Fraunhofer Institut Software- und Systemtechnik

ISST Bericht

# Metamodellbasierte Fallstudien der Entwicklungsprozesse repräsentativer Open Source Software Projekte

## Software-Entwicklungsprozesse im Rahmen der Projekte Linux Kernel, Apache HTTPD, Mozilla

Stefan Dietze

68/03
Oktober 2003

# Inhalt


| | | |
|---|---|---|
| **1** | **Einleitung** | **1** |
| 1.1 | Darstellung der Problemdomäne Open Source | 1 |
| 1.1.1 | Open Source Software (OSS) | 1 |
| 1.1.2 | OSS-Entwicklungsprozesse | 2 |
| 1.1.3 | Evolution der OSS | 4 |
| 1.2 | Ziel und Methode | 7 |
| 1.2.1 | Zielsetzung | 7 |
| 1.2.2 | Vorgehensweise | 7 |
| 1.3 | Metamodell zur Prozessanalyse | 10 |
| 1.3.1 | Methode und Notation | 10 |
| 1.3.2 | Abgrenzung zum klassischen Systementwicklungskontext | 11 |
| 1.3.3 | Metamodell der zu betrachtenden Entitäten | 11 |
| **2** | **Fallstudie des OSS-Projekts Linux Kernel** | **21** |
| 2.1 | Linux-Kernel: Projektüberblick | 21 |
| 2.1.1 | Evolution des Kernels | 21 |
| 2.1.2 | Systemeigenschaften | 23 |
| 2.1.3 | Projektstruktur | 24 |
| 2.2 | Identifizierte Prozesse | 26 |
| 2.2.1 | Projektphasen | 26 |
| 2.2.2 | Managementprozesse | 28 |
| 2.2.3 | Entwicklungsprozesse | 32 |
| 2.2.4 | Infrastrukturelle Prozesse | 42 |
| 2.3 | Rollen | 44 |
| 2.3.1 | User | 44 |
| 2.3.2 | Contributor | 45 |
| 2.3.3 | Developer | 45 |
| 2.3.4 | Committer | 45 |
| 2.3.5 | Modul Maintainer | 46 |
| 2.3.6 | Projekt Maintainer | 46 |
| 2.4 | Artefakte | 47 |
| 2.4.1 | Managementartefakte | 47 |
| 2.4.2 | Technologische Artefakte | 49 |
| 2.5 | Werkzeuge | 55 |
| 2.5.1 | Kommunikationswerkzeuge | 56 |
| 2.5.2 | Kollaborationswerkzeuge | 59 |
| 2.5.3 | Entwicklungswerkzeuge | 62 |
| **3** | **Fallstudie des OSS-Projekts Apache** | **65** |
| 3.1 | Apache Projektüberblick | 65 |
| 3.1.1 | Teilprojekte | 66 |
| 3.1.2 | Apache HTTPD | 67 |
| 3.1.3 | Apache Jakarta | 71 |
| 3.1.4 | Apache Foundation Projekt | 74 |
| 3.1.5 | Organisation | 74 |
| 3.1.6 | Community | 77 |

3.1.7 Allgemeine Charakteristika der Prozesse 80
3.2 Entwicklungsprozesse 84
3.2.1 Bezug der Software 84
3.2.2 Erstellung eines Bug Reports 89
3.2.3 Erstellung eines Enhancement Request 90
3.2.4 Entwicklung von Patches 91
3.2.5 Integration von Quellcode 96
3.2.6 Integration von Dokumentation 100
3.2.7 Software Test basierend auf Test Releases 100
3.2.8 Support 101
3.3 Managementprozesse 102
3.3.1 Initiale Prototypentwicklung und -release 102
3.3.2 Release Planung 104
3.3.3 Release Prozess 105
3.3.4 Definition teilprojektweiter Regularien 109
3.3.5 Aktualisierung der STATUS-Files 110
3.3.6 Zuweisung von Privilegien 110
3.3.7 Generierung eines neuen Teilprojekts 111
3.4 Infrastrukturelle Prozesse 111
3.4.1 Kommunikationsinfrastrukturen 111
3.4.2 Kollaborationsinfrastrukturen 112
3.5 Gesamtheitliches Prozessmodell 113
3.6 Rollen 114
3.6.1 User 115
3.6.2 Contributor 116
3.6.3 Developer 116
3.6.4 Committers 116
3.6.5 Release Manager 117
3.6.6 Project Management Committee 118
3.6.7 Teilprojektübergreifende Rollen 118
3.7 Artefakte 119
3.7.1 Managementartefakte 120
3.7.2 Technologische Artefakte 124
3.8 Werkzeuge 130
3.8.1 Kommunikationswerkzeuge 131
3.8.2 Kollaborationswerkzeuge 134
3.8.3 Entwicklungswerkzeuge 135

**4 Fallstudie des OSS-Projekts Mozilla 137**
4.1 Mozilla-Projektüberblick 137
4.1.1 Die Mozilla-Software 138
4.1.2 Spezifische Merkmale des Mozilla-Projekts 139
4.1.3 Organisation 140
4.1.4 Softwarelizenz 144
4.2 Entwicklungsprozesse 144
4.2.1 Allgemeine Merkmale 145

| | | |
|---|---|---|
| 4.2.2 | Allgemeiner Entwicklungsprozess | 147 |
| 4.2.3 | Aktive Mitwirkung | 150 |
| 4.2.4 | Bezug der Software | 151 |
| 4.2.5 | Subscribe to Mailing List/Newsgroup | 153 |
| 4.2.6 | Fehlergetriebene Entwicklung | 153 |
| 4.2.7 | Erstellung eines Bug Reports | 158 |
| 4.2.8 | Bug Triage | 159 |
| 4.2.9 | Entwicklung von Patches | 160 |
| 4.2.10 | Quality Assurance - Testprozess als Entwicklungsprozess | 171 |
| 4.2.11 | Dokumentation beitragen | 180 |
| 4.3 | Managementprozesse | 183 |
| 4.3.1 | Initiale Prototypentwicklung- und Release | 183 |
| 4.3.2 | Prozess der "Nightly Builds" | 185 |
| 4.3.3 | Software Release | 187 |
| 4.4 | Infrastrukturelle Prozesse | 190 |
| 4.4.1 | Infrastrukturelle Anforderungen | 191 |
| 4.4.2 | Bereitstellung von Kommunikationsinfrastrukturen | 192 |
| 4.4.3 | Bereitstellung von Kollaborationsinfrastrukturen | 193 |
| 4.5 | Rollen | 194 |
| 4.5.1 | User | 194 |
| 4.5.2 | Contributor | 194 |
| 4.5.3 | Developer | 195 |
| 4.5.4 | Drivers | 198 |
| 4.5.5 | QA Owner / QA Contact | 198 |
| 4.5.6 | Nightly Build Coordinator (Sheriff) | 199 |
| 4.5.7 | Build Engineer | 199 |
| 4.5.8 | Maintenance Instanz - Mozilla Organisation | 200 |
| 4.6 | Artefakte | 200 |
| 4.6.1 | Managementartefakte | 200 |
| 4.6.2 | Technologische Artefakte | 205 |
| 4.7 | Werkzeuge | 210 |
| 4.7.1 | Kommunikationswerkzeuge | 210 |
| 4.7.2 | Kollaborationswerkzeuge | 213 |
| 4.7.3 | Entwicklungswerkzeuge | 220 |
| 4.8 | Gesamtheitliche Prozessbetrachtung | 221 |
| | **Literatur** | 223 |

# Abbildungen


| | | |
|---|---|---|
| Bild 1 | Metamodell aller relevanten Entitäten | 12 |
| Bild 2 | Prozessdekomposition | 15 |
| Bild 3 | Artefaktmengen in Anlehnung an [Draxler02] | 16 |
| Bild 4 | Linux: Evolution des Kernels (aus [GoTu00]) | 22 |
| Bild 5 | Linux: Anzahl der Entwickler pro Land (nach [Tuomi00]) | 25 |
| Bild 6 | Linux: Akteure im Linux Fehlerbeseitigungs-Prozess (nach [Tuomi00]) | 27 |
| Bild 7 | Apache: Apache Web Server Release History (aus [ApWeekCom01], 2001) | 70 |
| Bild 8 | Apache: Kumulierter Anteil an Code Änderungen (aus [MoFiHe00]) | 78 |
| Bild 9 | Apache: Kumulierter Anteil an BR-basierten Änderungen (aus [MoFiHe00]) | 79 |
| Bild 10 | Apache: Gesamtheitliches Prozessmodell (nach [AtGaGe02]) | 114 |
| Bild 11 | Mozilla: Lebenszyklus eines Bug Report (nach [CaLeCh02]) | 156 |
| Bild 12 | Mozilla: Release Schedule des Mozilla-Projekts (aus [Eich03b]) | 188 |
| Bild 13 | Mozilla: Screenshot einer Tinderbox Ausgabe (aus [ReMa02]) | 218 |
| Bild 14 | Mozilla: Gesamtüberblick Mozilla-Entwicklungsprozess (nach [CaLeCh02]) | 222 |

# Tabellen


Tabelle 1 Apache HTTPD: Plattformkompatibilität (Quelle: [Jordan01]) 68

Tabelle 2 Apache: Release History Statistiken (Quelle: [Jordan01]) 70

Tabelle 3 Apache: Board of Directors, Stand 11/2002 (Quelle: [ApacheOrg02]) 77

Tabelle 4 Mozilla: Statistik der Code-Beiträge (Quelle: [ReMa02]) 144

Tabelle 5 Mozilla: Geplante bzw. erfolgte Milestone Builds (Quelle: [Eich03b]) 189

# 1 Einleitung

Dieses Dokument thematisiert vergleichende Fallstudien, welche die strukturierte Analyse repräsentativer Software-Entwicklungsprozesse im Kontext von Open Source Software (OSS) zum Ziel haben. Diese Fallstudien wurden im Rahmen eines Promotionsvorhabens erstellt, welches durch Frau Prof. Horn an der Universität Potsdam betreut wird (vgl.: [Dietze03]) und am Fraunhofer Institut für Software- und Systemtechnik (ISST, Berlin) in Kooperation mit Herr Prof. Weber und den Mitarbeitern am ISST ausgearbeitet wird.

