralisiertes Management der Entwicklungsprozesse keine adädquate Alternative dar, da dieser Ansatz automatisch eine Steigerung des Koordinierungs- und Verwaltungsaufwands im Zuge der Integration neuer Entwickler nach sich zieht und somit nur ein endliches Wachstum entsprechend der vorhandenen Managementkapazitäten ermöglicht [Weber00][174]. Die Abwesenheit einer kontrollierenden Instanz im OSSD-Kontext führt aber zu sehr hohen Anforderungen an die Transparenz über alle Entwicklungsprozesse, um überflüssige und unbeabsichtigte Prozessredundanzen zu minimieren.

A.2 Projektdiversifikation - Segmentierung in Subprojekte

Sehr häufig konnte die Segmentierung eines OS-Projekts in verschiedene Subprojekte identifiziert werden, welche z.T. auch durch explizite Managementinstanzen verwaltet werden. In derartig komplexen sozio-technologischen Systemen wie dem eines OSS-Projekts steht die Evolution der Community in direktem Zusammenhang mit der Evolution der entwickelten technologischen Artefakte, da verschiedene Software-Artefakte primär durch einen abgrenzbaren Personenkreis entwickelt werden. Daher führt die Entwicklung neuer Artefakte, respektive neuer Software-Module, häufig zur Entwicklung einer neuen Sub-Community bzw. eines neuen Sub-Projekts [Tuomi01][175]. Diese Tendenz konnte in allen analysierten Projekten wie z.B. im Rahmen des Linux Kernels und dessen Teilprojekts Linux Documentation Project[176], der verschiedenen Apache Teilprojekte wie z.B. HTTPD, Jakarta und der Jakarta-Teilprojekte oder der einzelnen Mozilla-Teilprojekte (z.B. Quality Assurance, Gecko oder XUL) identifiziert werden (vgl.[Dietze03], [ApacheOrg02][177]). Diese Teilprojektinitialisierung orientiert sich typischerweise an artefaktbezogenen Kriterien, z.B. für spezifische, abgrenzbare Softwareeinheiten, oder nach prozessbezogenen Gesichtspunkten, wie dies im Mozilla-Projekt zur Vereinigung aller qualitätssichernden Aktivitäten in einem Teilprojekt oder im Apache Foundation Projekt für allgemeine organisatorische Aufgaben beobachtet wurde.

[174] „What is clear is that the stark alternative -- a nearly omniscient authority that can predict what route is the most promising to take toward a solution, without actually travelling some distance down at least some of those routes -- is not a realistic counterfactual for complex systems." [Weber00]

[175] „A new artifact, therefore, creates a new community in the context of the originating community. This process leads to increasing differentiation in the social system." [Tuomi00]

[176] http://www.ldp.org

[177] URL: http://www.apache.org/foundation/projects.html; Abfrage: 20.12.2002

A.3 Prozesse zur Konfliktbewältigung bzw. Entscheidungsfindung

Als Ergebnis der verschiedenen, ex- und implizit identifizierbaren Rollen, Verantwortlichkeitsbereiche und Privilegien entsteht in allen Projekten eine informelle, quasi hierarchische Organisation. Da diese Hierarchie aber nur informell existiert, kaum klare Entscheidungsbefugnisse abgrenzbar sind und eine möglichst demokratische Entscheidungsfindung realisiert werden soll, ist eine Formulierung von Entscheidungsfindungsprozessen notwendig [Weber00][178].

Die z.T. demokratischen Strukturen innerhalb der Organisation von OSS-Projekten werden daher durch die Formulierung von dedizierten Prozessen zur Entscheidungsfindung bzw. Konsensbildung unterstützt und ermöglicht, wobei [FoBa02] hierfür den Begriff der *„selbstregulierenden Demokratie“* definiert. Besonders im Kontext der Maintenance durch ein Komitee sind diese Formalismen von großer Bedeutung, um die Entscheidungsfindung zu beschleunigen und nicht durch einen unerreichbaren Konsens zu behindern. Zum Beispiel konnten im Mozilla- und Apache-Kontext verschiedene Abstimmungsverfahren identifiziert werden, welche für Entscheidungen über die Integration von Patches oder die Vergabe von Commit-Privilegien instrumentalisiert werden (vgl. [Dietze03]). Außerdem wird ein Großteil der Konfliktbewältigung durch verschiedene Verhaltensregeln und Normen determiniert, die zum Teil auch in expliziten Dokumenten (Guidelines) definiert werden [Weber00][179].

A.4 Entwicklerfokus auf Implementation

Die Akteure im Kontext von Open Source agieren auf freiwilliger Basis und entscheiden selbst über die von ihnen ausgeführten Aktivitäten. Daher werden nicht alle Aktivitäten mit der gleichen Intensität ausgeführt und Aufgaben, welche als weniger motivierend, wichtig oder ansprechend wahrgenommen werden, werden u.U. nicht im erforderlichen Maß ausgeführt [GoTu00][180]. Entwickler im OSS-Entwicklungsprozess präferieren Aktivitäten, welche direkt zur Befriedigung eines persönlichen Bedürfnisses beitragen [Weber00][181]. Die Akteure im OSS-Entwicklungsprozess tendieren daher dazu, die Implementation von Quellcodemodifikationen anderen, weniger ergebnisorientierten Aktivitäten vorzuziehen [GaLaAr00][182]. Die nachfolgenden Aktivitäten werden daher typischerweise durch die Akteure im OSSD-Prozess vernachlässigt und erfordern eine dedizierte Ergänzung bzw. Unterstützung:

- Softwaredesign
- Software-Test
- Dokumentation
- Support

A.4.1 Design

Die Aktivitäten des Softwareentwurfs werden gar nicht bzw. nur implizit als Bestandteil anderer Prozesse ausgeführt. Sowohl das System Design, als auch das Detail Design werden i.d.R. nicht explizit ausgeführt und ist daher auch nur implizit in verschiedenen Dokumentationsartefakten dokumentiert (vgl. auch [GrTa99][183],

[178] „The organic result is what looks and functions very much like a hierarchical organization where decision making follows fairly structured lines of communication.” [Weber00]

[179] „In open source, much of the important conflict management takes place through behavioral patterns and norms […].” [Weber00]

[180] „This freedom also means that it can be difficult to convince developers to perform essential tasks, such as systematic testing or code restructuring, that are not as exciting as writing new code.” [GoTu00]

[181] „This means that they will tend to focus on an immediate and tangible problem (the 'itch') -- a problem that they themselves want to solve.” [Weber00]

[182] „Open source contributors tend to be more interested in coding than documenting or testing.” [GaLaAr00]

[183] „There is no explicit system-level design, or even detailed design.“ [GrTa99]

[Koch00]). Der detaillierte Softwareentwurf wird i.d.R. als der Implementation immanenter Prozess ausgeführt [Vixie99][184]. Auch das Systemdesign ist entweder implizit in verschiedenen Dokumenten definiert, oder aber es ist vollkommen undokumentiert und evoluiert vergleichsweise frei und unkontrolliert [Vixie99][185]. Die Vernachlässigung des Software-Entwurfs resultiert aber nicht nur aus dem Desinteresse der Entwickler, sondern auch aus dem Umstand, dass in einem Softwareverbesserungsprozess im Gegensatz zu einem vollständigen Software-Entwicklungsprozess kein umfassendes Design erstellt werden muss.

A.4.2 Dokumentation

Die Dokumentation von Quellcode bzw. der implementierten Funktionalitäten ermöglicht den Akteuren, die Intention des Entwicklers nachzuvollziehen. Dies stellt gerade im OSS-Kontext eine fundamentale Voraussetzung für die kollaborative Entwicklung verteilter Akteure an einer gemeinsamen Quellcodebasis dar [GaLaAr00][186]. Die Erstellung von Dokumentationsartefakten als wichtiger Bestandteil der Implementationsaktivitäten wird aber nicht immer mit derselben Intensität wie die eigentliche Implementation ausgeführt und zum Teil als unattraktiver Bestandteil der Systementwicklung wahrgenommen. Dies führt dazu, dass Dokumentationen von OSS häufig nicht aktuell oder in ungenügendem Umfang bereitgestellt werden und dieser Aspekt der Entwicklungsprozesse daher dedizierte Unterstützung erfordert (vgl. [Dietze03]).

A.4.3 Qualitätssicherung: Review und Software-Test

Der Aspekt der Qualitätssicherung findet auf Seiten der dezentralen Akteure z.T. sehr intensive Berücksichtigung und ist impliziter Bestandteil der Anwendungs- und Entwicklungsaktivitäten. Wie bereits dargestellt wurde, werden in diesem Kontext primär Review- und Testaktivitäten ausgeführt, welche inhärenter Bestandteil der Entwicklungs- und Anwendungsprozesse sind und in parallel praktizierten Testprozessen ausgeführt werden [Godfrey00][187]. Als Minimalreview wird das Patch i.d.R. durch einen privilegierten Committer evaluiert. Formalisierte Testprozesse, wie sie im proprietären Software Engineering praktiziert werden, sind in OSSD-Projekten kaum etabliert und werden im OSS-Kontext weitgehend durch die der Anwendung inhärenten Testaktivitäten ersetzt, wobei ein OSS-Projekt von einer möglichst großen Anwendergemeinde profitiert. Auch [Vixie99][188] dokumentiert den Umstand, dass im OSS-Kontext kaum explizite und formale System-Level-Tests ausgeführt werden.

Daher ist die Intensität der Qalitätssicherung zu keinem Zeitpunkt garantierbar und die Fehlerdichte neuer Software-Releases häufig vergleichsweise hoch, zumal die Software-Releases erst die Voraussetzung für die Ausführung der Testaktivitäten darstellen und somit erst im Rahmen des sukzessiven Softwareverbesserungsprozesses ausgeführt werden. Aufgrund der häufig sehr ausgeprägten Field Testing Aktivitäten, welche die gesamte Anwendergemeinde einbeziehen, ist dies aus langfristiger Perspektive aber nicht zwingend als der Softwarequalität abträglicher Aspekt zu betrachten. Es ist aber davon auszugehen, dass ein OSS-Projekt ohne die Existenz einer hinreichend großen Anwendergemeinde, welche u.a. auch das Field Testing ausführt, nicht erfolgreich existieren kann [MoFiHe00][189]
Die Anwendertests von Software-Releases und die damit assoziierte Generierung von Change Requests unterliegt keiner Tendenz analog zur Code Ownership und wird somit verhältnismäßig gleichmäßig durch die gesamte

[184] „Detailed Design ends up being a side effect of the implementation." [Vixie99]

[185] „Either the system design is implicit [...] or it evolves over time (like the software itself)." [Vixie99]

[186] „Good documentation allows people to use – and more specifically in open source projects, to understand and modify – the software." [GaLaAr00]

[187] „Code quality is maintained largely by *‚massively parallel debugging'* (i.e., many developers each using each other's code) rather than by systematic testing or other planned, prescriptive approaches." [Godfrey00]

[188] „But then there's usually no system-level test plan and no unit tests." [Vixie99]

[189] „Open source developments that have a strong core of developers but never achieve large numbers of contributors beyond that core will be able to create new functionality but will fail because of a lack of resources devoted to finding and repairing defects."[MoFiHe00]

Community ausgeführt, wie auch [MoFiHe00] belegt. Trotzdem konnte festgestellt werden, dass einige explizite Testprozesse durch die Core Developer ausgeführt oder zumindest koordiniert werden, wie dies z.B. innerhalb der verschiedenen Phasen im Rahmen eines Releaseprozesses geschieht. Präventive Maintenance des Quellcodes wird durch die verteilten Akteure eines Projekts kaum ausgeführt und wird daher lediglich durch die Core Developer ausgeführt, sofern dieser Aspekt überhaupt Berücksichtigung findet. Diesbezüglich kann daher ein erhebliches Optimierungspotential identifiziert werden [Godfrey00][190].

A.4.4 Anwender- und Entwicklersupport

Der Support der Anwender und Entwickler basiert ebenfalls nicht auf formalisierten Prozessen, sondern auf freiwilliger Interaktion der verteilten Akteure. Das Fehlen eines zentral organisierten Supports bzw. dedizierter Rollen, welche den Anwender- und Entwicklersupport zur Verfügung stellen, hat die Etablierung eines z.T. vergleichsweise gut funktionierenden Systems des kollaborativen Supports auf freiwilliger Basis ermöglicht, wie auch [LaHi00][191] im Kontext des Apache-HTTPD-Projekts identifiziert. Der aktive Support setzt sich aus der Generierung und Veröffentlichung von Dokumentationsartefakten bzw. der Kommunikation von Lösungsvorschlägen zusammen, während der passive Support vor allem auf der Recherche in den verschiedenen Informationsquellen und der Kommunikation von Problemen an die Community besteht, wie in [Dietze03c] detailliert dargestellt wird.

Typischerweise wird der Support über die Mailing Listen und Newsgroups eines Projekts ausgeführt und durch verschiedene Informationsquellen wie z.B. FAQs, Tutorials u.ä. ergänzt (vgl. [Dietze03c], [LaHi00][192]) Als ein Hauptproblem der analysierten Projekte konnte die Tatsache identifiziert werden, dass supportrelevante Informationen in den verschiedenen Artefakten und Informationsquellen existieren und daher die Recherche nach Supportanfragen oder Lösungsoptionen wesentlich erschwert wird. Außerdem ist das zur Verfügung stellen probater Informationen oder Lösungsvorschläge mit einem erheblichen Aufwand für die Informationsprovider verbunden, da sie erst in einer Vielzahl von relativ ungeordneten und unstrukturierten Anfragen recherchieren müssen, um eine für sie relevante Anfrage zu selektieren (vgl. [LaHi00][193]). Dies führt dazu, dass die Supportaktivitäten häufig durch einige wenige Akteure ausgeführt werden, die für die Mehrheit aller Beiträge verantwortlich zeichnen (vgl. [LaHi00][194]). Diese Abhängigkeit könnte durch die Verbesserung der Transparenz wesentlich minimiert und der Prozess des Supports somit stärker parallelisiert und effizienter gestaltet werden.

Aufgrund der Tatsache, dass OSS-Projekte keinen Support garantieren können, da auch dieser Aspekt auf der freiwilligen Mitwirkung der Akteure basiert, haben sich weiterhin bereits eine Vielzahl kommerziell orientierter Unternehmen etabliert, welche ausschließlich Dienstleistungen im Kontext spezifischer OSS-Produkte anbieten [Vixie99][195].

[190] „Planned evolution, testing, and preventive maintenance may suffer, since OSD encourages active participation but not necessarily careful reflection and reorganization." [Godfrey00]

[191] „Despite or because of this lack of ‚official support', a very effective on-line Apache field support system has evolved, operated by and for users themselves." [LaHi00]

[192] „Multiple sources of technical help for Apache users exist in addition to the Usenet help forum, ranging from books to online journals to an online collection of answers to frequently asked questions." [LaHi00]

[193] „[...] the Apache approach has a cost for information providers: multiple experts expend the time to read many questions that they are not able or willing to answer." [LaHi00]

[194] „[...] the system relies heavily on around 100 information providers who in aggregate post 50% of the messages, with the very top few frequent information providers." [LaHi00]

[195] „The lack of support [...] creates opportunities for consultants or software distributors to sell support contracts and /or enhanced and/or commercial versions." [Vixie99]

A.5 Häufige Releasezyklen

Im OSS-Kontext wird angestrebt, Softwaremodifikationen möglichst frühzeitig in Form neuer Software-Releases der gesamten Öffentlichkeit zugänglich zu machen (vgl. [Raymond98], [Nüttgens00]). Damit wird das Ziel verfolgt, möglichst viele Anwender und Entwickler der Software frühzeitig in den Review- und Testprozess zu integrieren. Dabei wird der Entwicklungsquellcode der bisher nur den Nutzern des Konfigurationsmanagementsystems (CVS) verfügbar war, in Form von frei zugänglichen Packages auf der zentralen Website des Projekts veröffentlicht und dadurch eine wesentlich größere Zielgruppe erreicht. Dies beinhaltet sowohl offizielle und getestete neue Versionen der Software als auch Test-Releases, welche lediglich zur Unterstützung der Software-Tests im Vorfeld der Veröffentlichung einer neuen Software Version publiziert werden. Da diese Releases i.d.R. sowohl den Quellcode als auch kompilierte Binärversionen der Software umfassen, kann eine große Anwender- und Entwicklergemeinde adressiert werden, welche das Release anwendet, dabei implizit testet und auch den Quellcode aus Entwicklerperspektive evaluiert.

Diese Releasepraxis impliziert zwar auch den vergleichsweise instabilen und ungetesten Zustand vieler OSS-Releases, kollidiert dabei aber nicht mit den Erwartungen der Anwender [Weber00][196]. Zudem werden ungetestete Vorabversionen der OSS stets auch als instabil (Alpha- oder Beta-Version) gekennzeichnet. Außerdem fördert die häufige Publikation von Software-Releases die Motivation der Entwickler, welche an einer schnellen Integration und Veröffentlichung der von ihnen entwickelten Patches interessiert sind. Einen weiteren Vorteil der schnellen und dynamischen Publikation von Software-Releases stellt nach [MoFiHe00] die damit verbundene Option zur schnellen Reaktion auf Anwenderanforderungen dar. Daher stellt der häufige und zum Teil auch sehr formalisiert ausgeführte Releaseprozess einen explizit zu unterstützenden Prozess dar, der bei der Entwicklung einer geeigneten Softwarestützung entsprechend berücksichtigt werden sollte.

A.6 Softwarequalität und -evolution im OSS-Kontext

Verschiedene OSS-Produkte haben können als Bestätigung für das Potential des OSSD-Modells herangezogen werden, Software von hoher Qualität hervorzubringen, die auch mit den Standards kommerziell entwickelter Software vergleichbar ist. Die Softwarearchitektur und -qualität steht dabei in starker Wechselwirkung mit den zugrundeliegenden Entwicklungsprozessen, da sich beide bilateral beeinflussen und determinieren. Daher werden an dieser Stelle einige allgemeine über die OSS-Produkte der analysierten Projekte und deren Evolution dargestellt, um daraus Implikationen für die Entwicklungsprozesse herleiten zu können.

A.6.1 Anforderungen an die OSS

Der kollaborative Entwicklungsprozess einer gemeinsamen Quellcodebasis stellt spezifische Anforderungen an die Softwarequalität. Basierend auf dem identifizierten Prozessmodell können die folgenden Anforderungen an die Architektur und Qualität des Quellcodes von OSS identifiziert werden, die zwar z.T. auch auf kommerzielle Software anwendbar, aber im OSS-Kontext von besonderer Bedeutung sind:

- Geringe Komplexität
- Modularisierung
- Definierte Schnittstellen
- Hohe Kohärenz, geringe Kopplung
- Lesbarkeit des Quellcodes
- Vermeidung proprietärer Softwarebibliotheken
- Hohe Usability

[196] „Open source user-developers have a very different set of expectations. In this setting, bugs are more an opportunity and less a nuisance." [Weber00]

Ein hoher Grad der Modularisierung ist Voraussetzung für einen verteilten Entwicklungsprozess, bei dem dezentral agierende Entwickler den Quellcode autonom weiterentwickeln (vgl. [Weber00][197]). Dabei ist ein hoher Grad der Kohärenz erstrebenswert, also ein möglichst starker funktionaler Zusammenhang des Quellcodes eines speziellen Moduls, dass somit nicht sinnvoll in weitere Module zerlegt werden kann. Zwischen den Modulen sollte eine möglichst geringe Kopplung, also eine geringe Anzahl von Abhängigkeiten existieren [Weber00][198]. Diese Kriterien reduzieren die Auswirkungen einer Quellcodemodifikation auf ein Minimum und stellen somit elementare Voraussetzungen für die Verringerung der Komplexität des Gesamtsystems aus Entwicklerperspektive dar. Die Definition von einfachen Schnittstellen für die einzelnen Subsysteme trägt somit ebenfalls zur Komplexitätsverringerung bei und ermöglicht den Entwicklern somit eine weitgehend vom Gesamtsystem unabhängige Entwicklung des Quellcodes eines Moduls.