Im Rahmen dieser Einleitung werden die Problemdomäne Open Source, die angewandte Methodik und das den Fallstudien zugrunde liegende Metamodell vorgestellt.

## 1.1 Darstellung der Problemdomäne Open Source

In diesem Abschnitt wird der wissenschaftliche Kontext vorgestellt, in welchen diese Arbeit einzuordnen ist. Es werden die zentralen Charakteristika im Kontext von Open Source definiert und erläutert, um ein Verständnis für die Zielsetzung dieser Arbeit zu ermöglichen.

### 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]

Der Begriff "Open Source Software" wurde von der Open Source Initiative geschaffen (vgl.: [HeNiHe02], [GaLaAr01]), die damit eine Variante des "Free Software"-Begriffes der Free Software Foundation definierte, welcher eine möglichst weitreichende Akzeptanz und Verbreitung sicherstellen sollte. Die Open Source Initiative (http://opensource.org) hat zum Zweck der konkreten Begriffsdefinition und -abgrenzung die Open Source Definition entwickelt, die im folgenden vorgestellt wird und den Begriff der Open Source Software allgemein und auch für den weiteren Verlauf dieser Arbeit definiert.

Die Open Source Definition (OSD, http://opensource.org/docs/definition.php) definiert verschiedene Kriterien, welche in den Lizenzbestimmungen der jeweiligen Software enthalten sein müssen, um als OSS entsprechend der OSD betrachtet werden zu können: "To this end, the OSI created a set of criteria

that software licences must meet in order to be officially (according to the OSI) open-source (Open Source Initiative, 1999)." [Jordan01]

- Möglichkeit zur freien Verteilung
- Freier Zugang zum Quellcode
- Modifikationen und Derivate möglich
- Keine Einschränkung von Personen/Gruppen oder Einsatzbereichen

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 weitervertrieben wird.

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 ([OSOrg02][1]). Im folgenden wird der Begriff Open Source Software (OSS) 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

Die zunehmende Bedeutung der Open Source Bewegung und die steigende Relevanz und Professionalität der entwickelten Software steht einem relativ stagnierenden Fortschritt in der Modellierung und Weiterentwicklung des in diesem Kontext praktizierten Prozesses der Softwareentwicklung gegenüber.

Trotz der sehr heterogenen Struktur der Entwicklungsprozesse und der sehr 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 Software Engineering abgrenzen (vgl.: [FoBa02], [GaLaAr01], [Vixie99], [Koch00], [Scacchi02]). Zwar unterliegen die verschiedenen OSS-Entwicklungsprojekte keinen gemeinsamen Reglementierungen, aber es lassen sich verschiedene gemeinsame Eigenschaften der verschiedenen

1 http://opensource.org/licenses/index.php

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
- Individuelle, parallel ausgeführte Entwicklungsprozesse einzelner Akteure
- Unabhängiger "Peer Review"
- Dynamische Releaseprozesse
- Kein explizites Management

Eric S. Raymond assoziiert in [Raymond01a] 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: "The OSS process generally involves (or has the potential to involve) large, globally distributed communities of developers collaborating primarily through the Internet." [FeFi02]. Die parallelen Entwicklungsprozesse ermöglichen die simultane Bearbeitung verschiedener Bestandteile oder Aspekte der OSS: "These Developers tend to work in parallel, with different individuals/groups working on different aspects of the system simultaneously." [FeFi02]. 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-Entwicklungsmodell sind die unabhängigen "Peer-Review"-Prozesse, welche den Prozess der Software Entwicklung im Open Source-Kontext prägen: "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]

Die Koordination, respektive das Management der Prozesse hat nach [Nüttgens00] und [Scacchi02] im OSS-Kontext lediglich korrigierenden und unterstützenden Charakter und entbehrt jeglicher direkter Steuerungsfunktion. Trotzdem existieren in allen Projekten derartige zentrale Instanzen, welche in verschiedenen Ausprägungen die primären Entwicklungsprozesse ermöglichen, unterstützen und koordinieren: "Although all projects utilize some kind of central coordinating organization, the influence this organization on the contributions to the actual software differs." [NoSt98]

Generell ist der OSS-Entwicklungsprozess durch nur sehr marginal stattfindende Planung und eine damit einhergende sehr 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 [FeFi02] 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 zentalen Managementaufgaben ist die Veröffentlichung von Software Releases, die durch sehr dynamische und häufige Release-Zyklen gekennzeichnet sind, und i.d.R. sehr begrenzte zusätzliche Funktionalitäten oder Fehlerbehebungen zur Verfügung stellen: "OSS projects are generally characterized by rapid, incremental release schedules, in which limited extra functionality is added in each release." [FeFi02]

Die genannten Spezifika, welche als eindeutige Distinktionsmerkmale die Entwicklung proprietärer Software von dem Prozess der Entwicklung von OSS abgrenzen, verdeutlichen die Unmöglichkeit der analogen Verwendung klassischer Vorgehensmodelle und Methoden in diesem Kontext: "... 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]

### 1.1.3 Evolution der OSS

Die Evolution der OSS-Produkte in Hinblick auf die Software Architektur wird erheblich durch die Entwicklungsprozesse determiniert, welche zur Entwicklung der OSS beitragen. Daher werden in diesem Abschnitt einige Ergebnisse von Studien der Softwareevolution von OSS dargestellt.

#### Linux Kernel

[BoHoBr99] unterscheidet zwischen der konkreten und der konzeptuellen Architektur, wobei die konzeptuelle Architektur als eine imaginäre Architektur der Entwickler und die konkrete Architektur als die tatsächlich implementierte Architektur beschrieben wird. Die konzeptuelle Architektur stellt dabei den Entwurf der Software, also die geplante Architektur dar und beschreibt das Verständnis der Entwickler von der Architektur. In [GrTa99] wird die

konzeptuelle Architektur als eine imaginäre oder visualisierte Betrachtung der Architektur auf einer hohen Abstraktionsebene definiert: "It is how people visualize or, more importantly, are able to understand a system from a high-level viewpoint." Die konkrete Architektur bescheibt hingegen den tatsächlichen Zustand der Architektur beschreibt: "... the concrete architecture of a system shows the relationships that exist in the implemented system." [BoHoBr99]

[BoHoBr99] untersuchte beide Architekturen am Beispiel des Linux Kernels und kommt dabei zu dem Ergebnis, dass sich die konkrete Architektur stark von der konzeptuellen unterscheidet und mehr Abhängigkeiten aufweist: "We found that the concrete architecture of the Linux kernel has substantially more dependencies than the conceptual architecture." Ähnliche Erkenntnisse der Differenzen beider Architekturen identifiziert auch [TrGoLeHo00]. Dies begründet [BoHoBr99] durch die folgenden Aspekte:

- Zusätzliche Subsysteme in der konkreten Architektur, die noch nicht in der konzeptuellen Architektur berücksichtigt wurden
- Einige Funktionalitäten wurden entgegen der konzeptuellen Architektur in der konkreten Architektur in mehreren Subsystemen implementiert
- Verbesserte Systemperformance durch Vermeidung von Schnittstellen der konzeptuellen Architektur

Trotzdem wurde eine vergleichsweise starke Modularisierung des Quellcodes realisiert.

[Tuomi00] und [GoTu00] identifizieren ein sehr starkes, superlineares Wachstum des gesamten Linux Kernels: "… we have found that at the system level its growth has been superlinear." [GoTu00]. Bei der Unterscheidung zwischen dem Kernel-Core und peripheren Subsystemen identifiziert [GoTu00] einen erheblichen Anteil von Subsystemen, die relativ unabhängig vom eigentlichen Kernel-Core existieren: "… more than half of the code consists of device drivers, which are relatively independent of each other; … and we found that the small, core kernel subsystems comprise only a small part of the full source tree.". Dies ermöglichte die weitere Erkenntnis, dass lediglich die Subsysteme, respektive das der Softwaretreiber, für das superlineare Wachstum verantwortlich sind und der Kernel-Core nur einem linearen und stetigen Wachstum unterliegt.

Weiterhin betrachtet [Godfrey00b] separat das Wachstum des Quellcodes des stabilen und des Entwicklungspfads und identifiziert ebenfalls gravierende Unterschiede. Während der Entwicklungsquellcode einem exponentiellen Wachstum unterliegt, steigt der Umfang des Quellcodes des stabilen Pfads vergleichsweise linear und stetig an. Als Indikator für die Komplexität der Software betrachtet [Godfrey00b] die Anzahl der globalen Variablen, Makros,

Funktionen und Abhängigkeiten, für die er ein Wachstum parallel zu dem Umfang des Quellcodes identifiziert.

### Apache HTTPD

[GrTa99] analysiert die Softwareevolution der OSS im Rahmen des Apache HTTPD-Projekts über die Versionen 1.1.1 bis 1.3.4 und kommt dabei zu der Erkenntnis, das sich die ursprüngliche konzeptuelle Architektur relativ konstant entwickelt hat, was er darin begründet sieht, dass sich auch die zugrundeliegende Funktionaliät nicht grundlegend geändert hat: "Based on analysis of the conceptual architecture, we realized that this remained consistent throughout each version." [GrTa99]. Die konkrete Systemarchitektur des HTTPD-Servers hat sich nur unwesentlich verändert, indem zusätzliche Module und Subsysteme hinzugefügt wurden: "... more modules were added and additional subsystems developed within the system core." [GrTa99]. Die konkrete Architektur führte dabei trotz immensen Wachstums des Umfangs des Quellcodes nicht zu einer Komplexitätssteigerung oder Verschlechterung der Wartbarkeit. So blieb zum Beispiel die ursprüngliche, modulare Struktur unverändert und auch die Verwendung der API wurde sogar im Laufe der Softwareevolution verbessert: "Over the course of evolution, the structure of the modules remained unchanged, although the number of modules increased…. It appears that, unlike other systems such Linux, the use of the API has not only remained consistant, but has improved, over the course of system evolution." [GrTa99]

### Mozilla

[ReMa02] beschreibt den Umfang der Quellcodebasis des Mozilla-Browsers mit zwei Millionen Lines of Code und ermöglicht dadurch eine Vergleichbarkeit mit dem Quellcode des Linux Kernels. [GoLe02] analysiert die Architektur des Mozilla-Browsers und kommt zu dem Fazit, dass die Architektur, respektive der Aspekt der Kohärenz, als vergleichsweise schlecht einzustufen ist: "Overall, we found the architectural coherence of Mozilla to be significantly worse than that of other large open source systems whose software architectures we have examined in detail (Linux and VIM) ... . ". Als Indikatoren dafür wurden die verschiedenen Abhängkeiten der elf identifizierten 'Top-Level-Subsysteme' betrachtet, wobei eine Vielzahl unerwarteter Abhängigkeiten eruiert wurden. Dies ermöglicht aber nicht zwingend Rückschlüsse auf den OSS-Prozess, da dies auch durch die ursprüngliche Architektur und Qualität des veröffentlichten Prototyps bedingt sein kann: "... its architecture has decayed significantly in its relatively short lifetime, or it was not carefully architected in the first place." [GoLe02]. Dies ist vor allem mit den noch dargestelleten spezifischen Umständen der initialen Prototypveröffentichung zu begründen.

**Sonstige OSS-Projekte**

[Godfrey00b] identifiziert für andere Projekte, respektive X-Windows, GCC und VIM ein annäherndes lineares Wachstum des Umfangs des Quellcodes. Weitere Studien der Evolution verschiedener OSS-Projekte dokumentieren vergleichbare Resultate (vgl.: [CaLaMo03], [MaBa03a], [MaBa03b]).

OSS unterliegt, wie bereits dargestellt wurde, einer stetigen Evolution und wird daher stark durch die zugrundeliegenden Entwicklungsprozesse determiniert. Totz eines linearen oder zum Teil noch rasanteren Wachstums konnte in vielen OSS die vergleichsweise konsequente Realisierung von klassischen Kriterien einer hochwertigen Softwarequalität identifiziert werden. Dies beinhaltet z.B. die weitgehende Modularisierung der Architektur bei einer vergleichsweise geringen Kopplung und starker Kohärenz und lesbaren, respektive strukturierten und kommentierten Quellcode.

## 1.2 Ziel und Methode

Nachfolgend werden die Zielsetzung unter der diese Fallstudien durchgeführt wurden und die gewählte Vorgehensweise dargestellt.

### 1.2.1 Zielsetzung

Diese Projektanalysen werden im Rahmen einer vergleichenden Fallstudie angefertigt, welche das Ziel verfolgt, ein möglichst fundiertes beschreibendes Prozessmodell der Software-Entwicklung im Kontext von Open Source erstellen zu können (vgl. [Dietze03]).

Diese Prozessstudien müssen daher möglichst fundiert erfolgen und auf einer gemeinsamen, konsistenten Struktur basieren, um die spätere Vergleichbarkeit im Rahmen der Generalisierung eines allgemeinen Prozessmodells zu gewährleisten.

### 1.2.2 Vorgehensweise

"… 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 primären Zielstellung der Arbeit in [Dietze03], respektive der Erstellung eines deskriptiven Prozessmodells der Systementwicklung im Open

Source Kontext, erscheint die Durchführung von vergleichenden Fallstudien als probate Methode. Diese projektbezogenen Studien können somit zur Identifikation von generalisierbaren Strukturen und Prozessen verwendet werden, die anschließend in einem verallgemeinerten Prozessmodell formal definiert werden: "… comparative analysis helps illuminate the behavioral and interactional dynamics of as-is software processes occurring in complex organizational settings." [Scacchi98]. Vergleichende Fallstudien repräsentieren nach [Scacchi98] ein geeignetes Instrument, um generalisierbare und beweisbare Aussagen über real existierene Strukturen zu treffen: "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."

#### 1.2.2.1 Metamodell

Die Prozessanalysen basieren auf einem gemeinsamen Metamodell, welches in Kapitel 1.3 "Metamodell zur Prozessanalyse" vorgestellt wird (vgl. auch [Dietze03]). Es wurde für die Analyse und Spezifikation aller drei Projekte im Rahmen der vergleichenden Fallstudie entwickelt. Dieses Metamodell definiert alle zu betrachtenden Aspekte und weist diesen bereits einige Kategorien in Form von spezialisierten Metaklassen zu, welche die Strukturierung der Analyse ermöglicht. Dadurch wird die Struktur der folgenden Kapitel und der analysierten Aspekte maßgeblich determiniert und eine Kategorisierung der identifizierten Prozesse ermöglicht. Die Dekomposition der Prozesse in Teilprozesse und (atomare) Aktivitäten und deren Definitionen, welche im Metamodell spezifiziert wurden, werden im Rahmen dieser vergleichenden Fallstudie noch nicht berücksichtigt, sondern erst im Kontext des generalisierten Prozessmodells modelliert (vgl. [Dietze03]).

#### 1.2.2.2 Projektauswahl

Die Basis für die Modellierung des Prozessmodells im Open Source Kontext bilden vergleichende Fallstudien, die anhand konkreter OSS-Entwicklungsprojekte durchgeführt werden. Dies setzt die Auswahl einiger Projekte voraus, welche als möglichst repräsentativ für die gesamte Klasse der OSS-Projekte betrachtet werden können. Hierzu wurden die folgenden Anforderungen definiert:

- Lizenzmodell kompatibel zur OSD
- Fortgeschrittener Projektstatus
- Evolutionär ausgeprägte Prozesse und Projektstrukturen
- Weitreichende Akzeptanz der entwickelten OSS

- OSS aus verschiedenen Anwendungsdomänen
- Mindestens eine OSS aus ehemals proprietärer Software-Entwicklung

Die Wahl der Softwarelizenz determiniert in hohem Maße die Rahmenbedingungen für die Evolution der Entwicklungsprozesse. Nur die Verwendung eines Lizenzmodells, welches mit der Open Source Definition 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 erleichtern. Weiterhin sollten in der ausgewählten Projektgruppe möglichst verschiedene Softwaretypen aus unterschiedlichen Anwendungsdomänen repräsentiert sein.

Aufgrund dieser Kriterien wurden die Entwicklungsprojekte ausgewählt, welche die Entwicklung der folgenden OSS zum Ziel haben:

- Linux Kernel (http://www.kernel.org)
- Apache HTTPD und Apache Jakarta (http://apache.org, http:// httpd.apache.org, http:// jakarta.apache.org)
- Mozilla (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 derartige OSS-Projekte betrachtet werden können.

Zudem sollte mindestens ein Projekt in die Analyse einbezogen werden, welches ursprünglich in einem kommerziellen Kontext initiiert wurde, da dies eine häufig auftretende Evolution von OSS-Projekten darstellt (vgl. [Koch00]). Dieser Anforderung wurde durch die Auswahl des Mozilla-Projekts Rechnung getragen.

#### 1.2.2.3 Durchführung der Fallstudien

Die Informationsgewinnung im Rahmen der Auswertung dieser Projekte basiert auf einer Symbiose aus einer empirischen Analyse der Projekte und der

Auswertung von bereits existierenden Fallstudien, welche spezifische Aspekte des Projekts betrachten aber bisher noch keine gesamtheitliche Analyse der Entwicklungsprozesse zur Verfügung stellen. 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, bewirken die Offenheit der Open Source Communities und der freie Zutritt zu jeder Entwicklergemeinde, dass die Partizipation und das umfassende Studium von Open Source Projekten ermöglicht wird. Alle Koordinations- und Kooperationswerkzeuge und -mechanismen sind frei zugänglich und auch die zentralen Ressourcen sind für jeden interessierten Akteur nutzbar und einsichtig.

Diese Aktivitäten werden durch die Auswertung von sekundärer Literatur, wissenschaftlichen Arbeiten und existierenden Fallstudien ergänzt. Dadurch entstehen primär qualitative aber z.T. auch quantitative Informationen über die verschiedenen Projekte, welche zur Konstruktion einer 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, die als probates Mittel für die Ableitung von generalisierbaren Prozessen betrachtet werden können, sofern sie im Rahmen einer vergleichenden Studie über mehrere Projekte hinweg erhoben werden: "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. But individual qualitative studies of a software process instance are not reliably generalized into a pattern, class, or process model. … 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]

## 1.3 Metamodell zur Prozessanalyse

Dieser Abschnitt spezifiziert und beschreibt das Metamodell, auf dessen Basis die Fallstudien durchgeführt und die identifizierten Aspekte definiert wurden.

Es definiert alle in den OSS Prozess involvierten und zu analysierenden Aspekte, wie z.B. Prozesse, Rollen und Artefakte und 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 (vgl. [Dietze03]).