Weiterhin muß der Quellcode von OSS in einem allgemein verständlichen und möglichst lesbaren Zustand entwickelt werden, um die Einstiegsbarrieren für neue Entwickler zu minimieren [Pavlicek00][199]. Dazu trägt außerdem die Verwendung von ausschließlich offenen und frei verfügbaren Quellcodeelementen bei (vgl. [Weber00][200]).

Da eine möglichst große Anwendergemeinde eine Prämisse für die umfassende Weiterentwicklung von OSS darstellt, sollte der Grad der Usability der Software möglichst hoch sein, um eine größtmögliche Zielgruppe mit der OSS erreichen zu können und eine weitreichende Adaption der Software zu ermöglichen. Dies wird aber nicht oder nur mangelhaft im OSS-Kontext gewährleistet und sollte daher im Rahmen der Qualitätssicherung explizit berücksichtigt werden [NiThYe02][201].

A.6.2 Implikationen für die weitere Prozessoptimierung und -unterstützung

Die verschiedenen Erkenntnisse bezüglich der Softwarequalität und der Anforderungen an OSS führen zu einigen Implikationen bezüglich der OSS bzw. für einen verbesserten Entwicklungsprozess. Die Ergebnisse verschiedener Studien der OSS-Architektur bzw. deren Evolution führen zu der Erkenntnis, dass ein möglichst freier Entwicklungsprozess auch zu Software von hoher Qualität entsprechend den klassischen Kriterien einer effizienten Softwarearchitektur führen kann (vgl. [Dietze03], [BoHoBr99], [Godfrey00], [CaLaMo03], [MaGo03], [MaGo03a], [Tuomi01], [GrTa99]). Dies setzt aber stets die Etablierung einer Community von hinreichender Größe voraus.
Daher sollten die Entwicklungsprozesse, welche zur Erstellung der OSS geführt haben, nicht weiter reglementiert und formalisiert, sondern lediglich durch verschiedene unterstützende Elemente erweitert werden. Die Vorteile des im OSS-Kontext praktizierten Selektionsprinzip im Rahmen der Quellcodeentwicklung im Sinne einer evolutionären Durchsetzung des hochwertigsten Quellcodes sollten nicht durch eine Determinierung dieser freien Prozesse eingeschränkt oder behindert werden. [Pavlicek00] beschreibt dies zum Teil mit dem Terminus „*Self Correcting Code*“, der durch eine umfangreiche Entwicklergemeinde konsequent weiterentwickelt wird.

Die häufige Aktualisierung des Quellcodes und die Vielzahl der sogenannten Low-Level-Changes durch die verschiedenen Entwickler stellen zwar ein permanentes Risiko für einen kontinuierlich evoluierenden Quellcode dar und wirken der Einhaltung von konsistenten Entwicklungsvorgaben entgegen (vgl. [Godfrey00],

[197] „The key to managing the level of complexity within the software itself, is modular design.“ [Weber00]

[198] „Good design and engineering is about limiting the interdependencies and interactions between modules.” [Weber00]

[199] „If the code is cryptic, undocumented or filled with obsolete routines, it will confuse and frustrate people as they seek to comprehend the code.“ [Pavlicek00]

[200] „An open source process will work more effectively when Contributions depend not on proprietary techniques but on knowledge that is widely available.” [Weber00]

[201] „[...] open-source development methods may need to adapt if they are to produce software for the desktop of the typical user. A community of developers will not necessarily pay sufficient attention to issues of usability that they themselves do not experience.” [NiThYe02]

[TrGoLeHo00][202]), wobei dies aber z.T. durch ein implizit ausgeführtes, kontinuierliches Refactoring des Quellcodes durch die Entwickler im Rahmen der Entwicklungsaktivitäten kompensiert wird (vgl. [Reis00][203]).

Trotzdem identifiziert [TrGoLeHo00][204] die Tendenz, dass die konkrete Architektur sich im Verlauf eines Projekts zunehmend von der konzeptuellen OSS-Architektur unterscheidet, was vor allem aus der fehlenden Koordination und Kontrolle der Quellcodemodifikationen resultieren könnte. Dies sieht [TrGoLeHo00][205] hauptsächlich dadurch begründet, dass die verschiedenen Entwickler individuell verschiedene und miteinander z.T. konkurrierende Entwurfsvorstellungen implementieren. Daher sollten sowohl die vor dem Commit eines Patches als auch die vor dem Release einer neuer Software-Version ausgeführten Reviewprozesse durch Etablierung unterstützender Prozesse ergänzt und erweitert werden.

A.7 Tendenz zum Boot Strapping

Wie im Rahmen der Projektstudien festgestellt und auch in [Halloran01] dokumentiert wurde, existiert in OSS-Projekten eine Tendenz zur Adaption oder Eigenentwicklung von ausschließlich OS-basierten Software-Werkzeugen und eine sehr geringe Akzeptanz von proprietärer Software zur Unterstützung der Entwicklungsprozesse. Dieses Merkmal wird auch als Boot Strapping bezeichnet [Halloran01][206]. Die Verwendung ausschließlich OS-basierter Software-Werkzeuge repräsentiert daher in vielen OSS-Projekten eine elementare Erfolgsvoraussetzung. Neben der ideologischen Motivation existieren verschiedene pragmatische Beweggründe für diese Tendenz. Zum einen senkt die Verwendung von OSS, die somit frei für jeden Akteur verfügbar ist, die Einstiegsbarrieren für neue Akteure und für Projektinitiatoren [Halloran01][207]. Außerdem entspricht die mit der OSS verbundene Möglichkeit zur eigenständigen Anpassung und Verbesserung der involvierten Werkzeuge der OSS-Philosophie. Um eine optimale Akzeptanz der zu entwickelnden Software-Infrastruktur durch die OSS-Community zu ermöglichen, muß dieser Aspekt bei der Entwicklung einer unterstützenden Software-Infrastruktur berücksichtigt werden.

[202] „System changes are often done without considering their effects on the system structure." [TrGoLeHo00]

[203] „OSS projects have been known to refactor continuously to avoid architecture breakdown." [Reis00]

[204] „The problem of architectural drift is especially pronounced in open source systems, where many developers work in isolation on distinct features with little co-ordination." [TrGoLeHo00]

[205] „Different developers may develop distinct, personal views of the system's architecture and their changes may interfere with one another." [TrGoLeHo00]

[206] „[...] many open-source projects engage in boot-strapping - the use of open source tools on open source projects." [Halloran01]

[207] „[...] it lowers the barrier to entry for new participants in an open source project, enabling participants to create their personal (client-side) developer sites at low cost and without reliance on particular products." [Halloran01]

B Unterstützende und erweiterte Prozesse

B.1 Software-Test

Ergänzend zu den impliziten Software-Tests des deskriptiven Prozessmodells, die als der Anwendung immanente Aktivitäten implizit identifiziert werden konnten, scheint die Definition von dedizierten Testaktvitäten sinnvoll. Dadurch könnte zu jedem Zeitpunkt der Quellcodeentwicklung ein definierter Qualitätsstandard der OSS sichergestellt werden, was durch die ausschließliche Durchführung der impliziten Testaktivitäten durch die Community nicht immer gewährleistet werden kann. Diese Software-Tests sollten die verschiedenen Software-Releases nach den folgenden Kriterien evaluieren:

- Funktionalität
- Performance
- Usability

Neben der Funktionalität und der Performance stellt die vor allem die Usability im Kontext von OSS ein wichtiges Kriterium dar, da nur eine hinreichend gute Bedienbarkeit die Adressierung einer größtmöglichen Anwenderzielgruppe ermöglicht.

Software-Tester sollten ergänzend zu den Anwendungstests der Community eine vorher definierte Menge von Testfällen insbesondere mit dem Quellcode ausfürhen im Rahmen eines Releaseprozesses als Bestandteil eines Software-Release veröffentlicht werden soll. Diese verschiedenen Test Cases können dabei in den verschiedenen Phasen der Quellcodeentwicklung bzw. zwischen verschiedenen Kategorien von Software-Releases (Test-Release, GA-Release) variieren und sollten in entsprechenden Test-Guidelines beschrieben werden. Die somit garantierbare Durchführung von expliziten Testaktivitäten wird daher vorrangig in den verschiedenen Phasen des Releaseprozesses durchgeführt, um das Qualitätsniveau eines Release bereits in den frühen Phasen seiner Reife (Alpha-, Beta-, GA-Release) zu verbessern. Außerdem erscheint die periodische Durchführung von Software-Tests des aktuellen Entwicklungsquellcodes sinnvoll, wobei diesen Aktivitäten aber die Erstellung von dedizierten, kompilierten Binärversionen des Quellcodes vorausgehen sollte, welche lediglich zu Testzwecken generiert werden. Diese Aktivitäten werden in *7.1.4 Erweiterter Releaseprozess* explizit erläutert.

Die Ergebnisse der Tests sollten, falls ein Überarbeitungsbedarf identifiziert wurde, an den Release Manager kommuniziert werden, welcher für die Durchführung der weiteren Aktivitäten zuständig ist. Zudem können die Software-Tester einen Change Request generieren, in welchem sie die durchzuführenden Quellcodemodifikation definieren.

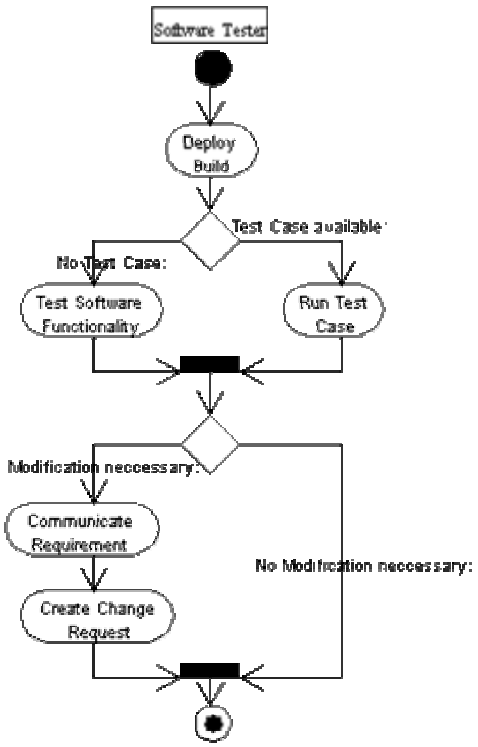

Abb. B.1: Aktivitätssicht *Test Software*

Falls der Quellcode einen bestimmten Umfang überschreitet, erscheint es sinnvoll, die Testaktivitäten entsprechend abgrenzbarer funktionaler Bereiche oder entsprechend der Subsysteme auf verschiedene Akteure zu verteilen (vgl. [FoBa02]). Außerdem erscheint es sinnvoll, Tests zu automatisieren, soweit dies durch die Spezifika der Software ermöglicht wird [FoBa02][208]. Dies kann manuelle Tests natürlich nur ergänzen und unterstützen, aber nie vollständig ersetzen.

Rolle: Software-Tester
Die dedizierte Rolle des Software-Tests erfordert aufgrund der hohen Formalisierbarkeit der Aufgaben kaum explizite Qualifikationen und kann daher auch durch weniger qualifizierte Anwender ausgeführt werden, um Entwicklungsressourcen für anspruchsvollere Aufgaben verwenden zu können. Die Rolle der Software-Tester sollte sinnvollerweise einer Gruppe von Akteuren und keiner Einzelperson zugewiesen werden, um die verschiedenen Testaktivitäten probat verteilen zu können. Die Ausführung der Software-Tests sollte in enger Kooperation mit den Build Coordinatoren (vgl. *7.1.4 Erweiterter Releaseprozess*) erfolgen, welche das Feedback der Tester sammeln und etwaige weitere Schritte koordinieren.

[208] „Bei jeder Gelegenheit, bei der das Programm diskrete Eingaben erwartet und vorhersagbare Ausgaben produziert, kann man theoretisch automatisch testen lassen.“ [FoBa02]

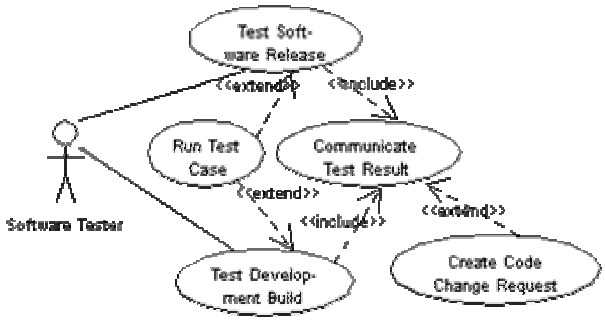

Abb. B.2: Use Case Sicht des Software Testers

B.2 Test Case Development

Um den Prozess der Software-Tests zu automatisieren, sollten regelmäßig dedizierte Testfälle entwickelt werden, welche die konkreten Aktivitäten der Software-Tester beschreiben und dadurch eine Formalisierung der Software-Tests ermöglichen. Mit den Test Cases wird somit das Ziel verfolgt, die Kriterien, welche durch die Tests evaluiert werden sollen (Funktionalität, Performance, Usability), systematisch zu überprüfen und dabei quantitative oder qualitative Bewertungen der verschiedenen Kriterien zu ermöglichen. Die definierten Test Cases sind keine zwingend für den dargestellten Prozess der Software-Tests notwendigen Artefakte, können diesen aber vereinfachen und standardisieren.

Rolle: Test Case Developer
Die Test Cases werden durch die dedizierte Rolle des Test Case Developers definiert, der über eine sehr umfangreiche Kenntnis der OSS und deren Anforderungen verfügen muss, um in der Lage zu sein, geeignete Testszenarien zu entwickeln, die eine möglichst umfangreiche Evaluierung der Software ermöglicht. Die Rolle des Test Case Developers repräsentiert daher einen sehr anspruchsvollen Aufgabenbereich, der nur durch eine entsprechend qualifizierte Personengruppe ausgeführt werden kann.

B.3 Management der Content- und Dokumentationsartefakte

Wie bereits dargestellt wurde, kann in OSS-Projekten häufig nicht die ausreichende Entwicklung und Veröffentlichung aktueller Content- und Dokumentationsartefakte gewährleistet werden. In einigen Projekten wird versucht, dies durch die Generierung eines dedizierten Subprojekts zu kompensieren [GaLaAr00][209]. Die hier dargestellten Prozesse dienen der Koordination der kollaborativen Dokumentationsentwicklung und sind daher auch nur von unterstützendem Charakter.

Neben den klassischen Dokumentationsobjekten existiert eine Vielzahl weiterer Content-Objekte, wie z.B. die Inhalte der Website. Die Organisation und der Review aller Content-Artefakte inkl. der klassischen Softwaredokumentationen (vgl. *7.2.3 Content-Artefakte*) sollten gesamtheitlich durch diesen Prozess unterstützt werden. Eine wichtige Voraussetzung zur optimalen Durchführung dieses Prozesses ist die Verwaltung aller Content-Objekte in einem zentralen Content Management System (vgl. *7.3 Unterstützende Software-Infrastruktur*). Dies ermöglicht z.B. die dezentrale Erstellung und Aktualisierung der Webseiteninhalte des Projekts durch die gesamte Community und die damit verbundenen Workflows.

[209] „There has been some effort in addressing the problem of lack of documentation (e.g. the Linux Documentation Project and Mozilla Developer Documentation web page), but this is still a rarity for smaller open source projects." [GaLaAr00]

Koordination der Entwicklung der Content-Artefakte
Um eine möglichst aktuelle und lückenlose Dokumentation der Software aus Anwender- und Entwicklerperspektive zu ermöglichen und einen aktuellen Stand aller Contentartefakte zu gewährleisten, ist es notwendig, die Aktivitäten zur Dokumentationserstellung in bestimmtem Maße zu überwachen und zu steuern. Diese Aktivitäten beinhalten den periodischen Review der Dokumentations- und der Content-Artefakte, wobei stets die Spezifika des aktuellsten Quellcodes des dokumentierten Software-Artefakts berücksichtigt werden sollten, um größtmögliche Aktualität zu gewährleisten. Für diese Aufgabe sollte eine dedizierte Rolle, die des Content Managers vorgesehen werden. Im Rahmen dieses allgemeinen Koordinationsprozesses ist es notwendig, die Inhalte aller Informationsartefakte zu betrachten und in Bezug auf eine konsistente und nicht redundante Zuordnung der Inhalte zu spezifischen Artefakten sicherzustellen. Falls eine Überarbeitung von Artefakten als notwendig erachtet wird, erstellt der Content Manager einen entsprechenden Content Change Request, in welchem er die erforderlichen Änderungen bzw. das neu zu erstellende Artefakt definiert.

Review von Content-Artefakten
Eingehende Content-Artefakte sollten durch den Content Manager einem ersten Review unterzogen werden. Dies wird in den meisten OSS-Projekten abhängig von der Art des Content-Artefakts als informeller Prozess durch den Maintainer oder die Core Developer ausgeführt. Da die Bandbreite, Formate und Zielsetzungen der verschiedenen Contentartefakte aber sehr groß ist, kann dieser Prozess kaum formalisiert oder automatisiert werden.

Der Content Manager sollte desweiteren einen Styleguide definieren und verwalten, welcher die Rahmenbedingungen, die Struktur und die relevanten Inhalte aller Content-Artefakte definiert und daher die Vorgaben für die kollaborative und dezentrale Erstellung der Artefakte durch die Community definiert. Im Rahmen des Reviewprozesses evaluiert der Content Manager die Artefakte anhand des Styleguide, analysiert die Inhalte und die Struktur, koordiniert nötigenfalls direkt die Überarbeitung des Artefakts mit dem verantwortlichen Contributor oder nimmt minimale Änderungen selbständig vor. Nach erfolgreich durchgeführtem Review sollte der Content Manager die Publikation des Artefakts über die zentralen Kollaborations- bzw. Kommunikationswerkzeuge durchführen. Sofern ein Artefakt für die Publikation eines Software-Release von Bedeutung ist, sollte der Content Manager dies mit dem Release Manager koordinieren.

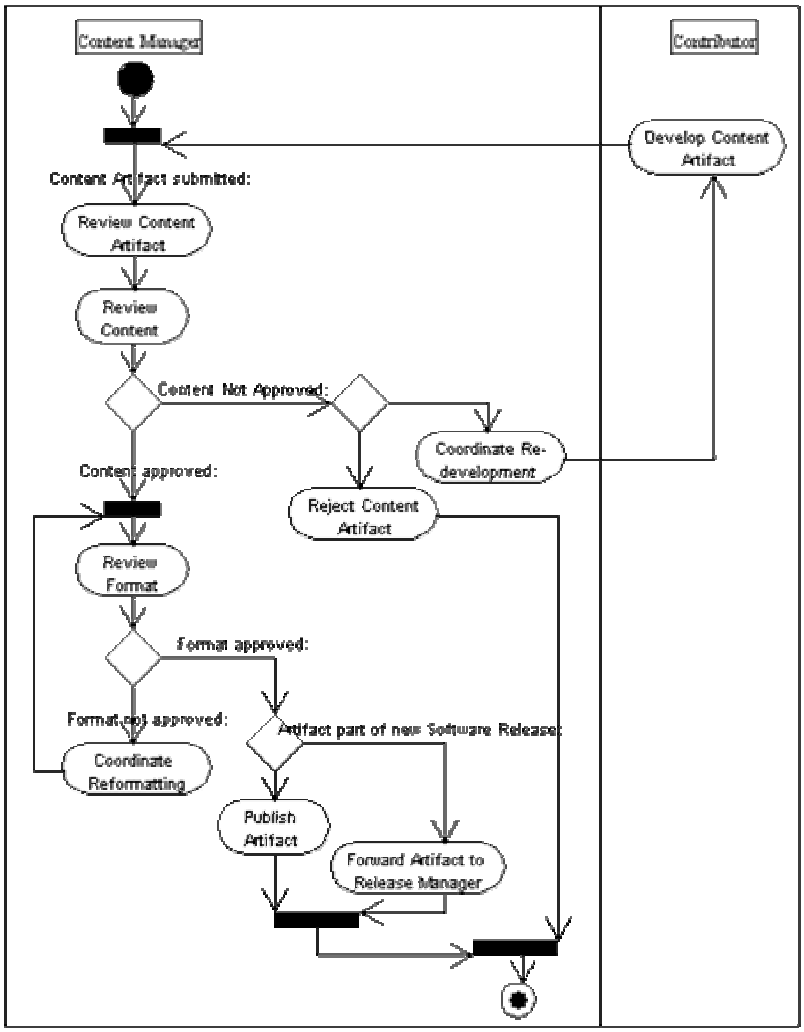

Abb. B.3: Aktivitätssicht des Review und der Publikation von Content-Artefakten

Rolle: Content Manager

Die Aktivitäten zur Verwaltung und dem Review der Content-Artefakte sollten durch einen Akteur ausgeführt werden, der dadurch die Rolle des Content Managers ausübt. Diese Rolle ist nach [Behlendorf99] besonders in OSS-Projekten notwendig und wird momentan häufig nicht berücksichtigt. Dieser Akteur sollte über eine sehr gute Kenntnis der Dokumentation, des Projekts und der Software verfügen, da er neben allgemeinen Webinhalten auch die Inhalte der Dokumentation in Bezug auf bestimmte Quellcodeartefakte evaluiert. Er repräsentiert die zentrale Kontaktperson für alle dezentral ausgeführten Aktivitäten zur Entwicklung von Dokumentationsobjekten und Inhaltsartefakten.

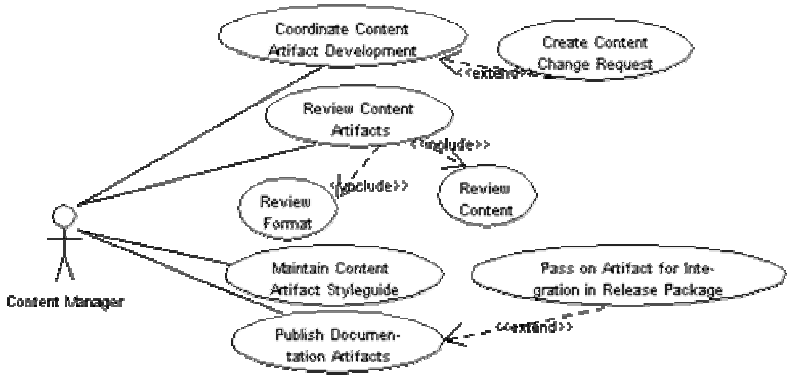

Abb. B.4: Use Case Sicht des Content Managers

B.4 Communication Review

Aufgrund des besonders hohen Informationsaufkommens (vgl. auch [Edwards01][210]) im Rahmen von OSS-Projekten stellen sich besondere Anforderungen an die Verwaltung der Informationsressourcen. Es ist zudem nicht stets gewährleistet, dass ein Kommunikationskanal von den Akteuren ausschließlich für die Kommunikation von explizit dafür vorgesehenen Inhalten instrumentalisiert wird. Zudem existieren für verschiedene Kommunikationsszenarien z.T. keine dezidierten Kommunikationskanäle. Dies führt zu einem u.U. diffusen Informationsaufkommen, in welchem wichtige Informationen nicht zwingend von allen Akteuren als solche wahrgenommen werden können. Daher erscheint es sinnvoll, die Kommunikation im Rahmen eines OSS-Projekts kontinuierlich zu überwachen, um entsprechend wichtige Informationen hervorzuheben, über die probaten Kommunikationswege zu kanalisieren oder notwendige Konsequenzen als Reaktion auf die Information einzuleiten (vgl. [Edwards01][211]).

Diese Kontrolle kann als permanenter Prozess durch eine dedizierte Rolle ausgeübt werden, und soll die Kommunikation der Akteure nicht einschränken, zensieren oder reglementieren, sondern lediglich ergänzen und unterstützen, indem wichtige Informationen über die probaten und explizit dafür vorgesehenen Kommunikationswege erneut transportiert werden. Dies erscheint sinnvoll, um verschiedene Softwareanforderungen der Anwender zu identifizieren, die über die Kommunikationskanäle diskutiert werden, oder um infrastrukturelle, organisatorische oder sonstige Anforderungen an das Projekt zu identifizieren und somit stets zu gewährleisten, dass die Rahmenbedingungen der kollaborativen Entwicklung möglichst an den Bedürfnissen der Community ausgerichtet sind.

[210] „The Linux kernel development is an example of a project with a high traffic mailing list. In week 20 year 2001 the Linux kernel mailing list received 1227 posts from 423 different contributors […].“ [Edwards01]

[211] „Participation in development requires a person to read or scan a lot of emails referred to as high traffic.“ [Edwards01]

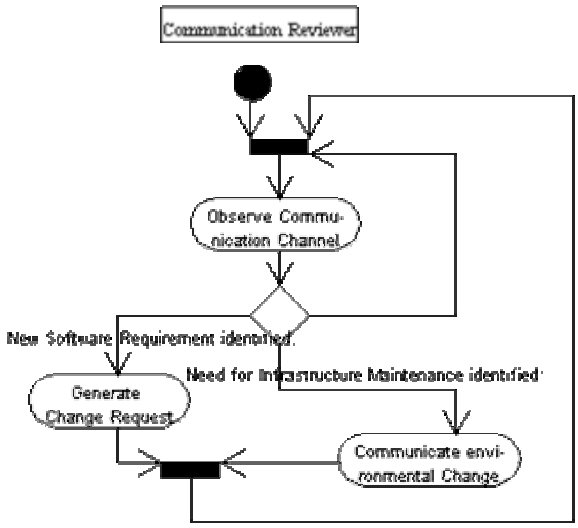

Abb. B.5: Aktivitätssicht *Communication Review*

Rolle: Communication Reviewer
Die Rolle des Communication Reviewers sollte durch einen Akteur ausgeübt werden, der möglichst umfangreiche Kenntnisse über die Anforderungen des Projekts, die instrumentalisierte Infrastruktur und die organisatorischen Merkmale verfügt. Daher erscheint es sinnvoll, dass diese Aktivitäten durch ein Mitglied der Maintainer bzw. der Core Developer des Projekts ausgeführt werden.

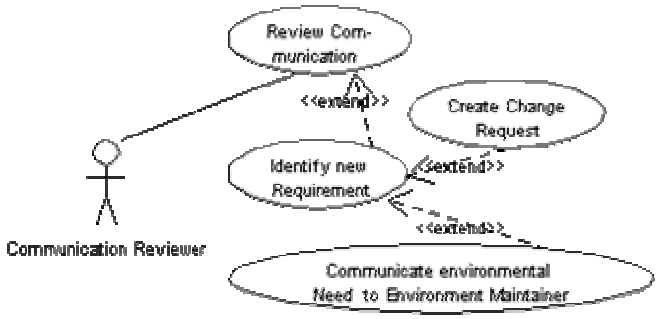

Abb. B.6: Use Case Sicht des Communication Reviewers

B.5 Maintenance der Infrastruktur durch dezidierte Rolle

Die Maintenance-Aufgaben im Rahmen eines OSS-Projekts lassen sich in die Managementprozesse zur Verwaltung, Organisation und Koordination der Community und die infrastrukturellen Prozesse zur Etablierung und Wartung einer der Infrastruktur einteilen. Da beide Prozesse durch Akteure der Maintenance-Institution und häufig durch lediglich einen Akteur (Maintainer) ausgeführt werden, erscheint es sinnvoll, die infrastrukturellen Aktivitäten durch eine dezidierte Rolle, den Environment Maintainer, ausführen zu lassen und dadurch eine striktere Aufgabentrennung zu ermöglichen.

Im Rahmen dieser Aktivitäten ist eine enge Kooperation mit den organisatorischen Maintainern eines Projekts notwendig, zumal der Environment Maintaier häufig direkt auf die Vorgaben der Maintainer angewiesen ist. Die Etablierung einer dezidierten Rolle für infrastrukturelle Aufgaben führt zu einer klareren Aufgabentrennung und

der Entlastung der Maintainer, die sich dadurch ausschließlich auf die Organisation der Community und die Koordination und Steuerung der Entwicklungsprozesse konzentrieren können. Tätigkeiten des Environment Maintainers umfassen zum Beispiel die Etablierung neuer Kommunikations- oder Kollaborationssysteme, die Einrichtung dedizierter Kommunikationskanäle oder die systemtechnische Umsetzung der Rollenkonzepte über eine dedizierte Rechtevergabe, beispielsweise im Request Tracking System des Projekts.
Außerdem ist die Verwaltung der Kollaborationsumgebung (vgl. *2.2.4 Werkzeuge*) in diese Aktivitäten integriert, indem zum Beispiel die Metadaten von Artefakten wie den Change Requests oder die verschiedenen Quellcodebranches durch den Environment Maintainer verwaltet oder realisiert werden. Die Identifikation eines infrastrukturellen Handlungsbedarfs erfolgt dabei primär durch den Environment Maintainer und den Communication Reviewer, wobei aber alle Akteure eine entsprechende Verbesserungsanfrage an die Maintainer richten können. Da diese Aktivitäten sehr informell und individuell verschieden praktiziert werden, wird an dieser Stelle auf die Modellierung einer Aktivitätssicht verzichtet.

Rolle: Environment Maintainer
Der Environment Maintainer führt die verschiedenen infrastrukturellen Prozesse aus, die i.d.R. chronologisch und inhaltlich unabhängig voneinander und entsprechend des identifizierbaren Handlungsbedarfs einzelfallspezifisch ausgeführt werden. Dafür ist der Environment Maintainer stark auf den Input anderer Akteure angewiesen. Zum Beispiel erhält er die Vorgaben für die systemtechnische Realisierung der Nutzerrechte von den Maintainern bzw. den Core Developers, während verschiedene andere Aktivitäten durch die Informationen des Communication Reviewers oder die dargestellten Maintenance Requests (vgl. *7.2.2.4 Maintenance Request*) einer gesamtheitlichen Task/Issue-Datenbank ausgelöst werden können.

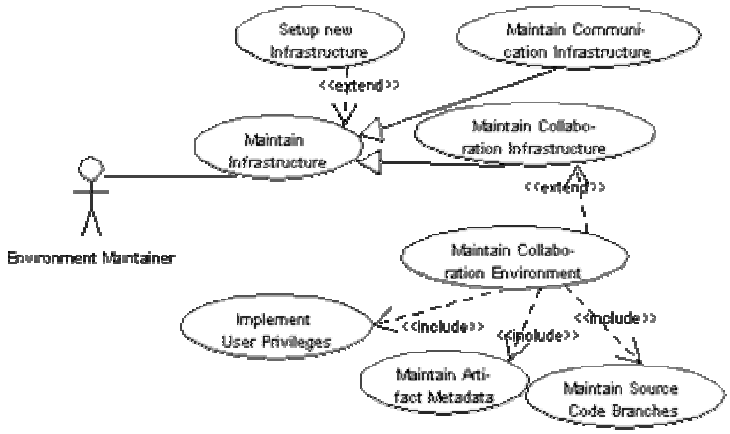

Abb. B.7: Use Case Sicht des Environment Maintainers

B.6 Erweiterte Managementprozesse

Durch die dargestellte Aufgabenteilung wird es ermöglicht, dass die Maintainer eines Projekts ausschließlich für die im Metamodell definierten Managementprozesse zuständig sind. Diese beinhalten im Kern die Etablierung und Organisation der Community und die unterstützende Koordination der kollaborativen Entwicklungsprozesse. Auf der Basis der bereits dargestellten modifizierten Prozesse und Rollen umfasst dies beispielsweise die folgenden Aktivitäten, welche die Managementprozesse des deskriptiven Prozessmodells z.T. erweitern:

- Etablierung einer Projekt-Community
- Steuerung der strategischen Ausrichtung des Projekts
- Etablierung und Organisation von Subprojekten
- Etablierung und Unterstützung der Core Developer
- Wartung der Managementartefakte

- Zuweisung und Organisation der zentraleren bzw. managementorientierten Rollen
- Etablierung demokratischer Prozesse zur Konfliktlösung und Entscheidungsfindung
- Definition der Vorgaben für die infrastrukturelle Wartung
- Publikation aller projektrelevanten Informationen
- Etablierung und Anpassung des Lizenzmodells
- Organisation der externen Kommunikation und externer Rechtsgeschäfte (Etablierung einer Rechtsform)

Bei der Entwicklung der Community ist es von großer Bedeutung, den meritokratischen Charakter des sozialen Systems OSS-Projekt hervorzuheben und eine Meritokratie zu etablieren, die jedem Akteur den leistungsspezifischen Zugewinn von Privilegien ermöglicht.

Etablierung einer Rechtsform

Die Tätigung rechtsgültiger Geschäfte setzt die Etablierung einer Rechtsform bzw. einer juristischen Person voraus, welche der jeweiligen Rechtslage entsprechend als fähig zu juristischen Geschäften betrachtet werden kann. Dies konnte z.B. im Kontext des Apache- und des Mozilla-Projekts identifiziert werden (vgl. [Dietze03]) und wird auch in [Scacchi02] und [Grassmuck02][212] als eine sinnvolle Aktivität beschrieben. Die Etablierung einer Rechtsform ermöglicht z.B. die Schließung rechtskräftiger Verträge über die Kooperation mit kommerziellen Unternehmen, die Mitwirkung in Industriekonsortien, die Entgegennahme von Spenden oder den Schutz projektbezogener Marken oder Warenzeichen. Als Beispiele sind neben der Apache Software Foundation (ASF) oder der XFree86Project, Inc. die Free Software Foundation (FSF) zu nennen, welche aus dem GNU-Projekt hervorgegangen ist und nach [Grassmuck02] die erste rechtsfähige Institution repräsentiert, die direkt aus einem konkreten OSS-Entwicklungsprojekt hervorgegangen ist.

[212] „Sobald jedoch die Außenverhältnisse zunehmen, sehen die meisten Projekte es als vorteilhaft an, sich eine Rechtsform zu geben. [...] Daher bilden sich meist parallel zu den weiterhin offenen Arbeitsstrukturen formelle Institutionen als Schnittstellen zu Wirtschaft und Rechtssystem.“ [Grassmuck02]

C Erweiterte bzw. optimierte Artefakte

C.1 Managementartefakte

In diesem Abschnitt werden neue Managementartefakte eingeführt bzw. die bereits im deskriptiven Modell identifizierten Managementartefakte um Eigenschaften ergänzt.

C.1.1 Strategische Managementartefakte

Die folgenden Artefakte werden gemäß der Definition des verwendeten Metamodells als strategische Managementartefakte kategorisiert.

C.1.1.1 Softwarelizenz – GNU GPL

Es konnten im Rahmen der Entwicklung des generalisierten Prozessmodells verschiedene Lizenzmodelle identifiziert werden, deren Spezifikationen aber inzwischen alle den Bestimmungen der GNU GPL (vgl. [FSF03][69]) entsprechen bzw. kompatibel zu diesem Lizenzmodell sind. Da sich dieses Lizenzmodell an der bereits dargestellten OSD ausrichtet, welche eine adäquate Unterstützung der OSS-Entwicklungsprozesse in vielen Projekten bereits unter Beweis gestellt hat und somit als geeignete Methode zur Definition der Rahmenbedingungen für die kollaborative Softwareentwicklung betrachtet werden kann, sollten OSSD-Projekte auf dieses Lizenzmodell zurückgreifen und von der aufwandsintensiven Definition eines individuellen Lizenzmodells absehen. Ein weiterer wichtiger Aspekt ist der zu erwartende Mangel an Akzeptanz eines OSD- oder GNU GPL-inkompatiblen Lizenzmodells bei der OSS-Community. Auch die Evolution der Lizenzmodelle der untersuchten Projekte, respektive der ASL (Apache Software License) und der MPL (Mozilla Public License), unterstützt diese Behauptung, da diese Lizenzen zwar ursprünglich nicht zur GNU GPL kompatibel waren, aber im Verlauf des jeweiligen Projekts optimiert werden mussten und sich dadurch stetig den Spezifika der GNU GPL angenähert haben und inzwischen vollständig konform zu der GNU GPL sind.

Diese Vorgabe lässt sich natürlich ausschließlich für das Lizenzmodell eines vollständig unkommerziellen und OSS-basierten Projektansatzes definieren. Um Elemente des Prozessmodells der OSSD in einem kommerziell orientierten, proprietären Projekt zu instrumentalisieren, ist die Definition eines angepassten, dezidierten Lizenzmodells notwendig und von entscheidender Bedeutung für den Erfolg des Projekts.

C.1.1.2 Prozessunterstützende Projekt-Richtlinien

Die Erstellung von prozessbeschreibenden und reglementierenden Projekt-Richtlinien oder –Leitfäden (Guidelines) unterstützt die Herstellung eines gemeinsamen Konsens und Verständnisses über zentrale Aspekte und Prozesse des Projekts und ermöglicht dadurch einen hohen Grad der Prozessautonomie und –parallelisierung. Diese sozio-technologischen Vereinbarungen sind im OSS-Kontext i.d.R. praktikabler als die erschöpfende technologische Realisierung und Reglementierung aller Prozesse und Rollen [Cubranic01][213]. Die Verfassung von Guidelines, die vor allem beschreibende Modelle des jeweiligen Prozesse beinhalten, forciert die Adaption der dargestellten Prozesse durch die Akteure, die somit die deskriptiven Modelle als präskriptive Prozessdefinitionen anwenden. Diese Methode einer undogmatischen Prozesssteuerung entspricht dem Charakter der OSS-Entwicklung und wird daher mit Erfolg bereits in den verschiedenen OSS-Projekten praktiziert (vgl. [Dietze03]). Da sich die elementaren Prozesse in den einzelnen Projekten stark ähneln, können die individuell definierten Guidelines z.T. auch projektübergreifend genutzt werden.