### 1.3.1 Methode und Notation

Analog zum Metamodell der UML wurde auch dieses Modell basierend auf Bestandteilen der UML-Spezifikation entwickelt und in Klassendiagrammen der UML modelliert. Von besonderer Bedeutung ist hierbei die Tatsache, dass anhand der folgenden Metamodelle sowohl die zu analysierenden Aspekte der zu betrachtenden Prozesse modelliert werden, als auch deren Repräsentationsmöglichkeiten durch ausgewählte Modellelemente der UML definiert werden. Zur probaten und erschöpfenden Modellierung aller Aspekte der darzustellenden Prozesse ist es notwendig, die formalen Modellierungsmethoden des Software Engineering geringfügig zu modifizieren und zu erweitern: "Formal representation of software process data and relations does not readily conform to procedural process programming or conventional object-oriented modeling paradigms." [Scacchi98]. Daher werden im Rahmen der Arbeit [Dietze03], die auf diesem Metamodell aufbaut, verschiedene Sichten aus dem Metamodell der UML adaptiert und durch weitere, neu definierte ergänzt.

### 1.3.2 Abgrenzung zum klassischen Systementwicklungskontext

Das im folgenden dargestellte Metamodell ist mit einigen Einschränkungen auch auf die Analyse und Modellierung von klassischen Entwicklungsprozessen anwendbar, wobei es aber auf die Spezifikation von Systementwicklungsprozessen im Open Source Kontext hin optimiert wurde.

So werden bereits verschiedene Kategorien aufgrund einer vorausgegangenen Prozessanalyse gebildet und in Form von Metaklassen in das Modell integriert. Diese Kategorien werden als abgeleitet Klassen der Meta-Klassen Prozess, Artefakt und Rolle definiert und dienen der Einteilung dieser Aspekte bzw. der besseren Strukturierung der auf dem Metamodell basierenden Modelle. Diese verschiedenen Metaklassen werden bereits auf der Ebene dieses Metamodells näher beschrieben, um in den anschließend im Rahmen von [Dietze03] zu erstellenden Modellen als Prämisse verwendet werden zu können und unterscheiden sich bezüglich ihrer Abgrenzungskriterien z.T. von den Kategorien, die in einem Metamodell im klassischen Systementwicklungskontext sinnvollerweise gebildet werden, wie aus den folgenden Abschnitten hervorgeht.

### 1.3.3 Metamodell der zu betrachtenden Entitäten

"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." [Scacchi01]

Die isolierte Betrachtung einzelner Aspekte, respektive der soziologischen oder der technologischen Aspekte, könnte nach [Scacchi01] im Rahmen der Ableitung von verallgemeinerbaren Aussagen zu fehlerhaften Rückschlüssen und Ergebnissen führen.

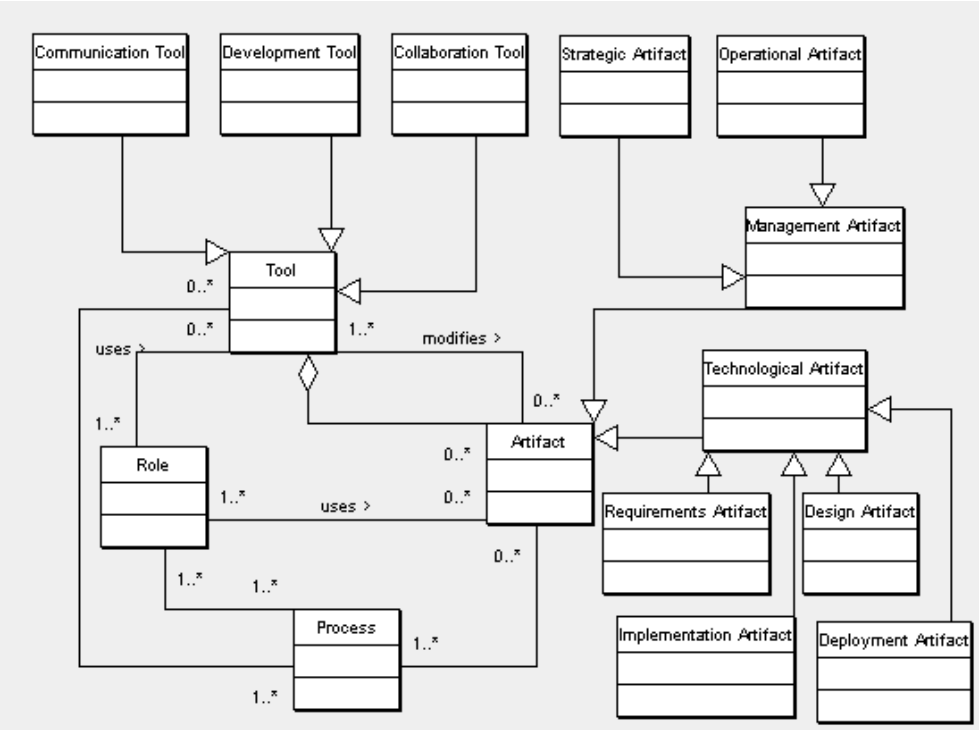
Die folgenden grundlegenden Entitäten und ihre diversen Beziehungen sollen im Rahmen der verschiedenen Fallstudien dargestellt werden:

- Rolle
- Artefakt
- Werkzeug
- Prozess

Die Dekomposition der Prozesse und ihre Integration in dieses Metamodell wird in einem expliziten Abschnitt dargestellt, um die Komplexität des Metamodells nicht zu überlasten.

Die folgende Grafik visualisiert das Metamodell der zu analysierenden und zu modellierenden Entitäten und berücksichtigt noch nicht die zu verwendende UML-Notation zur Modellierung der einzelnen dargestellten Aspekte:

Bild 1 Metamodell aller relevanten Entitäten




Das hier dargestellte Metamodell orientiert sich in den Grundzügen an dem Metamodell der klassischen Softwareentwicklung in [AcFe02] und wurde um verschiedene allgemeine und OS-spezifische Elemente erweitert. Im folgenden werden die zu analysierenden Aspekte explizit beschrieben und detailliert dargestellt.

### 1.3.3.1 Prozess

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 Bild 1 dargestellt wurde. Sie lassen sich in weitere Prozesse zerlegen, diese Dekomposition der Prozesse wird am Ende dieses Abschnitts definiert.

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. Alle elementaren Prozesse, welche direkt zur Weiterentwicklung der OSS bzw. peripherer Artefakte beitragen, werden dieser Kategorie zugeordnet. Zudem zeichnen sich die 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 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 Metaklassen geführt hat:

- Requirements Process
- Design Process
- Implementation Process
- Deployment Process

### Managementprozesse (Suprastructure)

Diese Klasse der Prozesse beinhaltet alle relevanten Prozesse zur Organisation und Koordination der Entwicklungsprozesse und ist nicht phasenspezifisch sondern -übergreifend wirksam. Diese Prozesse werden 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 werden sowohl die grundlegenden Voraussetzungen für die Entwicklung der OSS geschaffen, als auch die involvierten Entwicklungsprozesse koordiniert.

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. Naturgemäß werden diese Projekte hauptsächlich von einer zentralen Instanz zum Projekt Management im Sinne von Open Source ausgeführt.

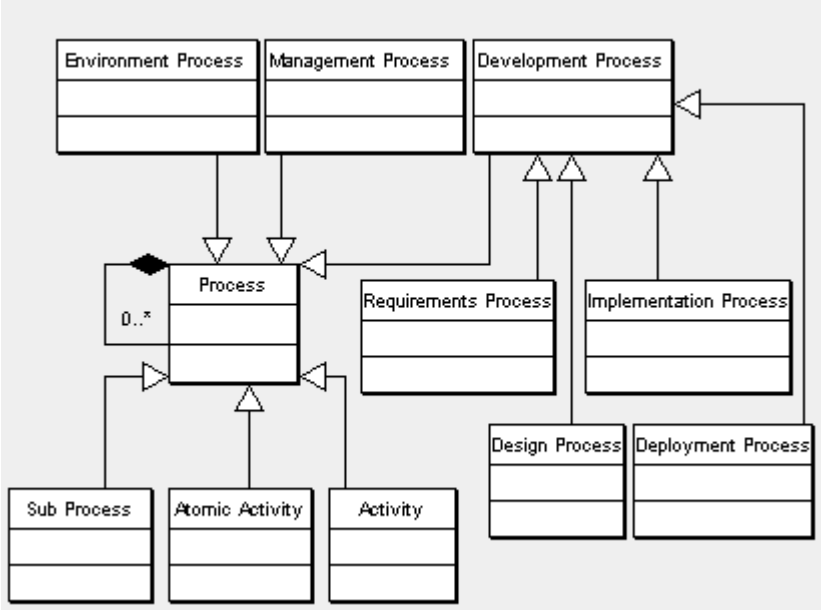
### Environment Prozesse (Infrastructure)

Die infrastrukturellen Prozesse werden auch Environment Prozesse genannte 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.

### Prozess-Dekomposition

Das im folgenden dargestellte Klassendiagramm repräsentiert die Dekomposition des Prozessmodells in Prozesse, Teilprozesse, Aktivitäten und atomare Aktivitäten in Abhängigkeit vom Abstraktionsniveau. Desweiteren werden die beschriebenen Spezialisierungen eines Prozesses in Infrastrukturelle, Entwicklungs- und Management Prozesse visualisiert.

Bild 2 Prozessdekomposition




Ein Prozess definiert somit die oberste Betrachtungsebene und wird auf einem detaillierteren Betrachtungsniveau in Teilprozesse unterteilt, welche wiederum aus Aktivitäten und atomaren Aktivitäten bestehen können. Die genaue Differenzierung zwischen diesen Prozesselementen ist für die Durchführung der

Fallstudien noch nicht von Bedeutung und findet erst im Rahmen der Modellierung eines generalsierten Prozessmodells in [Dietze03] Anwendung.

#### 1.3.3.2 Rolle

Eine Rolle beschreibt nach [Fähnrich02] die Aggregation der "notwendigen Erfahrungen, Kenntnisse und Fähigkeiten, über die ein Mitarbeiter verfügen muss, um eine bestimmte Aktivität durchzuführen." Eine konkretere Definition des Rollenbegriffs enthält [AcFe02]: "Role: describes 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 Metaklasse aggregiert werden können.