[213] „[…] developing 'social contracts' among participants in computer-mediated communication is often more effective than looking for a technological solution [...] ." [Cubranic01]

Die folgenden Kategorien von Guidelines konnten in verschiedenen Projekten identifiziert werden und werden als besonders notwendig für die adäquate Durchführung der kollaborativen Entwicklungsprozesse und zur Minimierung der Einstiegsbarrieren für interessierte Anwender betrachtet:

- Mailing Listen-Leitfaden
- Allgemeine Verhaltensrichtlinien
- Bug Reporting-Leitfaden
- Patchentwicklungsleitfaden (inkl. Verweis auf zu verwendende Programmierkonventionen)
- Dokumentationserstellungsrichtlinien (inkl. Verweis auf Styleguide)

C.1.1.3 Programmierkonventionen

Um den Prozess der lokalen Patchentwicklung zu unterstützen und die erstellten Artefakte (Patches) zu standardisieren, werden i.d.R. Programmierkonventionen definiert. Dies trägt auch zur Lesbarkeit des Quellcodes bei und sollte auch die Erstellung einer möglichst modularen Quellcodebasis forcieren. Durch die Gewährleistung von konsistenten Programmierstandards kann u.a. die Lesbarkeit des Quellcodes verbessert und somit die Eintrittsbarrieren für neue Entwickler minimiert werden [KiLe00][214].. Derartige Konventionen können zudem den Aufwand des Review des Quellcodes reduzieren und unterstützen eine bessere Integrierbarkeit in die gesamte Systemarchitektur des Entwicklungsquellcodes [KiLe00][215].

Ein derartiges Artefakt ermöglicht neben der verteilten Entwicklung auch die Standardisierung eines etwaigen Refactoring-Prozesses (vgl. [KiLe00][214]) Die folgenden Aspekte sollten in den Programmierkonventionen berücksichtigt und konkretisiert werden:

- Namenskonventionen
- Existierende Gesamtarchitektur
- Softwarequalität
- Modularisierung und Schnittstellen
- Verwendung standardisierter Quellcodeelemente
- Quellcodelayout
- Quellcodespezifische Dokumentation
- Integration des Lizenzmodells

C.1.1.4 Documentation (Content-) Style Guide

Wie bereits in *B.3 Management der Content- und Dokumentationsartefakte* dargestellt wurde, sollte die Dokumentationsentwicklung durch einen zentral definierten Styleguide formalisiert und somit auch die erstellten Dokumentationsartefakte determiniert werden. Um eine bestmögliche Verwendbarkeit der Dokumentationsartefakte sicherzustellen, sollte ein derartiges Artefakt die Verwendung von standardisierten Formaten vorschreiben. Üblicherweise werden Formate wie ASCII-Text oder HTML verwendet, da diese Standards i.d.R. von jedem Akteur verwendet und modifiziert werden können. Häufig werden auch Dokumentationsartefakte in Form von PDF-Dateien erstellt.
Ein Styleguide sollte dabei möglichst wenige und für verschiedene Kategorien von Dokumentationsartefakten jeweils nur ein Format definieren. Beispielsweise sollten quellcodespezifische Dokumentationen wie z.B. die weit verbreiteten ReadMe-Dateien, als reiner ASCII-Text erstellt werden, während allgemeinere, weniger quellcodebezogene Dokumentation z.B. häufig als HTML- und/oder PDF-Dokument erstellt wird.

[214] „Coding standards are necessary to support rapid familiarization and refactoring […].” [KiLe00]
[215] „[...] consistent coding style assists the gatekeepers in evaluating contributed code for inclusion in the product.” [KiLe00]

Neben der konsistenten Verwendung eines etablierte Datenformats sollte ein Styleguide auch die Formatierung und Struktur eines Dokumentationsartefakts beschreiben und falls notwendig alle bereitzustellenden Metadaten eines Dokumentationsartefakts definieren. Dadurch werden die Dokumentationsartefakte homogenisiert und können besser als zentrale, Ressource integriert werden. Die Strukturierung der Inhalte der Artefakte sollte ebenfalls in diesem Dokument oder in einem entsprechenden prozessbeschreibenden Leitfaden (Documentation Development Guide) spezifiziert werden. Dies sollte die genaue Methode der Aggregation von Inhalten zu einem expliziten Dokument definieren und somit die Vorgaben für die inhaltliche Artefakterstellung definieren, um eine größtmögliche Konsistenz aller Dokumentationsartefakte zu ermöglichen und inhaltliche Redundanzen zu vermeiden.

Die zentrale Verwaltung der Dokumentationsartefakte über eine geeignete Software-Infrastruktur sollte die strukturierte Beschreibung und Vereinheitlichung der zu erstellenden Artefakte und Metadaten unterstützen.

C.1.2 Operationale Managementartefakte

Die folgenden Artefakte dienen entsprechend dem Metamodell der operativen Steuerung der Entwicklungsprozesse.

C.1.2.1 Review- und Software-Test Guideline

Etwaige Review bzw. Software-Test Guidelines können zwar der Kategorie der prozessbeschreibenden Richtlinien zugeordnet werden, werden aber an dieser Stelle explizit dargestellt, da sie nicht primär an die gesamte Projekt-Community gerichtet sind, sondern Akteure der Core Developer, respektive die Source Code Reviewer und Software Tester adressieren. Diese Managementartefakte beschreiben die expliziten Prozesse des Quellcode Review und des Tests der Software. Ein Software-Test Guide kann außerdem etwaige explizite Test Cases beinhalten, sofern diese existieren. Um die Transparenz zu verbessern und die optimale Ausführung von Review und Test Prozessen auch durch die verteilten Akteure zu unterstützen, sollten diese Artefakte zentral und für die gesamte Community transparent publiziert werden.

C.1.2.2 Status Files

Um die Transparenz über den aktuellen Entwicklungsstatus einzelner Quellcodesegmente des Entwicklungsquellcodes bzw. der realisierten Modifikationen zu gewährleisten, werden in einigen Projekten die sogenannte Status-Files verwendet (vgl. [Dietze03], [Cubranic01][216]). Da diese Artefakte aber nicht projektübergreifend generalisiert werden konnten, wurden sie nicht explizit im deskriptiven Modell berücksichtigt und werden an dieser Stelle als sinnvolle, verallgemeinerbare Erweiterung dargestellt.

Die Status-Files tragen zur Koordination individueller Entwicklungsaktivitäten bei, indem sie verschiedene, die Erfassung von operativen, prozessbezogenen Informationen und deren zentrale Bereitstellung ermöglich. Zwar wird der Status einzelner Entwicklungszyklen häufig bereits durch die einzelnen Change Requests dokumentiert, aber der gesamtheitliche Status verschiedener Quellcodeartefakte und deren historische Entwicklung kann in diesem Rahmen nicht dokumentiert werden. Daher können die Status-Files als operatives Managementwerkzeug betrachtet werden.

Zum Beispiel können Status-Files für einzelne, funktional und architektonisch voneinander abgrenzbare Softwaremodule, oder für spezifische Pfade oder Verzeichnisse des zentralen Quellcodearchivs generiert und verwendet werden. Neben der allgemeinen Beschreibung der Funktionalität bzw. des implementierten Entwurfs und der verfügbaren Schnittstellen, sollte das Status-File für jede Modifikation des Quellcodes die folgenden

[216] „For example, the Apache Project's active code repositories contain a file called ‚STATUS' which is used to keep track of the agenda and plans for work within that repository." [Cubranic01]

Informationen beinhalten, sofern derartige Informationen nicht bereits durch das Quellcoderepository archiviert werden:

- Autor
- Committer
- Datum
- Beschreibung der implementierten Verbesserungen
- Referenz auf den entsprechenden Code Change Request

Innerhalb der Status-Files können außerdem etwaige Zeitpläne für die Durchführung von Software-Releases dieses Quellcodesegments erfasst werden, um auch in diesem Kontext die Transparenz zu verbessern. Häufig konnte auch die Erfassung von ToDo-Listen im Inhalt dieses Artefakts identifiziert werden. Da Quellcodemodifikationen und damit verbundene Änderungsanforderungen aber bereits zentral und konsistent in den Change Requests definiert werden, sollte auf eine redundante Verwaltung dieser Informationen verzichtet werden.

Die Aktualisierung eines derartigen Managementartefakts sollte sinnvollerweise durch den verantwortlichen Committer durchgeführt werden, der zudem auch über die erforderlichen Informationen verfügt. Zudem sollten die Status-Files nach jeder Modifikation an alle explizit an dem betreffenden Quellcode interessierten Akteure oder die gesamte Entwicklergemeinde versendet werden.

C.2 Technologische Artefakte

Die Spezifikation standardisierter technologischer Artefakte ermöglicht einen effizienten kollaborativen Entwicklungsprozess und ein gemeinsames Verständnis zentraler Ressourcen und stellt dadurch eine wichtige Voraussetzung vor allem für große OSS-Projekte dar. Dieser Anforderung wird zwar bereits durch die Definition verschiedener Managementartefakte Rechnung getragen, welche den Entwicklungsprozess und somit auch indirekt die erstellten technologischen Artefakte beeinflussen, können aber im OSSD-Kontext vor allem durch die standardisierte Definition und softwaretechnische Unterstützung von technologischen Artefakten gewährleistet werden.

C.2.1 Patch

Im Zentrum der Softwareverbesserungsprozesse stehen die individuellen Patchentwicklungsprozesse durch die Entwicklergemeinde des Projekts, da jegliche Quellcodemodifikation in Form von Patches transportiert wird. Die im deskriptiven Modell beschriebenen Patch-Artefakte, welche mit dem Werkzeug Diff erstellt werden, stellen eine adäquate Möglichkeit dar, den zu kommunizierenden Quelltext zu minimieren, die Patches zu standardisieren und dadurch den Prozess der Integration in den Entwicklungsquellcode zu ermöglichen.

Eine sinnvolle Erweiterung eines Patches stellt der Vermerk des eindeutigen Identifikators des zugrundeliegenden Code Change Request im Header der entsprechenden Quellcode-Datei dar. Zwar sollte ein Patch stets direkt dem zugrundeliegenden Anforderungsartefakt über dessen dedizierte Metadaten zugeordnet werden, zudem stellt aber die Integration dieser Information in den Quellcode langfristig die permanente Reproduzierbarkeit dieser Beziehung in beide Richtungen sicher, indem sowohl jeder Bug Report einem Patch als auch jedes Patch einem Bug Report zugeordnet werden kann.

C.2.2 Dokumentationsartefakte

Die Dokumentationsartefakte stellen eine spezifische Kategorie der Content-Artefakte dar und beinhalten konkrete Anwender- bzw. Entwicklerdokumentationen, die sich auf ein spezifisches Software-Release oder Software-Subsystem beziehen und somit direkt mit konkreten Quellcode-Elementen assoziiert werden können.

Entsprechend der beschriebenen Prozesse zur Erstellung und Verwaltung von Dokumentationsartefakten können die folgenden Charakteristika derartiger Artefakte definiert werden.

Format
Als Format werden idealtypischerweise nur standardisierte, frei verfügbare, plattformunabhängige und weit verbreitete Formate wie ASCII-Text oder HTML verwendet (vgl. [FoBa02]). Diese Formate sind mit frei verfügbarer Software editier- und darstellbar und stellen somit eine langfristige und weitreichende Verwendbarkeit sicher. Diese Formate können unter Umständen durch das proprietäre, aber trotzdem weit verbreitete und plattformunabhängige Portable Document Format (PDF) ergänzt werden.

Quellcodedokumentation
Die Dokumentation implementationsbezogener Aspekte spezifischer Quellcodeartefakte, z.B. von Patches stellt einen Spezialfall der Dokumentation dar und sollte möglichst quellcodenah verwaltet werden. Neben den direkt im Quelltext vermerkten Informationen in Form von Kommentaren oder als Bestandteil des Datei-Headers, werden häufig ReadMe-Dateien u.ä. Files im ASCII-Text Format erstellt.

Release Update
Bezüglich der Release-Artefakte werden häufig spezielle Dokumentationsartefakte, die sogenanten News-Dateien oder Release-Updates generiert. Diese beschreiben i.d.R. die Änderungen der letzten Releases und die Spezifika des aktuellen Artefakts. Dieses Dokumentationsartefakt sollte durch den Release Manager verwaltet und erstellt werden.

C.2.3 Erweitertes Software-Release

Die Software-Releases repräsentieren die zentralen Artefakte der Deploymentphase der Softwareentwicklung. Die im deskriptiven Prozessmodell dargestellten Eigenschaften der Alpha-, Beta- und General Availability Releases (vgl. *5.2.1 Software-Release*, [Dietze03c]) unterstützen die Entwicklungsprozesse bereits vergleichsweise adäquat. Durch die Alpha- und Beta-Releases werden sie auch der Forderung nach der möglichst frühzeitigen Publikation des Entwicklungsquellcodes weitgehend gerecht. Trotzdem können diese Artefakte durch die Etablierung des bereits dargestellten, zusätzlichen Prozesses zur periodischen Veröffentlichung von Nightly-Builds, also bereits kompilierte Binärversionen des möglichst aktuellen Quellcodes, um diese Klasse der Test-Builds erweitert werden.

Nightly Builds
In Anlehnung an die Releaseprozesse des Mozilla-Projekts und die in diesem Kontext praktizierte tägliche Generierung von Test-Builds, können diese Artefakte im folgenden als Nightly Builds bezeichnet. Nightly Builds repräsentieren somit kompilierte Snapshots des zentralen Entwicklungsquellcodes, die zentral über den HTTP- oder FTP-Server des Projekts veröffentlicht werden und somit einer umfangreichen Anwendergemeinde zur Verfügung stehen. Dies ermöglicht die frühzeitige Durchführung von Software-Tests und stellt durch die periodische Kompilation des Entwicklungsquellcodes dessen Kompilierbarkeit sicher.

D Unterstützende Softwarefunktionalitäten

D.1 Dezidierte Kommunikationskanäle

Eine wichtige Voraussetzung für die Durchführung eines OSS-Projekts stellt die Bereitstellung einer möglichst geeigneten Kommunikationsinfrastruktur dar. Dies resultiert vor allem aus dem hohen Grad der räumlichen Verteilung aller involvierter Akteure und der damit verbundenen Unmöglichkeit von direkter persönlicher Kommunikation (vgl. [Cubranic01][217]). Es konnte in den verschiedenen evaluierten Projekten eine sehr heterogene Kommunikationsinfrastruktur identifiziert werden, die eine konsistente, wohladressierte und redundanzfreie Informationsverteilung zum Teil behindert (vgl. [NoSt98][218]). Daher konnte die Anforderung identifiziert werden, möglichst wenig verschiedene Kommunikationstechnologien zu instrumentalisieren, mit welchen geeignete Kommunikationskanäle für alle erdenklichen Szenarien des projektrelevanten Informationsaustauschs bereitgestellt werden.

Synchrone vs. asynchrone Kommunikationstechnologien
Primär existiert in kollaborativ und dezentral geführten Projekten ein Bedarf für synchrone und asynchrone Kommunikationsoptionen [Cubranic01][219]. Die im generalisierten Prozessmodell identifizierten, hauptsächlich genutzten Kommunikationsmethoden unterstützen i.d.R. die asynchrone Kommunikation, da dies den Anforderungen der global verteilten Softwareentwicklung in hohem Masse gerecht wird (vgl. [KiLe00][220]). Als asynchrone Kommunikationstechnologien werden momentan typischerweise Newsgroups und Mailing Listen, z.T. auch parallel, eingesetzt. Um die Konsistenz der Werkzeuge zu verbessern und somit auch die Eintrittsbarrieren für interessierte Akteure zu reduzieren, erscheint es sinnvoll, eine dieser Technologien (z.B. Mailing Listen) zur projektweiten Nutzung zu etablieren und keine redundanten Kommunikationskanäle durch andere Technologien einzuführen.

Die synchronen Kommunikationsmethoden ergänzen die asynchrone Kommunikation um die Möglichkeit zur direkten und zeitnahen Kommunikation zwischen verschiedenen Akteure, aber haben aufgrund der globalen Verteilung und der damit verbundenen Heterogenität der Community eine untergeordnete Bedeutung [Cubranic01][221]. Trotz der lediglich sekundären Bedeutung ist die Bereitstellung eines einheitlichen Werkzeugs zur synchronen Kommunikation in Echtzeit als sinnvoll zu betrachten.

Dedizierte Kommunikationskanäle
Es erscheint sinnvoll, asynchrone Kommunikationskanäle für die verschiedenen Einsatzszenarien zu definieren, wobei zwischen frei zugänglichen Kanälen und zugangsbeschränkten Kanälen unterschieden werden sollte. Die rollenspezifische Definition von Kommunikationskanälen ermöglicht die zielgenaue Informationsadressierung im Rahmen der verschiedenen Teilprozesse und vereinfacht durch die Kanalisierung des Informationsaufkommens die Informationsrecherche für die Akteure.

[217] „One of the most important characteristics of distributed software development is that developers cannot any more rely on face-to-face meetings, but have to make use of technology to allow them to communicate over distance." [Cubranic01]

[218] „[...] often there are no single one preferred communication channel in a open-source project." [NoSt98]

[219] „[...] for effective distributed collaboration, developers of such projects need a rich array of tools offering both synchronous (collaborative editors, chat, and online meetings) and asynchronous capabilities." [Cubranic01]

[220] „Asynchronous communication media such as email and newsgroups are preferred, because they do not require simultaneous availability." [KiLe00]

[221] „[...] the extremely distributed nature of even a core group of developers of OSS projects [...] pretty much precludes the usage of synchronous communication." [Cubranic01]

Alle frei zugänglichen Rollen, sollten zum Beispiel durch einen eigenen Kommunikationskanal unterstützt werden. Dies umfasst somit frei zugängliche Kanäle für die Rollen User, Contributor, Developer, Reviewer. Im Rahmen einer spezifischen Rolle, kann eine weitere Differenzierung, z.B. nach bestimmten Software-Versionen oder anderen Kriterien, wie z.B. bestimmten Entwicklungsprozessen, vorgenommen werden, um die zu erreichende Zielgruppe einzugrenzen und dadurch die adressatengerechte Informationsverteilung zu verbessern.

Für alle dedizierten Rollen, welche dem Entwicklerkern des Projekts oder den Maintainern angehören, sollten zugangsbeschränkte Kommunikationskanäle eingerichtet werden, deren Adressaten zentral durch einen entsprechend privilegierten Akteur, z.B. den Environment Maintainer verwaltet werden.

Archivierung der Kommunikationsinhalte
Um die Kommunikation redundanter Informationen oder Anfragen zu vermeiden und die Nachhaltigkeit aller Kommunikationsbeiträge zu gewährleisten, sollte das gesamte Informationsaufkommen über einen größtmöglichen Zeitraum hinweg archiviert werden. Über Volltext-Suchmechanismen kann die Recherche in historischen Inhalten ermöglicht werden und dadurch die Inhalte aller Kommunikationskanäle in Form impliziter Informationsobjekte bereitgestellt werden, wodurch die Transparenz eines Projekts gesteigert werden kann.

D.2 Gesamtheitlicheres, metadatenbasiertes Artefaktmanagement

„[...] better integration and presentation of information from the various tools in open source projects may address some of the limitations of open source software development.“ [AnHeSy03]

Allgemein läßt sich feststellen, dass Software-Infrastrukturen von OSS-Projekten eine probate Verwaltung und Bereitstellung aller Informationen, respektive der Artefakte ermöglichen müssen, um dadurch eine implizite und Koordinierungsfunktion auszuüben und weitgehend selbstregulierende Prozesse zu ermöglichen [Cubranic01][222].

Gemeinsames Repository aller Artefakte und Metadaten
Die Verwaltung aller Artefakte in einer (logisch oder physisch) gemeinsamen Datenbasis ermöglicht die gesamtheitliche Betrachtung der involvierten Artefakte und unterstützt direkt die Möglichkeit zur Verknüpfung logisch und semantisch verwandter Artefakte. Dies wird zudem durch die strukturierte, metadatenbasierte Beschreibung und Verwaltung der Artefakte ermöglicht. Durch die eindeutige Beschreibung aller Artefakte, deren möglichst redundanzfreier und konsistenter Verwaltung und einer konsequenten Aktualisierung aller Metadaten (insbesondere des *Status*-Attributs) kann die Transparenz über die verteilten Entwicklungsprozesse erheblich gesteigert werden. Dies wirkt zudem der OSSD-immanenten Tendenz zu redundanten Aktivitäten entgegen.

Ausschließlich freie und standardisierte Artefakt-Formate und Software
Die ausschließliche Verwendung standardisierter Formate gewährleistet geringe Softwaranforderungen auf Seiten der Akteure, wodurch die Eintrittsbarrieren reduziert und die Etablierung einer größtmöglichen Community unterstützt wird. Desweiteren kann die Akzeptanz der verwendeten Softwareinfrastruktur erheblich gesteigert werden, wenn auch in diesem Rahmen ausschließlich OSS eingesetzt wird (vgl. *A.7 Tendenz zum Boot Strapping*).

D.2.1 Konfigurationsmanagement (Quellcode)

Eine wichtige Funktionalität der Entwicklungsprozesse, welche durch eine probate Infrastruktur unterstützt werden sollte, ist der Aspekt der zentralen Quellcodeverwaltung.

[222] „Based on published experience reports of open-source developers, more effective support for knowledge management and coordination will be needed as projects evolve and grow with time.” [Cubranic01]

Versionierung und Konfigurationsmanagement
Elementare Funktionalitäten einer geeigneten Infrastruktur müssen eine geeignete Versionierung und ein umfassendes Konfigurationsmanagement ermöglichen, um die kollaborative Entwicklung und gemeinsame Nutzung einer zentralen Quellcodebasis zu ermöglichen. In diesem Zusammenhang ist auch eine umfassende softwaretechnische Unterstützung von Nutzerrechten notwendig. Die Verwaltung verschiedener über alle Quellcodeversionen (Konfigurationen) ermöglicht die Isolation von Softwarefehlern in spezifischen Softwareversionen und vereinfacht somit den Prozess der Fehlerbeseitigung [KiLe00][223]. In allen evaluierten Projekten werden inzwischen diesbezügliche Software-Werkzeuge verwendet (vgl. [Dietze03c]), wobei die in diesem Kontext identfizierte Methode der Verwaltung über das System CVS (vgl. *4.2.2 Kollaborationswerkzeuge* und [Dietze03]) diesen Aspekt bereits auf geeignete Weise unterstützt. Trotzdem wird in verschiedenen anderen Projekten der Quellcode lediglich über die HTTP- und FTP-Server der Projekte veröffentlicht und eine kollaborative Entwicklung nicht dediziert unterstützt wird (vgl. [Cubranic01] oder *5.5 Validierung des deskriptiven Prozessmodells*).

Hypertextbasierte Quellcodepräsentation
Weiterhin erscheint es sinnvoll, eine webbasierte Darstellung der verwalteten Quellcodeartefakte auch auf zentraler Ebene zu ermöglichen und somit die Potentiale des Hypertexts zu nutzen, indem zum Beispiel logisch miteinander assoziierbare Quellcodeelemente durch Hyperlinks miteinander verknüpft werden. Dies ermöglicht beispielsweise die Verknüpfung von Variablen mit ihren globalen Definitionen oder von Funktionsaufrufen mit der Definition der jeweiligen Funktion [LXR03][224]. Im Mozilla Projekt wird bereits die Software LXR zu diesem Zweck eingesetzt (vgl. [Dietze03] und [Dietze03c]), wobei eine Integration derartiger Funktionalitäten in eine optimierte Software-Infrastruktur als nützlich betrachtet wird, um die Transparenz über den Quellcode und damit auch über die Entwicklungsprozesse zu verbessern.

D.2.2 Gesamtheitliches Request Tracking

Wie bereits dargestellt wurde, stellt die softwaretechnische Unterstützung der Softwarefehlererfassung und -verwaltung und eine darauf aufbauende Verwaltung von Anforderungsartefakten eine wichtige Voraussetzung für die Durchführung OSSD-Projekten dar. Neben den klassischen Change Requests wurden in *7.2.2 Request-Artefakte* verschiedene weitere Anforderungsartefakte, z.B. bezüglich organisatorischer oder infrastruktureller Anforderungen, definiert, welche die klassischen Bug Reports und Enhancement Requests erweitern und ein gesamtheitlicheres Management aller Requests und Aktivitäten durch die Community ermöglichen. Deren strukturierte Verwaltung und Beschreibung basierend auf Metadaten, analog zu den klassischen Change Requests in einem dezidierten System, ermöglicht die artefaktbasierte Beschreibung und Koordination aller Aktivitäten und die zentrale Kontrolle des Fortschritts ihrer Bearbeitung. Dadurch kommt einem derartigen System zentrale Bedeutung für das Projekt Management eines verteilten OSS-Entwicklungsprojekts zu.

Unterstützung artefaktspezifischer Metadaten
Eine zentrale, softwarebasierte Verwaltung aller Anforderungsartefakte sollte es ermöglichen, alle Metadaten abzubilden, welche in *7.2.2 Request-Artefakte* bereits für die strukturierte Verwaltung aller Request-Artefakte definiert wurden.

Suchmechanismen
Aufbauend auf der metadatenbasierten Verwaltung aller Informationen in einem DBMS können den Anwendern Funktionalitäten zur attributbasierten Kategorisierung und Suche in den Artefakten zur Verfügung gestellt werden [KiLe00][225]. Dadurch könnte das Informationsmanagement in einem OSSD-Projekt und die Transparenz über alle

[223] „Another important use of configuration management tools on large projects is bug isolation.“ [KiLe00]

[224] „Quick access to function declarations, data (type) definitions and preprocessor macros makes code browsing just that tad more convenient.“ [LXR03]; URL: http://lxr.linux.no; Abfrage: 12.03.2003

[225] „Tracking systems support searching and categorization, contributing to a better development process.” [KiLe00]

laufenden Aktivitäten und Prozesse entscheidend verbessert und somit auch die Eintrittsbarrieren für interessierte Akteure minimiert werden, da der Informationsbeschaffungsprozess erheblich vereinfacht und der dafür nötige Aufwand minimiert wird.

Support Database
In klassischen OSS-Projekten werden Support-Anfragen i.d.R. nicht strukturiert in einem dezidierten System verwaltet, sondern typischerweise lediglich über einen der Kommunikationskanäle versendet oder in einem Forum kommuniziert. Dies führt dazu, dass zwar das Aufwand-Nutzen-Verhältnis für den anfragenden Akteur sehr günstig ist, aber Akteure, welche aktiv Lösungen zu verschiedenen Problemen beitragen wollen, einen hohen Arbeitsaufwand investieren müssen, um eine Anfrage zu identifizieren, welche sie bearbeiten können, da sie erst einen großen Teil des Informationsaufkommens durchsuchen müssen [LaHi00][226]. Die Archivierung aller Kommunikationskanäle und darauf aufbauende Volltextsuchmechanismen kompensieren dieses Problem zwar zum Teil, ermöglichen aber noch keine strukturierte, konsistente und redundanzfreie Verwaltung der Support Requests. Dies wird erst durch die strukturierte, metadatenbasierte Beschreibung aller Support-Requests ermöglicht, wie bereits in *7.2.2.3 Support Request* veranschaulicht wurde.

D.2.3 Verknüpfung verwandter Artefakte

Nach [BoLaNuRa03][227] stellt die fehlende Verknüpfung von inhaltlich miteinander verwandten Artefakten ein großes Manko in der OSS-Entwicklung dar, was zu einem hohen Aufwand im Rahmen der Informationsbeschaffung für verschiedene Aktivitäten führt. Informationen, welche für einen bestimmten Prozess oder eine Aktivität notwendig sind, sind häufig in verschiedenen Artefakten und verschiedenen Informationsquellen verteilt, was einen sehr hohen Recherche- und Informationsbeschaffungsaufwand für den ausführenden Akteur bedingt. Beispielsweise ist es für einen Developer, der einen Softwarefehler beheben möchte, sinnvoll, neben den Informationen des entsprechenden Code Change Requests über alle archivierten Mailing Listen-Einträge bezüglich des betreffenden Software-Artefakts und des Code Change Requests zu verfügen (vgl. [BoLaNuRa03][228], [CuHoYiMu03]). Dieser Notwendigkeit kann durch die integrierte Verwaltung aller Artefakte der verschiedenen Datenquellen und ein möglichst gesamtheitliches Artefaktmanagement in einer, logisch oder physisch integrierten, Datenbank Rechnung getragen werden. In [AnHeSy03][229] wird dazu der Ansatz eines semantischen Netzes diskutiert. Die Bildung von Ontologien als Voraussetzung zur Entwicklung eines semantischen Netzes kann in diesem Zusammenhang direkt durch die metadatenbasierte Verwaltung aller Artefakte unterstützt werden.

D.2.4 Content- und Document Management

„[…] more ambitious automated content management is evolving into a useful and accepted piece of the open source development portal." [ErHaSc03]

Dokumentationsartefakte werden in allen evaluierten Projekten meist auf unstrukturierte Weise verwaltet und bearbeitet (vgl. [Dietze03]). Zudem enthalten die Dokumentationsartefakte keine konsistenten und redundanzfreien Inhalte, sondern werden weitgehend informell erstellt und veröffentlicht. Es existieren in den verschiedenen Projekten sehr heterogene Ansätze zur Verwaltung der Dokumentationsobjekte, was zu einer hohen

[226] „[…] the Apache approach has a cost for information providers: multiple experts expend the time to read many questions that they are not able or willing to answer." [LaHi00]

[227] „Currently there are no links between the artefacts and the informal documentation describing them (mailing list archives have to be searched, for example), which results in a loss of efficiency." [BoLaNuRa03]

[228] „[…] mailing list archives are the only record of design decisions. If these archives were readily related to the components which they describe (and, perhaps more usefully, vice versa), a developer wishing to understand the design of a component could more easily discover the rationale for decisions." [BoLaNuRa03]

[229] „Semantic web technologies can directly address the need for better integration and presentation of information in open source collaboration tools." [AnHeSy03]

Heterogenität der Artefakttypen geführt hat. Diese Intransparenz wird den Anforderungen der kollaborativen Entwicklung nicht adäquat gerecht und repräsentiert einen Grund für die häufig nur ungenügende Bereitstellung aktueller Dokumentationen in OSS-Projekten. Zudem konnten keine Ansätze identifiziert werden, welche eine kollaborative Bearbeitung der Content-Artefakte analog zur Quellcodeentwicklung unterstützen. Lediglich die CVS-basierte Verwaltung textbasierter Dokumentationsartefakte wird in einigen Projekten zur Befriedigung dieser Anforderungen praktiziert (vgl. [Dietze03]). In verschiedenen OSS-Projekten wurde dieses Problem bereits erkannt und es werden verschiedene, momentan meist noch unzureichende Ansätze diskutiert bzw. erprobt, Content Management Funktionalitäten bereitzustellen (vgl. [ErHaSc03]).

Im Rahmen der evaluierten Projekte konnten zudem keine expliziten Methoden bzw. Infrastrukturen identifiziert werden, welche die Verwaltung, redaktionelle Bearbeitung und Publikation der gesamten Inhalte der Website unterstützen. In keinem Projekt konnten Infrastrukturen identifiziert werden, welche entsprechend autorisierten Akteuren die selbständige redaktionelle Pflege bestimmter Webinhalte ermöglichen bzw. diesbezüglich realisierte Workflows unterstützen (vgl. [Dietze03]).

Daher sollten zur Verwaltung und zur Unterstützung der kollaborativen, verteilten Bearbeitung der webbasierten Inhalte der zentralen Web-Plattform des Projekts und der Dokumentationsartefakte Funktionalitäten zum Web Content- und Dokumenten Management in eine Projekt-Infrastruktur integriert werden (vgl. [ErHaSc03][230]). Web Content Management Systeme (WCMS) bzw. Dokumenten Management Systeme (DMS) unterstützen z.T. ähnliche Funktionalitäten und verfolgen primär die Zielsetzung, die autonome und kollaborative Entwicklung gemeinsam genutzer Dokumente bzw. Inhalte (Assets) durch dedizierte Rechtevergabe, die Abbildung von Rollenkonzepten u.ä. zu ermöglichen. Dadurch ermöglichen sie formalisierte Entwicklungs- und Aktualisierungsprozesse aller Informationsobjekte ähnlich zu den OSSD-typischen Patchentwicklungsprozessen und repräsentieren daher eine sinnvolle Erweiterung der Infrastruktur eines OSS-Projekts [ErHaSc03][231].

Die folgenden Aspekte werden durch WCMS- bzw. DMS-Funktionalitäten i.d.R. unterstützt:

- Metadatenbasierte Verwaltung von Inhalten
- Archivierung und Versionierung der Inhalte (Assets)
- Trennung von Layout und Inhalt der Assets
- Konfigurationsmanagement der Assets
- Unterstützung paralleler Bearbeitungsprozesse (Checkin/Checkout.Mechanismen)
- Rollenbasierte und/oder assetbezogene Rechtevergabe
- Unterstützung redaktioneller Workflows

Durch die Generierbarkeit von Metadaten kann die bereits definierte Artefaktspezifikation für Dokumentationsartefakte softwaretechnisch abgebildet werden. Content- bzw. Document Management Funktionaliäten ermöglichen die dedizierte Rechtevergabe und können somit ein definiertes Rollenkonzept unterstützen. Durch die Versionierung und Archivierung der Inhalte und durch die Bereitstellung von Checkin/Checkout-Mechanismen wird die verteilte dezentrale Entwicklung an einer gemeinsam genutzten Artefaktbasis ermöglicht. Zudem sind redaktionelle Workflows durch die Integration von CMS-Funktionalitäten weitgehend automatisierbar [ErHaSc03][232]. Dies kann zudem direkt zur softwaretechnischen Unterstützung der erweiterten Review- und Qalitätssicherungsprozesse verwendet werden.

[230] „It is within the Documentation database/content area that a CMS can provide the most immediate utility to an open source portal." [ErHaSc03]

[231] „It can aid project information awareness, assist project/personal workflow, and facilitate the use and maintenance of models." [ErHaSc03]

[232] „Workflow automation streamlines processes that are difficult today (e.g., routing proposed web page changes by noncommitters to committers, notifying all developers that have recently checked-in changes to a group of code that its documentation has been updated, tracking and communicating workflow progress to project leaders)." [ErHaSc03]

D.3 Einheitliche, webbbasierte Anwenderschnittstellen

Die Webfähigkeit aller involvierten Werkzeuge setzt voraus, dass auch die zentral bereitgestellten Kollaborationswerkzeuge über eine webbasierte Schnittstelle verfügen. Eine webbasierte Schnittstelle ermöglicht die Verwendung gemeinsamer Ressourcen ohne spezifische Anforderungen an die Client-Software des Akteurs zu implizieren. Zur Minimierung der Einstiegsbarrieren für neue Akteure ist neben einer hohen Transparenz auch ein hoher Grad der Usability der verwendeten Infrastruktur notwendig. Eine benutzerfreundliche und transparente Infrastruktur reduziert die Einstiegsbarrieren für neue Akteure und ermöglicht ein weitgehend selbstregulierendes und stark parallelisiertes Prozessmodell, in dem möglichst viele Akteure weitgehend autark agieren können.

D.4 Workflow Management

Gerade im Kontext verteilter Arbeitsprozesse sind Workflow Management Funktionalitäten von großer Bedeutung für die Prozessunterstützung (vgl. [AvCiLuStVi03][233]). Die Automatisierung von Prozessen (Workflows) wird zwar direkt durch die verschiedenen, bereits dargestellten Softwarefunktionalitäten unterstützt, kann aber auch als übergreifende und explizit zu unterstützende Funktionalität betrachtet werden. Die Abbildung des gesamten Rollenkonzepts ermöglicht u.a. die anschließende Automatisierung von Workflows, wodurch direkt einige der elementaren Prozesse unterstützt und abgebildet werden können. Dies ist z.B. im Kontext der wesentlich formalisierbareren Prozesse der Core Developer von Bedeutung und betrifft z.B. die softwaretechnische Umsetzung der Reviewprozesse, indem ein Artefakt (Patch, Content) direkt an die verschiedenen Review-Instanzen weitergeleitet wird oder nach erfolgreicher Validierung des Artefakts automatisch der nächste Task (Commit bzw. Publikation) durch Weiterleitung an entsprechend privilegierte Akteure eingeleitet wird.

D.4.1 Implementierung des Rollen- und Rechtekonzepts

Eine probate Software-Infrastruktur muss in der Lage sein, das entwickelte Rollen- und Rechtekonzept zu implementieren und die beschriebenen Prozesse zu unterstützen. Dies beinhaltet zum Beispiel eine rollenbasierte Bereitstellung der Benutzerschnittstelle und die rollenbasierte Rechtevergabe für die verschiedenen Artefakte. Dies ist vor allem für die zentrale Quellcodeverwaltung notwendig (vgl. [KiLe00]), betrifft aber alle Bereiche einer Software-Infrastruktur.

Rollenspezifische Benutzerschnittstelle
Die gesamte Benutzerschnittstelle sollte basierend auf der Rollenzugehörigkeit des jeweiligen Akteurs bereitgestellt werden, wobei verschiedene allgemeine Inhalte für alle Akteure und verschieden rollenspezifische Inhalte erst im Anschluss an die Authentifizierung des Akteurs bereitgestellt werden. Somit könnte beispielsweise ein lesender Zugriff auf das Request-Tracking-System und die Quellcodeverwaltung allen Akteuren eingräumt werden, während verschiedene erweiterte Inhalte und Funktionalitäten, wie z.B. Schreibrechte im Request Tracking System, erst nach erfolgter Authentifizierung und entsprechend der Spezifika der Rolle des jeweiligen Akteurs bereitgestellt werden.

Notwendige Metadaten eines Akteurs
Der Aktivität der Authentifizierung eines Akteurs sollte ein einmaliger Registrationsprozess vorausgegangen sein, welcher zur softwaretechnischen Generierung eines neuen Nutzer-Objekts mit mindestens den folgenden Metadaten führen sollte:

- *Name:* Name des Akteurs
- *ID:* Eindeutiger Identifikator (Automatische Vergabe durch das System)

[233] „Workflow Management Systems (WfMSs) are an effective technology for improving the management of business processes, particularly by improving the integration, coordination, and communication for both human and automatic tasks of a process in a cooperative networking environment.“ [AvCiLuStVi03]

- *E-Mail:* E-Mail-Adresse des Akteurs
- *Role:* Aktuelle Rollenzugehörigkeit des Akteurs
- *Date:* Datum der Registrierung

Die zu vergebenden Rollen werden durch die dedizierte Rechtezuweisung entsprechend der im deskriptiven Prozessmodell definierten und im Abschnitt *7.1.5 Erweitertes Rollenmodell* optimierten und erweiterten Rollen implementiert.

Frei zugängliche vs. zugangsbeschränkte Rollen
Die frei zugänglichen Rollen sollten durch jeden Akteur eingenommen werden können und werden einem Akteur aufgrund seiner Aktivitäten automatisch durch das System zugewiesen. Ein nicht authentifizierter Akteur erhält automatisch den Status eines Users, während mit der initialen Registrierung dem Anwenderobjekt automatisch die Rolle Contributor zugewiesen werden sollte, die über verschiedene erweiterte Privilegien verfügt. Dies umfasst zum Beispiel minimale Schreibrechte im Request Tracking System, wodurch ein Contributor in der Lage ist, ein entsprechendes Artefakt zu generieren.