#### 1.3.3.3 Artefakt

"Ein Artefakt ist ein Dokument, Bericht oder ausführbares Modul, das erstellt, verändert oder verbraucht wird." [Draxler02]
Artefakte werden im Rahmen von spezifischen Prozessen generiert, modifiziert oder verwendet: "An artifact or software product is developed and maintained in a process." [AcFe02]. Die Metaklasse 'Artifact' repräsentiert alle identifizierten Informationsobjekte, welche für Prozesse relevant sind. Damit stellen von dieser Klasse abgeleitete Unterklassen die Input- und Outputobjekte von Aktivitäten dar: "Einige Artefakte dienen direkt als Eingabe für folgende Aktivitäten, werden als Beleg für das Projekt aufbewahrt oder in einem vertraglich festgelegten Format erzeugt." [Draxler02]

Artefakte sind im Kontext dieses Metamodells lediglich eine bestimmte Klasse von Informationsobjekten und nicht die Medien, die als Container für diese Inhalte dienen. Zum Beispiel ist es bei dem Prozess des Berichtens von Softwarefehlern, in dessen Verlauf das Artefakt 'Bug Report' generiert wird, für die Spezifikation des Artefakts unerheblich, ob dieses Artefakt in Form einer Nachricht einer Mailing Liste oder in Form eines Eintrages in einem Bug Tracking System übermittelt wird, sofern in beiden Formaten die gleichen Metadaten enthalten sind. Von Relevanz für das Artefakt sind in diesem Zusammenhang lediglich die Restriktionen, welche das verwendete Werkzeug an das Format des Artefakts stellen.

Weiterhin sind nach [Draxler02] die evoluierenden Artefakte im Rahmen eines iterativen Software Entwicklungsprozesses in die folgenden Teilmengen unterteilbar, an welchen sich das vorgestellte Metamodell orientiert:

Bild 3 Artefaktmengen in Anlehnung an [Draxler02]

| Technological Artifacts | | | |
|---|---|---|---|
| **Requirements Set** | **Design Set** | **Implementation Set** | **Deployment Set** |
| ▪ Visions Document<br>▪ Requirements Models | ▪ Design model(s)<br>▪ Test model<br>▪ Software architecture description | ▪ Source Code baselines<br>▪ Associated compile-time files<br>▪ Component Executables | ▪ Integrated product executable<br>▪ Associated run-time files<br>▪ User manual |

| Management Artifacts | |
|---|---|
| **Planning Artifacts** | **Operational Artifacts** |
| ▪ Business Case<br>▪ Release Specification<br>▪ Software development plan | ▪ Release Description<br>▪ Status Assessment<br>▪ Deployment plan |

Diese hier dargestellten Teilmengen repräsentieren wieder Metaklassen, welche jeweils spezifische Eigenschaften aufweisen und mit einzelnen Modifikationen in das spezifizierte Metamodell integriert wurden: "Jede Menge besteht aus verwandten persistenten Artefakten [...]. Während eine Menge den vollständigen Aspekt des Systems repräsentiert, repräsentiert ein Artefakt eine zusammenhängende Information, die normalerweise als eine einzelne Entität entwickelt und geprüft wird."[Draxler02].

Die technologischen Artefakte bezüglich der Anforderungen, des Designs, der Implementierung und des Einsatzes sind phasenbezogen und können direkt einzelnen Aktivitäten zugeordnet werden. Demgegenüber stehen die größtenteils phasenübergreifend wirksamen Management-Artefakte.

Managementartefakte: 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 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 in [Draxler02] definierten Managementartefakte betreffen hierbei hauptsächlich finanzielle Aspekte, langfristig wirksame Entwicklungspläne oder

Projekt-Management Dokumente, welche in der OSS-Entwicklung nicht oder nur marginal thematisiert werden. Daher werden die Management Artefakte im Kontext dieses Metamodells in die folgenden Teilmengen unterteilt:

- Strategische Artefakte
- Operationale Artefakte

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

Die operationalen Artefakte orientieren sich dabei z.T. an [Draxler02] und 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 bzw. Entwicklungsartefakte:**

"In einer systematischen Software-Entwicklung nach dem Wasserfall-Modell entstehen nacheinander Dokumente, die Modell-Eigenschaften aufweisen..." [Ludewig02]
Diese Eigenschaften repräsentieren den Umstand, dass ein Artefakt dieser Kategorie stets als deskriptives bzw. präskriptives Modell betrachtet werden kann. Diese Menge der Artefakte kann auch als technologische Artefakte bezeichnet werden und orientiert sich an den klassischen Phasen des Softwareentwicklungsprozesses Analyse, Design, Implementierung und Einsatz:

- Anforderungsartefakte
- Entwurfsartefakte
- Implementationsartefakte
- Einsatzartefakte

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 des Moduls und definieren die Rahmenbedingung für die Implementierung, wie z.B. Architektur, Modularisierung oder Schnittstellen. Aufgrund der bisher durchgeführten OSS-Studien wird angenommen, dass dieser Metaklasse wenige oder gar keine expliziten Artefakte zugeordnet werden können.

Implementierungsartefakte beinhalten alle implementationsabhängigen Artefakte und umfassen somit die elementaren Artefakte im OSS-Kontext, dessen Fundament sich stark an der Entwicklung und sukzessiven Konkretisierung von prototypischen Softwaremodulen orientiert.

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

Diese Mengeneinteilungen der Artefakte werden ebenfalls als Metaklassen in das Metamodell integriert, wobei die verschiedenen Elemente dieser Teilmengen als Basis dienen, aber im Verlauf der konkreten und projektspezifischen Artefaktidentifizierung noch durch verschiedene Ausprägungen, respektive weitere Spezialisierungen, erweitert werden können.

#### 1.3.3.4 Werkzeuge

Verwendete 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 ('Role') 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 verwendet, die Artefakte ('Artifact') zu verwenden oder zu modifizieren und können somit zu einer Zustandsänderung eines Objekts beitragen.

Nach [Draxler02] sind die meisten Entwicklungs- bzw. Management-Tools direkt mit einer der entsprechenden Artefaktmengen assoziierbar, wodurch sich diesbezüglich eine ähnliche Einteilung vornehmen ließe. Aufgrund der Tatsache, dass die Werkzeuge und Artefakte im OSS-Kontext eine, relativ zur klassischen Systementwicklung, recht divergente Gewichtung aufweisen, wurde diesbezüglich eine andere Unterteilung in die folgenden Kategorien vorgenommen, die auf einer Klassifizierung in [Scacchi02] basiert:

- Collaboration Tool
- Communication Tool
- Development Tool

Diese Kategorien werden ebenfalls durch von der Oberklasse abgeleitete Metaklassen in das Metamodell integriert.

'Collaboration Tool' repräsentiert 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) oder Datenbanken zur Verwaltung von registrierten Softwarefehlern (z.B. Bugzilla).

Die Metaklasse 'Communication Tool' umfasst alle Werkzeuge, welche primär zur projektinternen und -externen Kommunikation verwendet werden, wie z.B. Mailing Listen, Newsgroups oder Internet Relay Chat.

'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 Fallstudie des OSS-Projekts Linux Kernel

## 2.1 Linux-Kernel: Projektüberblick

"Linux is a clone of the operating system Unix, written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers across the Net." [KernelOrg02] Linus Torvalds ist sowohl Initiator des Projektes als auch immer noch aktiver Maintainer: "The development of this OSS was started in 1990 by the Finnish computer science student Linus Torvalds, who is still managing the project." [HeNiHe02]

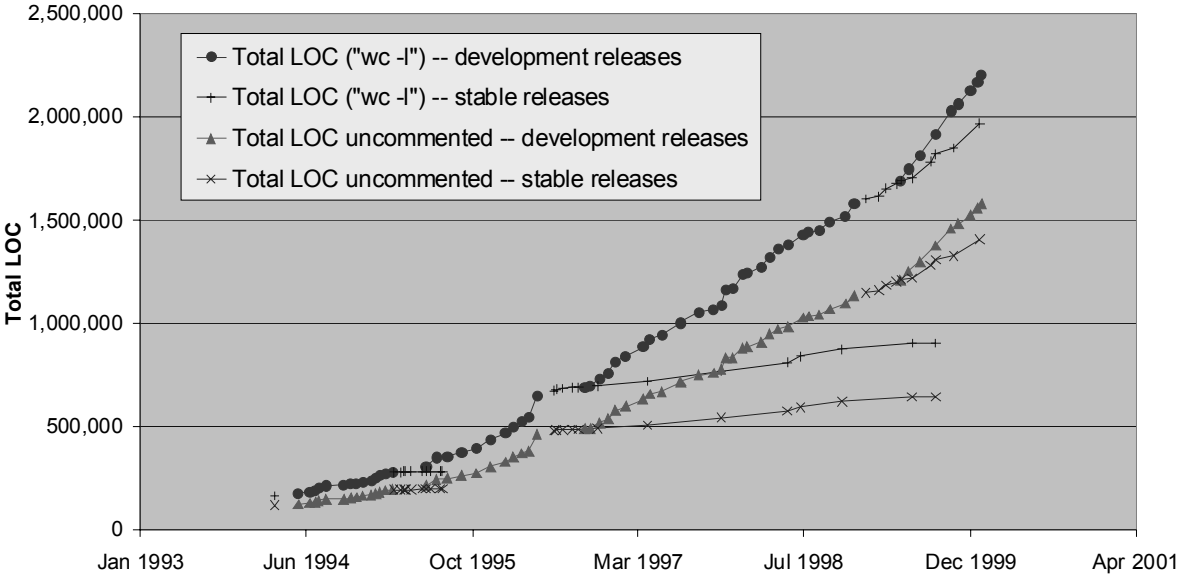
### 2.1.1 Evolution des Kernels

Der erste Release des Prototypen war die Version 0.01, welche noch sehr stark von der ursprünglichen Ausgangssoftware, dem Minix-Betriebssystem abhängig war und dessen Vorhandensein voraussetzte. Diese Version wurde im Jahr 1991 veröffentlicht, enthielt bereits eine Benutzerschnittstelle und konnte einen C-Compiler ausführen: "By October 5, 1991, version 0.01 was able to run the bash shell that provided an interface for sending commands to the kernel, as well as gcc, the GNU C Compiler." [Sandred01]