Alle Rollen aus der Gruppe der Core Developer sollten nicht automatisch durch das System, sondern nur manuell durch einen entsprechend privilegierten Akteur zugewiesen werden. Die Kriterien für die Zuweisung dieser Rollen sind nicht formal definierbar, sondern erfordern einzelfallbezogene Entscheidungen durch entsprechend qualifizierte Akteure. Primär wird diese Aufgabe durch den oder die Maintainer wahrgenommen.

Die Committer verfügen beispielsweise über Schreibrechte im Quellcodeverwaltungssystem, wobei zwischen allgemeinen Schreibrechten und partiellen Schreibrechten bezüglich dedizierter Quellcodesegmente unterschieden werden kann. Ein Release Manager verfügt über erweiterte Rechte bezüglich der Aktualisierung von Inhalten der Webplattform und bezüglich des CVS-Archivs. Im Rahmen der Release-Prozesse ist er z.B. befähigt, den CVS-Branch zu sperren, welcher für den Veröffentlichungsprozess verwendet wird und entsprechend erstellte Software-Releases auf der Website und dem projektinternen FTP-Server zu publizieren. Der Environment Maintainer sollte über administrative Rechte bezüglich der gesamten Software-Infrastruktur inkl. aller integrierten Werkzeuge verfügen und ist somit in der Lage, jegliche Anpassung der Software-Infrastruktur zu realisieren. Dies umfasst z.B. die Bereitstellung neuer Kommunikationskanäle, die Etablierung neuer Quellcodezweige (Branches) im CVS-Archiv oder die Modifikation der Metadaten bestehender Artefakt- bzw. Objekttypen. Der Maintainer eines Projekts sollte als oberste Stufe in der *Rollenhierarchie* über alle Rechte bezüglich der gesamten Infrastruktur des Projekts verfügen.

D.4.2 E-Mail-Integration

Um artefaktbezogene Workflows zu unterstützen bzw. zu automatisieren, sollte eine Software-Infrastruktur über die Möglichkeit verfügen, zumindest E-Mail-Benachrichtigungen in Zusammenhang mit den verwalteten Artefakten zu ermöglichen oder sogar ein komplexeres Workflow Management unterstützen. Dies ermöglicht es zum Beispiel, die Maintainer oder Committer bezüglich verschiedener Artefakte oder Quellcodedateien automatisch über alle Aktivitäten bezüglich dieses Quelltexts zu benachrichtigen, alle Änderungen von Anforderungsartefakten automatisch an die in dessen Bearbeitung involvierten Akteure zu kommunizieren oder auch komplexere Prozesse systemtechnisch abzubilden und zu automatisieren.

D.5 Entwicklungswerkzeuge

Neben der Verwaltung von Artefakten (Kollaborationsinfrastruktur) und der Kommunikation von Artefakten (Kommunikationsinfrastruktur) stellt die Entwicklung der Artefakte einen wesentlichen Aspekt dar, der durch eine adäquate Infrastruktur softwaretechnisch unterstützt werden muss. Im folgenden wird dabei zwischen der zentralen Entwicklung, z.B. von Release Artefakten, und der dezentralen Entwicklung von z.B. Patches oder Dokumentationsartefakten unterschieden.

D.5.1 Zentrale Entwicklung

Die zentralen Entwicklungsfunktionalitäten sind primär im Kontext der Releaseprozesse von Bedeutung.

Release Developement - Packaging
Zur Generierung von Software-Releases ist es notwendig, adäquate Softwarefunktionalitäten zur Erstellung von Softwarepaketen bereitzustellen. Diese Funktionalität ermöglicht die Bündelung von verschiedenen Quellcode- und Dokumentationsartefakten zu einer komprimierten Datei, welche als Release Artefakt im Rahmen der Publikationsprozesse veröffentlicht wird.

Kompilationsautomatisierung
Im Rahmen der periodischen Erstellung und Publikation von Test-Builds, ist der Prozess der Kompilation von elementarer Bedeutung und wird zudem für alle unterstützten Plattformen und somit sehr häufig ausgeführt. Daher erscheint es sinnvoll, die Kompilation des gesamten Entwicklungsquellcodes oder einzelner Quellcodeelemente für die verschiedenen Plattformen zu automatisieren und softwaretechnisch zu unterstützen. Dadurch kann erreicht werden, dass der Release Manager nur noch in Ausnahmefällen und bei erfolgloser Kompilation in den automatisierten Kompilationsprozess eingreifen muss. Um die Transparenz über den Kompilationsprozess zu gewährleisten, sollte durch eine entsprechende Infrastruktur das Ergebnis des Kompilationsprozesses automatisch aufbereitet und über die Kommunikationskanäle bzw. die Website des Projekts kommuniziert werden. Derartige Funktionalitäten werden beispielsweise durch die in [Dietze03] und [Dietze03c] dargestellte Software Tinderbox bereitgestellt.

Automatisierung der Software-Tests
Der bereits in *B.1 Software-Test* beschriebene dedizierte Prozess des Ausführens von Software-Testaktivitäten durch entsprechende Akteure auf Seiten der Core Developer kann auch automatisiert werden, falls die Charakteristika der Tests eine Automatisierung zulassen [FoBa02][234]. Dies kann nach [FoBa02] als Ergänzung der manuellen Software-Tests durchgeführt werden, kann aber die manuellen Tests nicht ersetzen.

D.5.2 Dezentrale Entwicklung

Die dezentral verwendeten Softwaretools zur Entwicklung von Artefakten sind für ein zentrales Softwareframework nur von peripherer Bedeutung. Daher geht dieser Abschnitt nur dann auf dezentral auf Seiten der individuellen Akteure verwendete Softwarefunktionalitäten ein, wenn diese in direkter Wechselwirkung mit der zentralen Infrastruktur stehen. Dies ist der Fall, wenn die Verwendung einer zentralen Funktionalität Anforderungen für eine bestimmte dezentrale Software, z.B. einen entsprechenden Client, definieren oder wenn sich aus einer dezentralen Softwarefunktionalität Implikationen für eine zentrale Infrastruktur ergeben. Die in diesem Abschnitt thematisierten Softwarefunktionalitäten repräsentieren somit die Bestandteile einer minimalen Entwicklungsumgebung der dezentralen Entwickler, wobei ein Webbrowser auf Seiten der Akteure als minimaler Konsens aller Akteure vorausgesetzt wird.

Die dezentrale Entwicklungssoftware umfasst, wie in [Dietze03c] dargestellt wurde, clientseitig eingesetzte Softwaretools, wie Editoren, Compiler, Debugger oder Diff zur Patch-Extraktion.

Editoren
Zur Erstellung oder Modifikation von Artefakten ist es notwendig, dass die aktiv partizipierenden Akteure über entsprechende Werkzeuge verfügen. Diese dienen z.B. der Modifikation von Quellcode- oder Dokumentationsartefakten. Da bei der Artefaktdefinition und bei der Spezifikation der zentralen Infrastruktur die Verwendung von standardisierten Formaten und Werkzeugen als elementare Bedingung betrachtet wurde, ist

[234] „Bei jeder Gelegenheit, bei der das Programm diskrete Eingaben erwartet und vorhersagbare Ausgaben produziert, kann man theoretisch automatisch testen lassen.“ [FoBa02]

lediglich ein weit verbreiteter und frei verfügbarer Editor zur Entwicklung von ASCII-Text zwingend notwendig. Dieser kann z.B. verwendet werden, um Quellcode zu modifizieren oder Dokumentationsartefakte im ASCII-Text- oder HTML-Format zu erstellen. Ergänzend können zwar noch weitere Editoren-Werkzeuge, wie z.B. Integrated Development Environments (IDE), HTML-Editoren o.ä. instrumentalisiert werden, was aber keine elementare Voraussetzung für die Mitwirkung am kollaborativen Entwicklungsprozess darstellt.

Compiler
Zur lokalen Kompilation des Quellcodes im Rahmen der Patchentwicklungsaktivitäten ist ein entsprechender Compiler eine wichtige Voraussetzung. Da für alle etablierten Programmiersprachen frei verfügbare Compiler-Werkzeuge verfügbar sind, hat dies keinen negativen Einfluss auf die Eintrittsbarrieren eines Projekts. Die Abhängigkeit zur zentralen Infrastruktur basiert lediglich auf der verwendeten Programmiersprache, wodurch die verwendbaren Compiler determiniert werden.

Patch-Extraktion (Formatierung)
Wie bereits umfassend dargestellt wurde, basiert die Kommunikation und Integration von Patches i.d.R. auf der Verwendung von dezidierten Werkzeugen zur Extraktion der Quellcodemodifikationen. Die Standardisierung des Extraktionsprozesses mit einem dedizierten Werkzeug (Diff), ermöglicht die spätere Integration dieser extrahierten Quellcodeänderungen in eine zentrale Quellcodebasis, z.B. basierend auf einem CVS-Repository, und die Rekonstruktion des gesamten Quellcodes basierend auf dem Patch und dem Ausgangsquellcode. Diese Formatierung der Patches stellt eine wichtige Bedingung dar, um eine weitreichende Verständlich- und Verwendbarkeit zu gewährleisten. Zudem bewirkt diese Formalisierung des Patchformatierungsprozesses die Minimierung des Umfangs der zu kommunizierenden Quellcodeänderungen, da durch die Extraktion lediglich der veränderten Quelltextelemente nur eine minimale Informationsmenge kommuniziert, transportiert und verwaltet werden muss.

Hypertextbasierte Quellcodepräsentation
Die hypertextbasierte Darstellung des Quelltexts erscheint sowohl auf zentraler Ebene der Quellcodeverwaltung als auch auf lokaler Ebene auf Seiten der einzelnen Entwickler als adäquates Hilfsmittel. Die Verwendung einer webbasierten Darstellungsmethodik ermöglicht die Integration von Hyperlinks zwischen logisch miteinander verknüpften Quellcodebereichen, wie z.B. Schnittstellen und deren Implementation oder Variablen und deren globaler Definition.

E Implementationsansätze

E.1 Alternative Implementationsansätze

Es existieren verschiedene Varianten, zur Implementierung des OSSD-Modells, der involvierten Rollen und Artefakte und der dargestellten Softwarefunktionalitäten. Besonders in Betracht gezogen wurden dabei die folgenden Optionen.

Integrierte Kombination infrastruktureller OSS
Als mögliche Variante ist z.B. eine Integration und Bereitstellung verschiedener OSS-Tools wie CVS, Bugzilla oder LXR (vgl. [Dietze03]) basierend auf einem zentralen Application Server wie z.B. Zope[235] in Betracht zu ziehen, wobei Zope durch die Integration verschiedener Systemerweiterungen auch die Funktionalität eines Content- und Document Managements bereitstellen kann. Diese Methode lässt eine hohe Akzeptanz bei der OSS-Community erwarten, da hierbei ausschließlich OSS verwendet, die zudem z.T. auch bereits in verschiedenen OSSD-Projekten etabliert sind. Die wesentliche Innovation in diesem Rahmen wird durch die Kombination und Integration aller relevanter Werkzeuge zu einer möglichst gesamtheitlichen Plattform erzeugt, die sich an den in *7.3 Unterstützende Software-Infrastruktur* skizzierten Anforderungen orientiert und das OSSD-Modell insbesondere aller Rollen und Artefakte möglichst gesamtheitlich unterstützt.

GENESIS und OPHELIA
Zwei weitere, bereits im wissenschaftlichen Kontext diskutierte Ansätze basieren auf der OS-basierten Prozessmanagementsoftware GENESIS[236] und der ebenfalls OSS-basierten Software OPHELIA[237] (vgl. [BoSmWe03][238]) Beide Projekte verfolgen das gemeinsame Ziel, verteilte Softwareentwicklungsprojekte durch die Entwicklung einer probaten Software-Infrastruktur zu unterstützen. Während GENESIS eine gesamtheitliche Infrastruktur repräsentiert, die auf die Verwaltung und Organisation zentraler Artefakte fokussiert, basiert OPHELIA auf der Abstraktion von Schnittstellen zur Integration verschiedenster Softwaresysteme [BoSmWe03][239] Da OPHELIA somit lediglich bestehenden Systemen eine gemeinsame Abstraktionsebene hinzufügt und nicht die gesamtheitliche Verwaltung aller Artefakte in einem einzigen Repository und die damit verbundenen Möglichkeiten zur semantischen Verknüpfung ermöglicht, wird auf die weitere Erläuterung dieses Ansatzes verzichtet und im folgenden ausschließlich auf das GENESIS-Projekt eingegangen. Zudem erfordert es die Verwendung von OPHELIA, dass diese gemeinsame Abstraktionsschicht auch clientseitig für die dezentralen Entwicklungswerkzeuge realisiert wird [BoSmWe03][240] und eine webbasierte Benutzerschnittstelle somit nicht ohne weiteres einsetzbar ist.

[235] Z Object Publishing Environment; URL: http://www.zope.org

[236] GENESIS: GENeralised Environment for ProceSs Management In Cooperative Software Engineering, URL: http://www.genesis-ist.org

[237] OPHELIA: Open Platform and Methodologies for DeveLopment Tools Integration, URL: http://www.opheliadev.org

[238] „At present, there are two complementary projects working on the development of support for collaborative software engineering: GENESIS [...] and OPHELIA." [BoSmWe03]

[239] „Its primary product is a set of core interfaces that support interoperability between a range of tool categories: project management, requirements capture, modelling and software design, code generation and bug tracking accompanied by a methodology appropriate to working in a distributed manner." [BoSmWe03]

[240] „[...] OPHELIA relies on modification of the client tools." [BoSmWe03]

E.2 GENESIS

Ein vielversprechender Ansatz zur Implementierung einer Software-Infrastruktur basiert auf der OSS-basierten Prozessmanagement-Software GENESIS[241] (vgl. [BoLaNuRa03], [BoNuRa02]), die im Rahmen eines OSS-Projekts entwickelt wird, welches durch das Information Society Technology Programm[242] der Europäischen Kommission[243] gefördert wird (vgl. [GENESIS03][244]).

Zielsetzung
Mit der Entwicklung von GENESIS wird die Bereitstellung einer Prozessmanagementumgebung für Softwareentwicklungsprozesse in verteilten Umgebungen verfolgt. GENESIS soll die Abbildung und Organisation aller softwareentwicklungsrelevanten Prozesse und Artefakte über den gesamten Projekt- bzw. Softwarelebenszyklus hinweg unterstützen [GENESIS03][245]. Die Software wird nach OSSD-Methoden entwickelt und unter einem OSD-konformen Lizenzmodell veröffentlicht werden (vgl. [GENESIS03][246]).

Mit dem GENESIS-Projekt werden nach [GENESIS03][247] aber nicht ausschließlich pragmatische technologische, sondern auch verschiedene wissenschaftliche Zielsetzungen verfolgt:

- Evaluierung der Einsatzmöglichkeiten von Workflow-Technologien in verteilten Softwareentwicklungsszenarien
- Bereitstellung von Methoden zur formalisierten Beschreibung von individuell adaptierbaren Softwareentwicklungsreferenzprozessen (Prozessmodellierungsmethode)
- Identifikation von Metriken bezüglich der Beschreibung und Evaluierung von SE-Prozessen und Artefakten

E.2.1 Architektur

Die folgende Grafik visualisiert die einzelnen Elemente der Software und ihre Beziehungen:

[241] URL: http://www.genesis-ist.org/

[242] URL: http://europa.eu.int/information_society/index_en.htm

[243] URL: http://europa.eu.int/comm/index_en.htm

[244] URL: http://www.genesis-ist.org/english/description.htm; Abfrage: 14.07.2003

[245] „GENESIS intends to develop an Open Source environment that supports the co-operation and communication between software engineers belonging to distributed development teams involved in modeling, controlling, and measuring software development and maintenance processes." [GENESIS03]; URL: http://www.genesis-ist.org/english/default.htm; Abfrage 25.06.2003

[246] URL: http://www.genesis-ist.org/english/description.htm; Abfrage: 25.06.2003

[247] URL: http://www.genesis-ist.org/english/default.htm; Abfrage 25.06.2003

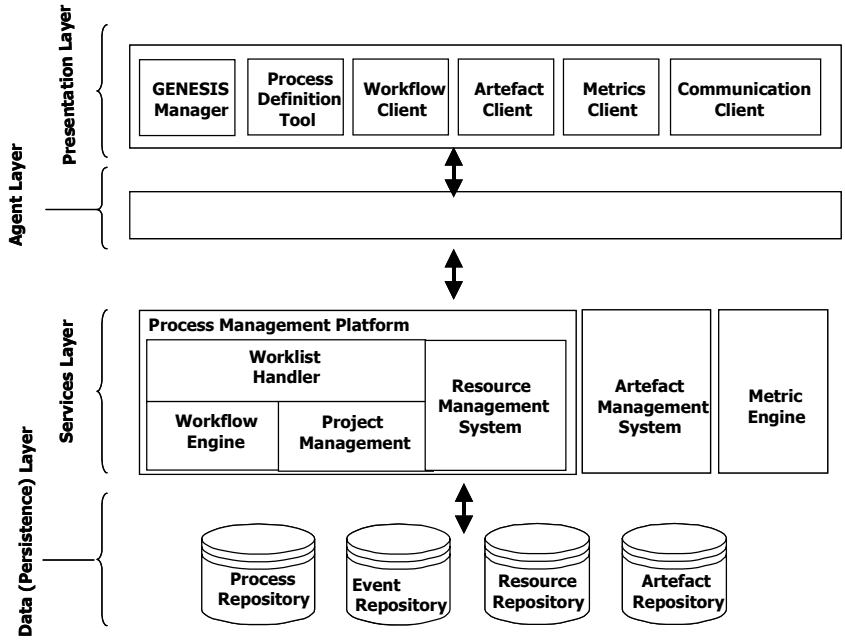

Abb. E.1: Architektur von GENESIS (aus [GENESIS03][248])

Die zentralen Bausteine der Software zur Realisierung des dargestellten Prozessmodells werden durch die Workflow Engine und das Artefakt Management System (OSCAR) repräsentiert, die durch eine Communication Engine ergänzt werden (vgl. [BoSmWe03] und [GENESIS03][248]) Über eine dezidierte Prozessmodellierungskomponente sind die Prozesse über speziell entwickelte Ausdrucksmittel strukturier- und beschreibbar und können als eigenständige Artefakte basierend auf einem dezidierten Artefakt-Typ gespeichert und instanziiert werden [BoLaNuRa03][249].

E.2.2 Gesamtheitliches Artefaktmanagement - OSCAR

Als wichtigstes Modul der Software wird in diesem Abschnitt das gesamtheitliche Artefaktmanagementsystem vorgestellt, welches zur integrierten Verwaltung aller relevanten, formalen und informellen Informationsobjekte und Daten und auch zur formalisierten Beschreibung und Verwaltung der verschiedenen Prozesse und Aktivitäten verwendet wird [BoLaNuRa03][250]. Zudem werden prinzipiell alle relevanten Informationen und Entitäten als Artefakt in OSCAR verwaltet, wodurch auch eine metadatenbasierte Verwaltung von Rollen, Akteuren, Prozessen oder Softwaretools ermöglicht wird [BoNuRa02a][251].

Das Konfigurationsmanagement der Artefakte wurde in einer eigenständigen Schicht abstrahiert, die es ermöglicht, beliebige Konfigurationsmanagementsysteme (z.B. CVS) für die Versionierung und Konfigurationsverwaltung aller Artefakttypen zu integrieren [BoSmWe03][252]. In der folgenden Grafik wurden die

[248] URL: http://www.genesis-ist.org/Doc/DL24_FINAL%20revision%20v1_1.zip; Abfrage: 30.06.2003

[249] „The GENESIS platform's process-modelling component GOSPEL allows project managers to model their organisation's existing processes." [BoLaNuRa03]

[250] „The repository is designed to store both formal software artefacts (such as code, design documents, test cases, process models, etc.) and informal material." [BoLaNuRa03]

[251] „[...] within OSCAR, everything shall appear as an artefact, even superficially different entities such as actors or software tools that OSCAR employs:" [BoNuRa02a]

[252] „Version control of artefacts is achieved through an abstraction over core configuration management system functionalities." [BoSmWe03]

wichtigsten Elemente des Artefaktverwaltungssystems und ihre Beziehungen dargestellt, die im folgenden kurz erläutert werden:

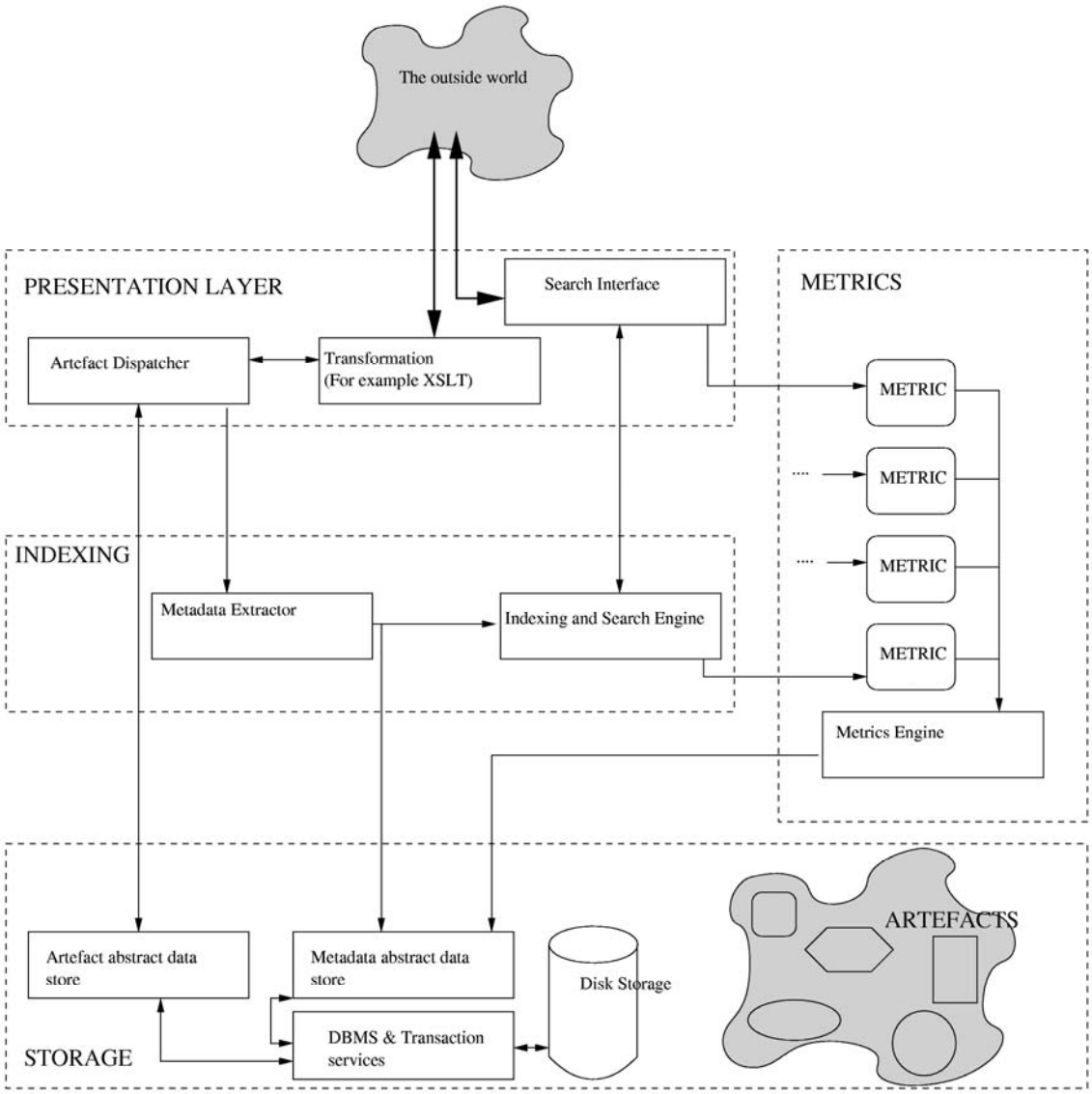

Abb. E.2: Artefaktmanagementkomponente der GENESIS Software (aus [BoNuRa02])

Webbasierte Benutzerschnittstelle

Die Präsentationsschicht des OSCAR-Systems stellt eine Schnittstelle zum Anwender des Systems bereits und kann die Inhalte z.B. webbasiert aber auch per WebDAV oder RPC zur Verfügung stellen (vgl. [BoNuRa02]). Da sich die Software noch in einem sehr frühen Prototypstatus befindet, kann die Usability der Benutzerschnittstelle noch erheblich verbessert werden, was aber durch die Schichtenarchitektur keine bedeutende Problematik darstellt.

Metadatenbasierte Artefaktverwaltung

Die Speicherung und strukturierte Beschreibung der Artefakte wird nach [BoNuRa02] durch die Definition von Metadaten in einem RDBMS unterstützt, die auch zur Verwaltung von informellen Daten, wie z.B. IRC- oder

Mailing Listen Einträge verwendet werden können. Zur Prozessabbildung und –unterstützung werden auch alle relevanten Prozesse, Rollen und Akteure im System als Artefakte verwaltet [BoSmWe03][253]. [BoNuRa02a][254] beschreibt die mit OSCAR verwalteten Artefakte als *„Active Artefacts"*, die sich dadurch auszeichnen, dass ihre Metadaten auch prozessbezogene Informationen und historische Daten über deren Modifikation beinhalten. Dieser Aspekt wird in [BoNu03] umfassend unter dem Begriff *„Historical Awareness"* diskutiert. Daher werden die Metadaten dieser Artefakte permanent analog zu dem eigentlichen Artefakt aktualisiert und ermöglichen dadurch auch die softwaretechnische Beschreibung von Artefakt-Lebenszyklen, wie sie z.B. in *7.2.2.1 Code Change Request* dargestellt wurden. Zur Prozessabbildung und –unterstützung werden auch alle relevanten Prozesse, Rollen und Akteure im System als Artefakte verwaltet [BoSmWe03][255].

Metamodell der Artefakte

Die folgende Grafik illustriert das Metamodell der Artefakte in OSCAR:

[253] „The artefact management system also holds process descriptions and personnel profiles as artefacts to assist project managers." [BoSmWe03]

[254] „'Active' artefacts are distinguished from their passive counterparts by their enriched meta-data model which reflects the work-flow process that created them, the actors responsible, the actions taken to change the artefact, and various other pieces of organisational knowledge." [BoNuRa02a]

[255] „The artefact management system also holds process descriptions and personnel profiles as artefacts to assist project managers." [BoSmWe03]

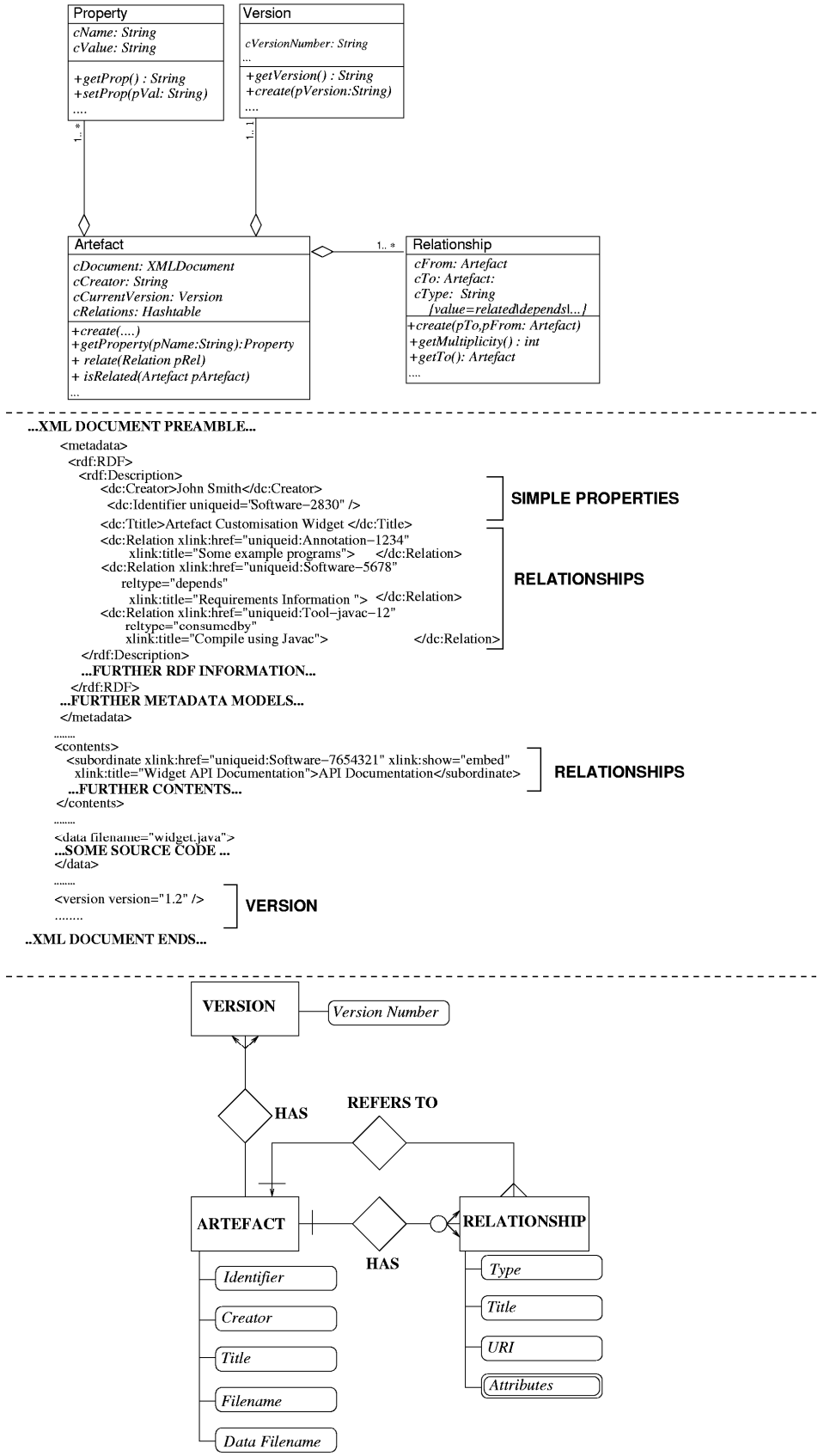

Abb. E.3: Metamodell der Artefakte in OSCAR aus [BoNuRa02a]

Artefakte verfügen somit in OSCAR über Metadaten (*Property*) und verschiedene Informationen zu jeder Version (*Version*) können erfasst werden. Beziehungen zwischen Artefakten wurden als explizite Klasse (*Relationship*) integriert und ermöglichen es, inhaltlich miteinander verwandte Artefakte zu verlinken, wodurch direkt die in *D.2 Gesamtheitlicheres, metadatenbasiertes Artefaktmanagement* definierten Anforderungen unterstützt werden [BoNuRa02][256]. Der Prozess der Verlinkung könnte zudem basierend auf den erfassten Metadaten, insbesondere der Schlüsselwörter, automatisiert werden.

E.3 Weitere integrierbare OSSD-Werkzeuge

Es existieren noch verschiedene weitere Software-Werkzeuge, welche Potentiale zur partiellen Unterstützung des dargestellten OSS-Prozessmodells bieten und daher für eine Integration in eine gesamtheitlichere Software-Infrastruktur zur Erweiterung oder Ergänzung der bereitgestellten Softwarefunktionalitäten in Betracht gezogen werden können. In diesem Kontext sollten vor allem die folgenden Tools erwähnt werden:

- LXR (Linux Cross Reference) [LXR03]
- CoxR (Code Cross Reference)
- Hipikat
- Wiki

LXR ermöglicht beispielsweise die webbasierte Navigation durch den Quellcode und unterstützt somit die Verlinkung von inhaltlich verwandten Quellcodeelementen, wie auch in [Dietze03] und [Dietze03c] dargestellt wird. CoxR ermöglicht zudem die Verknüpfung weiterer Artefakttypen mit Quellcodeelementen, indem z.B. inhaltlich relevante Mailing Listen-Einträge mit Quellcodeänderungen verknüpft werden können (vgl. [MaSaTaIsIn03]). Eine ähnliche Zielsetzung verfolgt das Werkzeug Hipikat, das momentan im Eclipse Project[257] entwickelt und eingesetzt wird und eine partielle Verknüpfung von inhaltlich verwandten Artefakten ermöglicht [CuHoYiMu03][258]. Nach [CuHoYiMu03] wurde Hipikat zur einfachen Adaption an prinzipiell jedes OSS-Entwicklungsprojekt entwickelt. Weitere Potentiale zur kollaborativen Wissensbildung und Entscheidungsfindung bieten diesogenannten Wiki-Systeme zur Unterstützung möglichst unreglementierter, projektweiter Diskussion von Problemstellungen.

[256] „Relations between artefacts (linking and dependencies) may be encoded as part of the stored metadata in the DBMS […].“ [BoNuRa02]

[257] URL: http://www.eclipse.org

[258] „For instance, Hipikat infers which source revisions correspond to which Bugzilla items, and infers similarity between Bugzilla items.“ [CuHoYiMu03]

Literaturverzeichnis

[AcFe01] Silvia T. Acuna, Xavier Ferré: Software Process Modelling; In *Proceedings der 1ª Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento*, Buenos Aires, Argentinien 2001; URL: http://www.unse.edu.ar/congres/jornadas/silvac/documentos/congres1.pdf; Abfrage: 23.06.2002.

[AnHeSy03] Anupria Ankolekar, James D. Herbsleb, Katia Sycara: Adressing Challenges to Open Source Collaboration with the Semantic Web; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; May 2003.

[ApacheOrg02] Apache Software Foundation: Website des Apache Projekts; URL: http://www.apache.org; Abfrage 2002/2003 (verschiedene Dokumente).

[ArGaLa00] Budi Arief, Christina Gacek, Tony Lawrie: Software Architectures and Open Source Software – Where can Research Leverage the Most?; Position Paper für *1st Workshop on Open Source Software Engineering at the ICSE 2001*; URL: http://opensource.ucc.ie/icse2001/ariefgaceklawrie.pdf; 2000; Abfrage: 23.06.2002.

[AtGaGeLaRo02] Chad Ata, Veronica Gasca, John Georgas, Kelvin Lam, Michele Rousseau: Open Source Software Development Processes in the Apache Software Foundation; URL: http://www.ics.uci.edu/~cata/Apache/Apache.htm; Abfrage: 03.04.2002.

[AvCiLuStVi03] Lerina Aversano, Aniello Cimitile, Andrea De Lucia, Silvio Stefanucci, Maria Luisa Villani: Workflow Management in the GENESIS Environment; Draft des Research Center on Software Technology, Department of Engineering, University of Sannio, Italien; URL: http://nash.crmpa.unisa.it/GenesisDemo/Documentation/wf_CSSE.pdf; Abfrage: 30.10.2003.

[Beck99] Kent Beck: Extreme Programming Explained: Embrace Change; Addison Wesley Longman Inc., 1999.

[Behlendorf99] Brian Behlendorf: Open Source as a Business Strategy; In *Open Sources – Voices from the Open Source Revolution*; Herausgeber: Chris DiBona, Sam Ockman, Mark Stone; O'Reilly & Associates 1999.

[BoHoBr99] Ivan T. Bowman, Richard C. Holt, and Neil V. Brewster: Linux as a Case Study: Its Extracted Software Architecture; In *Proceedings of the 21st Intl. Conf. on Software Engineering (ICSE-2001)*; May 1999.

[BoLaNuRa03] Cornelia Boldyreff, Janet Lavery, David Nutter, Stephen Rank: Open-Source Development Processes and Tools; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[BoNu03] Cornelia Boldyreff, David Nutter: Historical Awareness Support and Its Evaluation in Collaborative Software Engineering; Draft Paper, University of Durham; URL: http://www.dur.ac.uk/genesis.project/docs/wetice03/wetice03.pdf; Abfrage: 25.09.2003.

[BoNuRa02] Cornelia Boldyreff, David Nutter, Stephen Rank: Architectural Requirements for an Open Source Component and Artefact Repository system within GENESIS; In *Proceedings of the Open Source Software Development Workshop*; Februar 2002; URL: http://www.dur.ac.uk/genesis.project/docs/OSS-workshop/index.html; Abfrage 11.06.2003.

[BoNuRa02a] Cornelia Boldyreff, David Nutter, Stephen Rank: Active Artefact Management for Distributed Software Engineering; In *Proceedings of the 26th International Computer Software and Applications Conference (COMPSAC 2002)*; URL: http://www.dur.ac.uk/genesis.project/docs/COMPSAC2002/CompSac2002.ps; Abfrage 11.06.2003.

[BoSmWe03] Cornelia Boldyreff, Mike Smith, Dawid Weiss: Environments to Support Collaborative Software Engineering; Draft Paper, University of Durham; URL: http://www.dur.ac.uk/genesis.project/docs/benevento02/csmre-paper.pdf; Abfrage: 20.07.03.

[BoRuJa99] Grady Booch, Jim Rumbaugh, Ivar Jacobson: Das UML-Benutzerhandbuch; Addison-Wesley-Longman, 1999.

[Brooks95] Frederick P. Brooks: The Mythical Man-Month: Essays on Software Engineering; 1sst Edition, Addison-Wesley Pub Co, 1995.

[CaLaMo03] Andrea Capiluppi, Patricia Lago, Maurizio Morisio: Evidences in the Evolution of OS projects through Changelog Analyses; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[CaLeCh02] Bryce Carder, Baolinh Le, Zhaoqi Chen: The Process of Quality Assurance in the Mozilla Project's Release Cycle; Research Report am Institute for Software Research (University of California, Irvine); URL: http://www.ics.uci.edu/%7Eacarder/225/report.htm; Abfrage: 12.08.2002.

[CoDa94] Steve Cook, John Daniels: Designing Object Systems: Object-Oriented Modeling with Syntropy; Prentice Hall, 1994.

[Cubranic01] Davor Cubranic: Open Source Software Development; URL: http://sern.ucalgary.ca/~maurer/ICSE99WS/Submissions/Cubranic/Cubranic.html; Abfrage: 26.03.2002.

[CuHoYiMu03] Davor Cubranic, Reid Holmes, Annie T.T. Ying, Gail. C. Murphy: Tools for light-weight knowledge sharing in open-source software development; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[DeSeVo03] Jianjun Deng, Tilman Seifert, Sascha Vogel: Towards a Product Model of Open Source Software in a Commercial Environment; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[Dietze03] Stefan Dietze: Metamodellbasierte Fallstudien der Entwicklungsprozesse repräsentativer Open Source Software Projekte; Bericht am Fraunhofer ISST; Oktober 2003.

[Dietze03a] Stefan Dietze: Quality Assurance, Project Maintenance and Requirements Documentation in Open Source Software Development Projects; In *Proceedings of the Workshop 'Making Free/Open Source Software Development work better' at the 4th international Conference on eXtreme Programming and Agile Process in Software Engineering (XP 2003)*; Genua, Juni 2003.