Mit Version 0.11 wurde Linux zu einem Standalone-System weiterentwickelt, welches nicht mehr vom Minix-Betriebssystem abhängig war. Zu diesem Zeitpunkt existierten bereits Mailing Listen und Newsgroups, die nach [Sandred01] von Hunderten von Anwendern frequentiert wurden. Durch denfrühzeitigen Release des Prototypen, häufige Updates und die Tatsache, dass bereits eine rudimentäre Kommunikationsinfrastruktur von Linus Torvalds zur Verfügung gestellt wurde, wurde die Basis für das sich so entwickelnde Entwicklungsmodell zur Softwareverbesserung geschaffen: "By releasing the source code for free at a very early stage and also updating the releases often, Torvalds quickly found help, support, and feedback from other programmers." [Sandred01]

Die folgende Grafik aus [GoTu00] visualisiert das Wachstum des Kernels über den gesamten Lebenszyklus anhand der enthaltenen Lines of Code (LOC):

Bild 4 Linux: Evolution des Kernels (aus [GoTu00])




"The first official release of the kernel, version 1.0, occurred in March 1994. This release contained 487 source code files comprising over 165,000 lines of code (including comment and blanks lines)." [GoTu00]

Inzwischen ist der Umfang des Quellcodes und die damit verbundene Komplexität auf ein sehr hohes Niveau angewachsen: "The Linux kernel is a large and complex software product that currently contains about 2 million lines of source code…" [HeNiHe02]. In [BoHoBr99] wird das Wachstum des Linux-Kernels als ein vergleichsweise rasanter Prozess beschrieben: "Linux is also a system that is growing rapidly; the source code for the Linux system has approximately doubled every year from 10 KLOC in 1991 to 1.5 MLOC in 1998…. The Linux kernel is itself a large system (approximately 800 KLOC)."

Außerdem verdeutlicht obige Grafik die unterschiedliche Evolution zweier Quellcodepfade, die eines stabilen und die eines Entwicklungspfades. Der Interessenkonflikt zwischen den Endnutzern, welche ein stabiles und getestetes System erwarten, und den Entwicklern, welche Patches schnell und direkt in den Quellcode integrieren wollen, führte zu der Realisierung zweier unterschiedlicher Quellcodepfade: "As a result of this tension between end-users and developers, the development of Linux has been split into two development paths." [Tuomi00]. Ein Pfad dient dabei als Basis für Entwickler und zur Integration von ungetestetem Quellcode, während der stabile Pfad stets getesten und weitgehend stabilen Quellcode beinhaltet: "One path is a "stable" path, where only essential changes are introduced. The other path is a

"development" path, with continuous integration of new and useful code components." [Tuomi00]

### 2.1.2 Systemeigenschaften

Die Funktionalitäten des Linux-Kernels sind mit denen professioneller Unix-Derivate vergleichbar, "including true multitasking, virtual memory, shared libraries, demand loading, shared copy-on-write executables, proper memory management, and TCP/IP networking." [KernelOrg02]. Der Linux-Kernel übernimmt die klassischen Aufgaben eines Betriebssystem-Kerns: "[...] is responsible for process, memory, and hardware device management." [BoHoBr99]

Linux ist auf allen gebräuchlichen Plattformen verfügbar: "Linux was first developed for 32-bit x86-based PCs (386 or higher). These days it also runs for example on the Compaq Alpha AXP[2], Sun SPARC[3] and UltraSPARC[4], Motorola 68000[5], PowerPC[6], PowerPC64[7], ARM[8], Hitachi SuperH[9], IBM S/390[10], MIPS[11], HP PA-RISC[12], Intel IA-64[13], DEC VAX[14], AMD x86-64[15] and CRIS[16] architectures. Linux is easily portable to most general-purpose 32- or 64-bit architectures as long as they have a paged memory management unit (PMMU) and a port of the GNU C compiler (gcc)[17]." [KernelOrg02]

Dadurch existieren Portierungen selbst für Mainframe-Architekturen oder Embedded Systems wie z.B. mobile PDA-Systeme (Personal Digital Assistant).

2 http://www.linuxalpha.org/
3 http://www.ultralinux.org/
4 http://www.ultralinux.org/
5 http://www.linux-m68k.org/
6 http://penguinppc.org/intro.shtml
7 http://linuxppc64.org/
8 http://www.arm.linux.org.uk/
9 http://linuxsh.sourceforge.net/
10 http://oss.software.ibm.com/linux390/index.shtml
11 http://oss.sgi.com/mips/howtos.html
12 http://www.parisc-linux.org/
13 http://www.linuxia64.org/
14 http://linux-vax.sourceforge.net/
15 http://www.x86-64.org/
16 http://developer.axis.com/software/linux/
17 http://gcc.gnu.org/

### 2.1.3 Projektstruktur

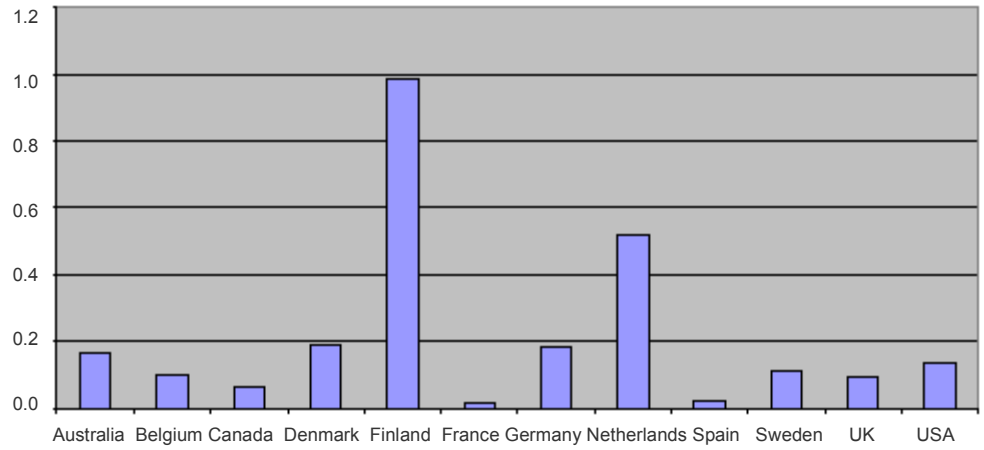
"In contrast to many arguments that effective virtual collaboration requires existing social interactions, the Linux developer community has been almost completely virtual. Furthermore, it started as a virtual community." [Tuomi00]

Eine wichtige Eigenschaft des Linux-Kernel-Projekts ist der Umfang und die Heterogenität der Projekt-Community: "The CREDITS file as a list of the contributors to the kernel contains 400 entries, and about 3.500 people were subscribed to the Linux kernel mailing list at the time of this study." [HeNiHe02]. In [FeFi02] wird auf eine Entwicklergemeinde verwiesen, die über 40.000 Entwickler umfasst.

Wichtige Charakteristika des Linux-Projekts sind nach [Godfrey00b] neben der sehr großen Anzahl der Releases, welche über einen vergleichsweise langen Zeitraum veröffentlicht wurden, die sehr umfangreiche Community, die inzwischen sehr große Akzeptanz bei den Endanwendern erreicht hat und ein damit verbundener steigender Marktanteil im Segment der client- und serverseitigen Betriebssysteme. Das Linux-Projekt repräsentiert einen wichtigen Prototyp für OSS-Projekte und kann als das weltweit größte kollaborative Softwareprojekt betrachtet werden: "[...] Torvalds claims, with uncharacteristic immodesty, that it represents the largest collaborative project in the history of the world." [FeFi02]. Als weltweit erstes Softwareentwicklungsprojekt, welches unter dem Paradigma von Open Source durchgeführt wurde, repräsentiert es den Prototyp des OSS-Modells, der sich aufgrund der Abstinenz von Erfahrungen in vergleichbaren Projekten komplett evolutionär entwickelt hat.

Bereits von der Initialisierung des Projektes an basierte die Organisation und Koordination der Entwicklung auf internet-gestützter Zusammenarbeit und ermöglichte dadurch die Etablierung einer dezentral organisierten und sehr heterogenen Struktur der Projekt-Teilnehmer. Dies zeigte sich bereits kurz nach dem initialen Release und wird in der folgenden Abbildung anhand der Nationalitäten der Projektmitglieder visualisiert:

Bild 5 Linux: Anzahl der Entwickler pro Land (nach [Tuomi00])




Die Daten dieser Grafik wurden anhand des ersten CREDITS-Files im März 1994 gewonnen (vgl. [Tuomi00]).

[Rosenberg00] kommt basierend auf [DeWeJoGr99] zu der Erkenntnis, dass das gesamte Linux System zwar von einer Vielzahl von Entwicklern entwickelt wird, aber die Entwicklung des Kernels primär von wenigen verteilten Entwicklern vorangetrieben wird: "The Metalab study shows, that there are fewer than 200 accepted contributors to the kernel..." [Rosenberg00]. Der Begriff ‚accepted contributors' repräsentiert dabei alle Entwickler, die bereits erfolgreich Quellcode in den Kernel integriert haben und umfasst demzufolge nicht die Vielzahl an Entwicklern, welche Patches entwickelten, die nicht in eine Kernel-Version integriert wurden.