[Dietze03b] Stefan Dietze: Metamodell zur Analyse und formalisierten Modellierung von Softwareentwicklungsprozessen; Bericht 07/03 am Institut für Informatik der Universität Potsdam.

[Dietze03c] Stefan Dietze: Formalisiertes Modell der Softwareentwicklungsprozesse im Kontext von Open Source; Bericht 08/03 am Institut für Informatik der Universität Potsdam.

[DiGa03] Jamie Dinkelacker, Pankaj K. Garg: Corporate Source: Applying Open Source Concepts to a Corporate Environment; *1st ICSE International Workshop on Open Source Software Engineering*; Mai 2001; URL: http://home.earthlink.net/~gargp/data/corpsrc01.pdf; Abfrage: 12.05.2003.

[Edwards01] Kasper Edwards: Epistemic Communities, Situated Learning and Open Source Software Development; Research Paper an der Technical University of Denmark; URL: http://opensource.mit.edu/papers/kasperedwards-ec.pdf; Abfrage: 23.11.2001.

[Eich02] Brendan Eich: Mozilla Development Roadmap; URL: http://www.mozilla.org/roadmap.html; Abfrage: 22.09.2002.

[ErHaSc03] Justin R. Erenkrantz, T.J. Halloran, William L. Scherlis: Beyond Code: Content Management and the Open Source Development Portal; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[Erenkrantz03] Justin R. Erenkrantz: Release Management within Open Source Projects; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[FeFi00] Joseph Feller, Brian Fitzgerald: A Framework Analysis of the Open Source Software Development Paradigm; In *Proceedings of the 21st International Conference in Information Systems (ICIS 2000)*; URL: http://www.josephfeller.com/publications/ICIS2000.pdf; Abfrage: 11.04.2002.

[FeFi02] Joseph Feller, Brian Fitzgerald: Understanding Open Source Software Development; Pearson Education Limited, 2002.

[FoBa02] Karl Fogel, Moshe Bar: Open Source-Projekte mit CVS; MITP-Verlag 2002.

[Fowler00] Martin Fowler: UML Distilled – Second Edition; Addison Wesley Longman; Inc. 2000.

[FreeNet03] Freenet Project: Website des Freenet Projekts; URL: http://www.freenetproject.org; Abfrage 2003 (verschiedene Dokumente).

[FSF03] Website der Free Software Foundation, URL: http://www.fsf.org; Abfrage 2002/2003 (verschiedene Dokumente).

[GaLaAr00] Christina Gacek, Tony Lawrie, Budi Arief: The many meanings of Open Source; Forschungsbericht, Department of Computing Science, University of Newcastle, 2001; URL: http://www.cs.ncl.ac.uk/research/pubs/trs/papers/737.pdf; 2000; Abfrage: 12.11.2002.

[GaGr00] Martin Gaedke, Guntram Gräf: WebComposition Process Modell: Ein Vorgehensmodell zur Entwicklung und Evolution von Web-Anwendungen; In *Tagungsband 2. Workshop Komponentenorientierte betriebliche Anwendungssysteme (WKBA 2)*, Wien 2000; URL: http://www.teco.edu/~gaedke/paper/2000-wkba2.pdf; 2000; Abfrage: 03.04.2002.

[GENESIS03] Website des GENESIS-Projekts; URL: http://www.genesis-ist.org/; Abfrage 2002/2003 (verschiedene Dokumente).

[GENESIS03a] User Manual der GENESIS-Software; URL: http://nash.crmpa.unisa.it/GenesisDemo/Documentation/GenesisUserManualV1_0.pdf; Abfrage: 30.10.2003.

[Godfrey00] Michael W. Godfrey, 2000: Toward an Understanding of Software Evolution; Vortrag an der University of Waterloo; URL: http://plg.uwaterloo.ca/~migod/papers/evolution.pdf; Abfrage: 09.11.2002.

[GoLe00] Michael Godfrey, Eric Lee: Secrets from the Monster: Extracting Mozilla's Software Architecture; In *Proceedings of the Second Intl. Symposium on Constructing Software Engineering Tools (CoSET-00)*; Limerick, Ireland, June 2000; URL: http://plg.uwaterloo.ca/~migod/papers/coset00.pdf; Abfrage: 09.11.2002.

[Gooch02] Richard Gooch: Reporting Bugs for the Linux Kernel; Projektbegleitendes Leitfaden; URL: http://www.kernel.org/pub/linux/docs/lkml/reporting-bugs.html; Abfrage: 09.09.2002.

[GoTu00] Michael W. Godfrey and Qiang Tu: Evolution in Open Source software: A case study; In *Proceedings of the International Conference on Software Maintenance (ICSM 2000)*; San Jose, California, 2000; URL: http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf; Abfrage: 09.11.2002.

[Grassmuck02] Volker Grassmuck: Freie Software; Bundeszentrale für politische Bildung, 2002.

[GrTa99] Richard Gregory, Ladan Tahvildary: Architectural Evolution: A Case Study of Apache; Research Report an der University of Waterloo; URL: http://citeseer.nj.nec.com/431508.html; Abfrage: 08.11.2002.

[HaSc01] T.J. Halloran, William J. Scherlis: High Quality and Open Source Software Practices; Position Paper, *2nd Workshop on Open Source Software Engineering at the ICSE 2002*; URL: http://opensource.ucc.ie/icse2002/HalloranScherlis.pdf; Abfrage: 07.09.2002.

[Hertel02] Guido Hertel, Sven Niedner, Stefanie Herrmann: Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel; Manuskript von Dr. Guido Hertel am 07.06.2002 zur Verfügung gestellt (vgl. URL: http://www.psychologie.uni-kiel.de/linux-study/writeup.html).

[HoRe02] Erika Horn, Thomas Reinke: Softwarearchitektur und Softwarebauelemente; Carl Hanser Verlag München Wien; 2002.

[HoSc93] Erika Horn, Wolfgang Schubert: Objektorientierte Software-Konstruktion; Carl Hanser Verlag München Wien; 1993.

[JaBoRu99] Ivar Jacobson, Grady Booch, James Rumbaugh: The Unified Software Development Process; Addison Wesley, 1999.

[Jendro03] Oliver Jendro: Web Anarchie – User-Generated Content; In *Internet World* 06/2003.

[Jordan01] John Jordan: Detecting Dominant Designs: Applying the theory in Network Economies; Master Thesis, University of Waterloo, Kanada 2001; URL: http://opensource.mit.edu/papers/jordan.pdf; Abfrage: 01.04.2002.

[KiLe00] Michael Kircher, David L. Levine: The XP of TAO – eXtreme Programming of large Open Source Frameworks; In *Proceedings of the First International Conference on eXtreme Programming and Flexible Processes in Software Engineering*, Cagliari, Italy, Juni 2000; URL: www.cs.wustl.edu/~levine/research/xp2k.ps.gz; Abfrage: 07.05.2001.

[Koch00] Stefan Koch: Entwicklung von Open Source und kommerzieller Software: Unterschiede und Gemeinsamkeiten; Vortrag, *8. Workshop der Fachgruppe 5.11 der Gesellschaft für Informatik - Leichte Vorgehensmodelle*; URL: http://wwwai.wu-wien.ac.at/~koch/forschung/sw-eng/vg01-folien.pdf; Abfrage: 12.06.2002.

[Koch01] Stefan Koch: Produktivität in der Software Entwicklung – Spezialfall Open Source: Produktivität in Open Source Projekten; Vorlesungsunterlagen am Institut für Informatik an der Universität Wien; Dezember 2001; URL: http://wwwai.wu-wien.ac.at/~koch/lehre/oss-ws-01/oss_prod/oss_prod-slides.pdf; Abfrage: 19.07.2002.

[KrKr03] Per Kroll, Phillippe Kruchten: The Rational Unified Process Made Easy; Addison Wesley, 2003.

[Kruchten00] Philippe Kruchten: The Rational Unified Process; Addison Wesley Longman, Inc. 2000.

[LaHi00] Karim Lakhani, Eric von Hippel: How Open Source software works: 'Free' user-to-user assistance; Full Paper, MIT Sloan School of Management, 2001, Cambridge - USA; URL: http://opensource.mit.edu/papers/lakhanivonhippelusersupport.pdf; Abfrage: 17.12.2001.

[LeTi02] Josh Lerner, Jean Tirole: The Simple Economics of Open Source; In *Journal of Industrial Economics*, 52 (Juni 2002); URL: http://www.people.hbs.edu/jlerner/simple.pdf ; Abfrage: 15.04.2002.

[Ludewig02] Jochen Ludewig: Modelle im Software Engineering – eine Einführung und Kritik; In *M. Glinz, G. Müller-Luschnat (Hrsg.): Modellierung 2002* Tutzing, 25.-27. März 2002; URL: http://www.iste.uni-stuttgart.de/se/publications/download/Modelle.pdf; Abfrage: 22.09.2002.

[LXR03] Website des Linux Cross Reference Project; URL: http://lxr.linux.no; Abfrage 2002/2003 (verschiedene Dokumente).

[Malotki03] Max von Malotki: Standing on the Shoulders of your Peers – Creative Commons; De:Bug – Elektronische Lebensaspekte, Ausgabe 73, Juli/August 2003.

[MaGo03] Gregorio Robles-Martínez, Jesús M. González-Barahona: Studying the evolution of libre software projects using publicly available data; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[MaGo03a] Gregorio Robles-Martínez, Jesús M. González-Barahona: Unmounting the 'code-gods' assumption; In *Proceedings of the Workshop Making Free/Open-Source Software Work Better at the XP-2003*; Mai 2003.

[MaMaAg03] L. Markus, B. Manville, C. Agres: What makes a virtual organization work ?; Sloan Management Review; Ausgabe 42.

[MaSaTaIsIn03] Makoto Matsushita, Kei Sasaki, Yasutaka Tahara, Takeshi Ishikawa, Katsuro Inoue: Integrated Open Source Software Development Activities Browser CoxR; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[Massey03] Bart Massey: Why OSS Folks Think SE Folks Are Clue-Impaired; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[MoFiHe00] Audris Mockus, Roy Fielding, James Herbsleb: A Case Study of Open Source Software Development: The Apache Server; In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*; URL: http://citeseer.nj.nec.com/mockus00case.html; Abfrage: 22.05.2002.

[MozillaOrg02] Mozilla Foundation: Website des Mozilla Projects; URL: http://www.mozilla.org; Abfrage 2002/2003 (verschiedene Dokumente).

[NiThYe02] David M. Nichols, Kirsten Thomson, Stuart A. Yeates: Usability and Open-Source Software Development; In *Proceedings of the Symposium on Computer Human Interaction*, (eds.) Kemp, E., Phillips, C., Kinshuk & Haynes, J., 49-54. Palmerston North, New Zealand. ACM SIGCHI New Zealand; URL: http://www.cs.waikato.ac.nz/~daven/docs/oss.pdf; Abfrage: 28.09.2002.

[NoSc99] John Noll, Walt Scacchi: Supporting Software Development in Virtual Enterprises; In *Journal of Digital Information*, Volume 1 Issue 4, Article No. 13, 1999-01-14; URL: http://jodi.ecs.soton.ac.uk/Articles/v01/i04/Noll/; Abfrage: 18.10.2001.

[NoSt98] Ludvig A. Norin, Fredrick Stöckl: Open-Source Software Development Methodology; Master Thesis, Lulea University of Technology, Schweden, 1998; URL: http://www.ludd.luth.se/users/no/os_meth.html; Abfrage: 12.05.2002.

[Nüttgens00] Markus Nüttgens: Open Source: Anwendungsentwicklung im Internetzeitalter; In *Gesellschaft für Informatik e.V. (GI) (Hrsg.): Fachausschuß Management der Anwendungsentwicklung und -wartung im Fachbereich 5*, Rundbrief 1/2000, 6(2000)10, S. 52-68; URL: http://www.iwi.uni-sb.de/nuettgens/Veroef/Artikel/Gi-fa_51/Gi-fa_51.pdf; Abfrage: 22.06.2002.

[ODP03] Website des Open Directory Projects; URL: http://dmoz.org; Abfrage 2003 (verschiedene Dokumente).

[OMG03] Website der Object Management Group; URL: http://www.omg.org; Abfrage 2002/2003 (verschiedene Dokumente).

[OSI02] Website der Open Source Initiative; URL: http://www.opensource.org; Abfrage 2002/2003 (verschiedene Dokumente).

[OT03] Website des Open Theory Projects; URL: http://www.opentheory.org; Abfrage 2003 (verschiedene Dokumente).

[Pavlicek00] Russel C. Pavlicek: Embracing Insanity – Open Source Software Development; Sams Publishing, September 2000.

[Pawlo03] Mikael Pawlo: Das Freenet verlässt die Staaten; In *De:Bug – Elektronische Lebensaspekte*, Ausgabe 75, Oktober 2003.

[Perens99] Bruce Perens: The Open Source Definition; In *Open Sources – Voices from the Open Source Revolution*, Herausgeber: Chris DiBona, Sam Ockman, Mark Stone, O'Reilly & Associates 1999.

[PhOSCoCom03] Website des OSSD-Projekts PhOSCo; bereitgestellt durch den Maintainer Mike Calder; URL: http://www.PhOSCo.com; Abfrage 2003 (verschiedene Dokumente).

[PhOSCoDe03] Website des Kompetenz- und Referenzzentrums PhOSCo des Koordinierungszentrum für klinische Studien (KKS) Düsseldorf; URL: http://www.PhOSCo.de; Abfrage 2003 (verschiedene Dokumente).

[PhOSCoGL03] Guillemot Design Limited: PhOSCo General Licence – General Terms and Conditions; URL: http://www.PhOSCo.com/glic4.pdf, Abfrage: 03.04.2003.

[PhOSCoOrg03] Support-und Entwicklungsplattform für die Software PhOSCo; URL: http://www.PhOSCo.org; Abfrage 2003 (verschiedene Dokumente).

[Raymond98] Eric Steven Raymond: Homesteading the Noosphere; In *First Monday – Peer reviewed Journal on the Internet*, Volume 3, Number 10 - October 1998; URL: http://www.firstmonday.dk/issues/issue3_10/raymond/; Abfrage: 20.11.2001

[Raymond01] Eric Steven Raymond: The Cathedral and the Bazar; O'Reilly UK, 2001.

[ReFo02] Christian Robottom Reis, Renata Pontin de Mattos Fortes: An Overview of the Software Engineering Process and Tools in the Mozilla Project; In *Proceedings of the Workshop on Open Source Software Development*, University of Newcastle, UK, Februar 2002; URL: http://www.dirc.org.uk/events/ossdw/OSSDW-Proceedings-Final.pdf ; Abfrage: 19.08.02.

[Rosenberg00] Donald K. Rosenberg: Open Source – The Unauthorized White Papers; IDG Books Worldwide, Inc., 2000.

[RuZuSu03] Barbara Russo, Paolo Zuliani, Giancarlo Succi: Toward an Empirical Assessment of the Benefits of Open Source Software; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[Sandred01] Jan Sandred: Managing Open Source Projects; John Wiley & Sons, Inc., 2001.

[Scacchi98] Walt Scacchi: Comparative Case Analysis for Understanding Software Processes; Research Draft, University of Southern California, submitted for publication, 1998; URL: http://www.usc.edu/dept/ATRIUM/Papers/New/CCA-Draft.html; Abfrage: 26.06.2002.

[Scacchi01a] Walt Scacchi: Understanding the Requirements for Developing Open Source; submitted *IEEE-Proceedings – Software*; Paper Number 29840; Accepted for Publication with revisions; URL: http://opensource.mit.edu/papers/Scacchi.pdf; Abfrage: 20.04.2002.

[Scacchi01b] Walt Scacchi: ITR/SOC: Understanding Open Software Communities, Processes and Practices: A Socio-Technical Perspective; Research Proposal; URL: http://www.ics.uci.edu/~wscacchi/Software-Process/Readings/NSF-Open-Software-Proposal.pdf; Abfrage: 20.08.2002.

[Scacchi01c] Walt Scacchi: Software Development Practices in Open Software Development Communities: A Comparative Case Study; Position Paper zum *1st Workshop on Open Source Software Engineering at the ICSE 2001*; URL: http://opensource.ucc.ie/icse2001/scacchi.pdf; Abfrage: 19.11.2001.

[Scacchi02] Walt Scachi: Course Materials for ICS 225: Software Process; Vorlesungsunterlagen an der School of Information & Computer Science der University of California, Irvine, USA; URL: http://www.ics.uci.edu/%7Ewscacchi/Software-Process/; Abfrage: 20.01.2002.

[Sixtus03] Mario Sixtus: WikiWiki – Edit this Page; In *De:Bug – Elektronische Lebensaspekte*, Ausgabe 72, Juni 2003.

[Sommerville96] Ian Sommerville: Software Engineering - Fifth Edition; Addison-Wesley, 1996.

[Stallman99] Richard Stallmann: The GNU Operating System and the Free Software Movement; In *Open Sources – Voices from the Open Source Revolution*; Herausgeber: Chris DiBona, Sam Ockman, Mark Stone, O'Reilly & Associates 1999.

[Stallman01] Richard Stallman: Gefahr aus Den Haag; In *Informatique/Informatik – Zeitschrift der schweizerischen Informatikorganisation*, Nr. 06/2001.

[Torvalds99] Linus Torvalds: The Linux Edge; In *Open Sources – Voices from the Open Source Revolution*, Herausgeber: Chris DiBona, Sam Ockman, Mark Stone; O'Reilly & Associates, 1999.

[TrGoLeHo00] John B. Tran, Michael W. Godfrey, Eric H.S. Lee, Richard C. Holt: Architectural Repair of Open Source Software; *The 2000 International Workshop on Program Comprehension*, 2000; URL: http://plg.uwaterloo.ca/~migod/papers/iwpc00.pdf; Abfrage: 22.07.2002.

[Tuomi01] Ilkka Tuomi: Internet, Innovation and Open Source: Actors in the Network; In *First Monday – Peer reviewed Journal on the Internet,* Volume 6, Nummer 10, 2001; URL: http://opensource.mit.edu/papers/Ilkka%20Tuomi%20-%20Actors%20in%20the%20Network.pdf; Abfrage: 07.03.2002.

[Versteegen00] Gerhard Versteegen: Projektmanagement mit dem Rational Unified Process; Springer Verlag, 2000.

[VoCo03] Michael P. Voightmann, Charles P. Coleman: Open Source Methodologies and Mission Critical Software Development; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[Vixie99] Paul Vixie: Software Engineering; In *Open Sources – Voices from the Open Source Revolution*; Herausgeber: Chris DiBona, Sam Ockman, Mark Stone; O'Reilly & Associates 1999.

[WaAb03] Juhani Warsta, Pekka Abrahamson: Is Open Software Development Essentially an Agile Method?; In *Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03*; Mai 2003.

[Waldt03] Anton Waldt: Softwarepatentierung; In *De:Bug – Elektronische Lebensaspekte*, Ausgabe 75, Oktober 2003.

[Weber92] Herbert Weber: Die Software-Krise und ihre Macher; Springer Verlag Berlin Heidelberg; 1992.

[Weber00] Steven Weber: The Political Economy of Open Source Software; Working Paper, University of California, Berkeley, 2000; URL: http://e-conomy.berkeley.edu/publications/wp/wp140.pdf , Abfrage: 17.04.2002.

[Wortmann01] Jan Wortmann: Concurrent Requirements Engineering with a UML Subset based on Component Schema Relationships; Doctoral Dissertation, Technische Universität Berlin; URL: http://edocs.tu-berlin.de/diss/2001/wortmann_jan.pdf; Abfrage: 15.08.2002.

Index

W

X