Da diese Studie ausschließlich auf die Entwicklung des Linux Kernels fokussiert ist, werden in den folgenden Analysen auch lediglich die damit assoziierten Prozesse und die darin involvierten Akteure betrachtet.

## Organisation

[HeNiHe02] kommt zu der Erkenntnis, dass die Kernel-Entwickler in verschiedenen Quasi-Teilprojekten organisiert sind, die entsprechend der verschiedenen Subsysteme des Kernels identifizierbar sind, aber nicht als explizit organisierte Teilprojekte zu sehen sind.

Neben der Organisation der Entwickler entsprechend der von ihnen bearbeiteten Subsysteme, existiert eine Organisation der Anwender und Entwickler in regional etablierten User Groups, welche inzwischen in einer Vielzahl von Ländern existieren und sich über Meetings und internetbasierte

Kommunikation organisieren: "Some hold monthly or quarterly meetings, while others maintain active mailing lists and newsletters." [LinuxOrg02]

## 2.2 Identifizierte Prozesse

Dieses Kapitel beschäftigt sich mit der semantischen Beschreibung aller identifizierten Prozesse, Teilprozesse und Aktivitäten, die im Rahmen des Linux-Projekts eruiert werden konnten. Diese werden nicht bis auf die Ebene der atomaren Aktivitäten konkretisiert, sondern bis zu einem sinnvoll darstellbaren Detaillierungsgrad dargestellt. Dies bildet die Basis für die weitere Identifikation der im Metamodell definierten relevanten Aspekte, wie involvierte Rollen, Artefakte und Werkzeuge.

Zuerst werden in Kapitel 2.2.1 die Phasen des Projektes beschrieben. Im Anschluss daran erläutern Kapitel 2.2.2 bis Kapitel 2.2.4 die über die gesamte Laufzeit des Projektes identifizierten Prozesse, eingeteilt in die 3 Kategorien Management-, Entwicklungs- und infrastrukturelle Prozesse.

### 2.2.1 Projektphasen

Der Entwicklungsprozess im Rahmen der Linux-Kernel Entwicklung konnte in zwei grundlegende Phasen unterteilt werden:

1. Initiale Prototypentwicklung
2. Iterativer Entwicklungszyklus

#### 2.2.1.1 Die initiale Prototypentwicklung

Der Prozess der Entwicklung des initialen Prototyps des Kernels wurde durch den Initiator des Projektes, Linus Torvalds, ausgeführt und kann in dieser Prozessbetrachtung nur peripher thematisiert werden, da er nicht dokumentiert ist und in keinerlei Informationsquellen spezifiziert wurde. Die weitere Prozessbetrachtung beschränkt sich daher in erster Linie auf den iterativ ausgeführten Entwicklungszyklus zur sukzessiven Fehlerbehebung, Systemerweiterung und -verbesserung, der durch die stetig wachsende Entwicklergemeinde ausgeführt wird.

Weiterhin konnte eine Phase des Re-Engineering der Software identifiziert werden, welche sich an die initiale Prototypentwicklung anschließt. Im Rahmen dieser Phase wurde der Quellcode den Anforderungen der OSS-Entwicklung angepasst, indem der Kernel modularisiert wurde und dadurch parallele Entwicklungsprozesse durch verteilte Akteure ermöglicht wurden: "With the

Linux Kernel it became clear very quickly that we want to have a system whis is as modular as possible. The open source development model really requires this, because otherwise you can't easily have people working in parallel." [Torvalds99]]

#### 2.2.1.2 Der grundlegende iterative Entwicklungsprozess

[Tuomi00] definiert den Entwicklungsprozess als einen sukzessiven Verbesserungsprozess des initialen Prototypen, der von verschiedenen peripheren Prozessen begleitet wird:

Bild 6 Linux: Akteure im Linux Fehlerbeseitigungs-Prozess (nach [Tuomi00])

| processing phase | | | information resource | tool | community resource |
|---|---|---|---|---|---|
| detect | | | compiled code<br>documentation | man | LDP |
| debugging | characterize | | source code<br>linux-kernel list FAQ<br>JitterBug<br>oops-tracing.txt<br>Kernel Traffic<br>LDP<br>project-specific sites<br>linux-kernel archives<br>README files<br>log files<br>bug reporting form | editor<br>gcc<br>make<br>gdb<br>ksymoops<br>IRC<br>computer configuration | linux-kernel list<br>JitterBug<br>personal email<br>IRC channels<br>kernel-newsflash<br>LDP<br>project-specific lists |
| debugging | remove | | source code | editor<br>gcc<br>make | |
| debugging | remove | test | patch<br>MAINTAINERS file | diff<br>gcc<br>make<br>editor<br>ftp | personal email<br>linux-kernel list |
| distribute | | | patch<br>MAINTAINERS file | gzip<br>tar<br>email<br>ftp | linux-kernel list<br>JitterBug |
| integrate | | | patch<br>release | CVS<br>vger<br>package managers | Maintainers<br>vger |

Die "Information Resources" stellen in dieser Übersicht Artefakte dar, die im folgenden noch näher erläutert werden. Ebenso wird auf die in den Kategorien "tool" und "community resource" enthaltenen Werkzeuge explizit im Abschnitt "Werkzeuge" eingegangen. Die Kategorisierung in dieser Übersicht wird im weiteren Verlauf der Analyse nicht übernommen, sondern die

einzelnen Aspekte des Prozesses werden entsprechend des zugrundeliegenden Metamodells klassifiziert und dargestellt.

Die Prozesse zur Bereitstellung und Wartung dieser Werkzeuge werden im Kapitel 2.2.4 "Infrastrukturelle Prozesse" erläutert.

### 2.2.2 Managementprozesse

In diesem Abschnitt werden die zentral ausgeführten Prozesse thematisiert, welche nach den Spezifikationen des Metamodells als Managementprozess betrachtet werden.

#### 2.2.2.1 Projekt Maintenance

"Originally, Linus Torvalds maintained the kernel. As the kernel has matured, he has delegated maintenance for older stable versions to others, while he continues development of the latest "bleeding edge" release." [LkmlFaq02]

Neben dem Projektinitiator Linus Torvalds, der momentan noch die Verantwortung für die jeweils aktuelle Kernel-Version innehat, sind inzwischen nach [LkmlFaq02] verschiedene Personen mit der Pflege alter Kernel Versionen beschäftigt:

- Release 2.0: David Weinehall <tao@acc.umu.se>
- Release 2.2: Alan Cox <alan@lxorguk.ukuu.org.uk>
- Release 2.4: Marcelo Tosatti <marcelo@conectiva.com.br>
- Release 2.5: Linus Torvalds <torvalds@transmeta.com>

#### 2.2.2.2 Initiale Prototypentwicklung und -release

Über diesen Prozess existieren nur rudimentäre Informationen, daher kann auch nur auf einem entsprechend hohen Abstraktionsniveau darauf eingegangen werden.

##### 2.2.2.2.1 Initiale Prototypentwicklung

Es ist dokumentiert, dass Linus Torvalds im Jahre 1990 den Kernel mit der Intention zu entwickeln begann, ein Unix-orientiertes Betriebssystem zu entwickeln, welches frei verfügbar sein sollte (vgl.: [Weber00]). Die Entwicklungstätigkeit von Linus Torvalds basierte dabei auf dem von Andrew Tanenbaum entwickelten Unix-Derivat Minix, wobei anfänglich Assembler und später C als Programmiersprache verwendet wurde (vgl. [Grassmuck02],

[Sandred01]). Minix selbst verfügte zu diesem Zeitpunkt bereits über eine relativ große Anwendergemeinschaft: "Within two months of its release in 1987 there was a newsgroup, comp.os.minix, with over 40000 users worldwide." [Sandred01]

Somit basierte auch die initiale Entwicklung des Linux-Kernels auf einem prototypischen Softwareprodukt: "Linux development process, for example, relied heavily on the existence of the Unix operating system, especially its BSD and Minix variants, newsgroups and listservers on the Internet, the GNU c-compiler and its libraries, and the GNU General Public License." [Tuomi00]

#### 2.2.2.2.2 Initialer Prototyprelease

Primär wurde im Rahmen der initialen Projektveröffentlichung der Quellcode im Internet bereitgestellt und über die Newsgroup des Minix-Projekts bekanntgemacht: "In autumn 1991, Torvalds let go off the Minix scaffold and released the source code for the kernel of his new operating system, which he called Linux, onto an Internet newsgroup…" [Weber00]. Die erste veröffentlichte Kernel-Version umfasste lediglich einen Unix-Kernel und wies noch starke Abhängigkeiten zum Minix System auf: "The first release version 0.01 of Linux contained nothing more than a rudimentary Unix kernel, and it was still hosted by Minix (the kernel could not run without Minix)." [Sandred01]. Die ursprüngliche Intention von Linus Torvalds umfasste nach [Sandred01] damals noch nicht die Anforderungen, ein möglichst portables, vollständiges und professionell einsetzbares Betriebssystem zu entwickeln.

Im Anschluss an die Veröffentlichung erhielt Linus Torvalds sehr viel konstruktives Feedback, wodurch die prototypische Evolution eines Entwicklungsmodells unter dem Paradigma von Open Source begann: "Of the first ten people to download Linux, five sent back bug fixes, code improvements and new features." [Sandred01]

Für die Evolution des weiteren Entwicklungsprozesses waren die verschiedenen frei verfügbaren Werkzeuge wie Compiler, Editoren oder ein geeignetes Lizenzmodell von großer Bedeutung: Linux development process, for example, relied heavily on the existence of the Unix operating system, especially its BSD and Minix variants, newsgroups and listservers on the Internet, the GNU c-compiler and its libraries, and the GNU General Public License. Before Linux development started as a collaborative effort, many technology and knowledge creation communities had already been translated as resources for it." [Tuomi00]. Die normalerweise typischen Aktivitäten im Rahmen dieses Teilprozesses, wie die Wahl oder Erstellung einer Softwarelizenz oder die umfassende Bereitstellung einer Projektinfrastruktur wurden hier erst in späteren Phasen des Projekts ausgeführt, da das Linux-Projekt als eines der ersten Projekte im Kontext von Open Source noch nicht auf entsprechende

Erfahrungen zurückgreifen konnte und die Bedeutung von verschiedenen infrastrukturellen Voraussetzungen und der Wahl der Lizenz erst im späteren Verlauf des Projektes offenbar wurden.

#### 2.2.2.2.3 Etablierung des Lizenzmodells

"The arrival of the Linux System in the early 1990s was also made possible by its Free Software licensing scheme…" [Rosenberg00]

Dieser Teilprozess ist i.d.R. fester Bestandteil des initialen Veröffentlichungsprozesses. Da aber die Bedeutung einer sinnvollen Softwarelizenz noch nicht hinreichend bekannt war und noch keine nennenswerten Erfahrungen aus anderen Projekten der Freien Software existierten, wurde diese Aktivität erst im späteren Verlauf des Projekts ausgeführt.

Nach der Veröffentlichung des Quellcodes wurde der Kernel durch die Community um verschiedene Elemente des GNU-Systems erweitert und später auch die GNU General Public License (GNU GPL) übernommen: "Ebenfalls vom GNU-Projekt übernommen wurde die Copyleft-Lizenz (GPL), die die Freiheit von Linux und aller abgeleiteter Software sichert." [Grassmuck02]. Daraus resultiert auch die heutige Bezeichnung GNU/Linux.

### 2.2.2.3 Entwicklung und Kommunikation von Leitlinien

Dieser Teilprozess der Managementprozesse stellt eine wichtige Basis für den Entwicklungsprozess dar, da die projektweiten Guidelines den Entwicklungsprozess beschreiben und damit als Vorgabe für die gesamte Community dienen und den Prozess maßgeblich determinieren.

Wichtige Artefakte, welche auf diesem Teilprozess basieren, sind die "Bug Reporting Guidelines" [Gooch02], verschiedene Programmierkonventionen (vgl.: [Salzmann01], [Ebling03]) oder allgemeine Verhaltensrichtlinien wie z.B. beschrieben in [Raymond01b].

### 2.2.2.4 Releaseprozess

Die Entscheidung über den Zeitpunkt und den Umfang einer neuen Software Version wird durch den Maintainer des Projektes getroffen und alle notwendigen Aktivitäten von ihm geplant und ueberwacht. Für den Veröffentlichungsprozess ist es kaum von Bedeutung, ob es sich bei der zu veröffentlichenden Version um ein neues Release des stabilen oder des Entwicklungspfades handelt.

In den meisten Fällen wird der Release einer neuen Version vom Maintainer über die LKML angekündigt. Diese Information signalisiert allen Entwicklern, die noch an Patches arbeiten, diese schnellstmöglich abzuschließen und dem Maintainer für eine Integration in den Quellcode zur Verfügung zu stellen.

Nach dem Ablauf eines hinreichenden Zeitraumes wird der Quellcode "eingefroren", um damit das Hinzufügen neuer Funktionalitäten, also auch zusätzlicher potentieller Fehlerquellen zu verhindern. Hierbei wird nach dem Grad der Restriktivität zwischen den zwei Varianten "Feature Freeze" und "Code Freeze" unterschieden:

#### 2.2.2.4.1 Feature Freeze

Um eine Destabilisierung des Quellcodes zu vermeiden, wird durch den Maintainer der Feature Freeze ausgeführt, bei dem keine neuen Funktionalitäten mehr in den Quellcode integriert werden. "[the maintainer] announces a feature freeze to indicate that he will not be considering any more new features until the release of a new stable kernel. ... The idea is to introduce a period of stability to hunt down and fix the worst of the bugs introduced by all the new features." [Ebling03]. Um eine Destabilisierung des Quellcodes zu vermeiden, wird durch den Maintainer der Feature Freeze ausgeführt, bei dem keine neuen Funktionalitäten mehr in den Quellcode integriert werden.

#### 2.2.2.4.2 Code Freeze

"Code freezes are similar in concept to feature freezes, but are more restrictive; only fixes for serious bugs are accepted." [Ebling03]
Damit stellt der restriktivere Code Freeze einen weiteren wichtigen Schritt im Veröffentlichungsprozess einer neuen stabilen Kernelversion dar.

#### 2.2.2.4.3 Release der neuen Version

Nachdem der Quellcode eine bestimmte Zeit eingefroren wurde, in diesem Zeitraum nur noch gravierende Fehler beseitigt wurden und der Quellcode letzten Tests unterzogen wurde, wird die neue Version mit allen zugehörigen Artefakten in den Linux Kernel Archiven publiziert und damit allgemein verfügbar. Die Veroeffentlichung der neuen Version wird zudem primär über die LKML und auch über die anderen einschlägigen Kommunikationswege (siehe Kapitel 2.5.1) kommuniziert.

### 2.2.3 Entwicklungsprozesse

Dieser Abschnitt thematisiert die Prozesse und Aktivitäten, welche direkt an der Softwareentwicklung beteiligt und i.d.R. den klassischen Phasen der Softwareentwicklung zugeordnet werden können.

#### 2.2.3.1 Aktive Mitwirkung

Anwender des Linux-Kernels können auf verschiedene Weise am iterativen Entwicklungsprozess teilhaben und dadurch zu aktiven "Contributors" bzw. "Developers" werden. Diese Rollen werden im Kapitel 2.3 naeher beschrieben.

Im Rahmen des Linux Kernels kann ein individueller Akteur verschiedene Beiträge zum Gesamtprojekt leisten:

- Anwenderprobleme kommunizieren und diskutieren
- Fehlerberichte erstellen
- Anregungen bezüglich funktionaler Erweiterungen geben
- Dokumentationsartefakte beitragen oder modifizieren
- Quellcode erstellen oder modifizieren

Im folgenden werden einzelne Teilprozesse beschrieben, die im Laufe der Recherche eruiert werden konnten und für den Softwareentwicklungsprozess von Bedeutung sind.

#### 2.2.3.2 Kommunikation über dedizierte Kommunikationskanäle

"The behavior of kernel programmers is mainly organized by the Linux kernel mailing list. This mailing list acts as a central place for discussions of technical and organizational aspects of the kernel development." [HeNiHe02]
Die zentrale Mailing Liste des Linux Kernels steht unter linux-kernel@vger.kernel.org zur Verfügung. Das Eintragen in diese Liste steht prinzipiell jedem Interessierten frei.

Neben der zentralen Linux Kernel Mailing List, existieren verschiedene spezielle Mailing Listen. [HeNiHe02] beschreibt diese Listen wie folgt: "dealing with more specific technical aspects of the kernel development, such as the development of certain filesystem or hardware drivers. These fields are also called 'subsystems' of the kernel."

[LkmlFaq02] definiert verschiedene Verhaltensregeln und Vorgaben bezüglich Format, Formatierung und erlaubter Inhalte der Message, welche die Kommunikation über Mailing Listen regulieren und vereinfachen sollen und damit die kollaborative Zusammenarbeit ermöglichen. Diese Notwendigkeit

resultiert u.a. aus der Tatsache, dass die Mailing Listen i.d.R. nicht durch eine zentrale Instanz moderiert werden und somit die Akteure absolut frei und autark miteinander kommunizieren. In [LkmlFaq02] wird als projektintern zu verwendende Sprache Englisch definiert. Desweiteren wird auf E. S. Raymonds Verhaltensregeln in [Raymond01b] verwiesen.

#### 2.2.3.3 Bezug des Quellcodes

Der Quellcode des Linux Kernels wird auf verschiedenen Web- und FTP-Seiten zum Download zu Verfügung gestellt: "The primary site for the Linux kernel (experimental and production) sources is hosted by Transmeta (the company Linus Torvalds works for) on a dedicated Web server at http://www.kernel.org. This site is mirrored across the world, and has pointers to mirrors for each country." [LkmlFaq02]. Das Archiv wird vom Maintainer des Projektes, Linus Torvalds verwaltet und stellt damit die offizielle und aktuelleste Basis für den Quellcode von Kernel-Releases dar.

Durch die vielfache Spiegelung des originalen Linux Kernelarchivs ist sichergestellt, dass jederzeit und von überall auf den Quellcode der aktuellen Releases zugegriffen werden kann: "They [the kernel archives] are kept on ftp://ftp.cs.helsinki.fi and all of the other web sites shadow them." [Rusling02]

Darauf basierende, gespiegelte Websiten sind zum Beispiel unter URLs, die dem folgenden Schema entsprechen, zu erreichen: http://www.countrycode.kernel.org. 'Countrycode' muss dabei durch das entsprechende Kürzel für das jeweilige Land, z.B. durch 'de' für Deutschland, ersetzt werden.

Die Releases auf den Seiten der Linux Kernel Archive werden unterteilt in noch weitgehend instabile Beta-Releases und die stabilen Kernelversionen, welche bereits eine Testphase durchlaufen haben. Neben diesen offiziellen Releases steht über verschiedene alternative Quellcodearchive (z.B. CVS) der aktuellste Entwicklungsquellcode im Rahmen des jeweiligen Entwicklungsprojekts zur Verfügung.

Weiterhin enthalten auch die verschiedenen verfügbaren Linux-Distributionen wie RedHat, Debian oder SuSe eine meist ältere, aber getestete und stabile Version des Linux Kernels, weswegen die Distributoren ebenfalls als Bezugsquelle betrachtet werden können.

Updates der Software werden in Form von Patches kommuniziert, welche z.T. auch direkt veröffentlicht werden: "Changes to the kernel sources are distributed as patch files. The patch utility is used to apply a series of edits to a set of source files. [...] A good source of kernel patches (official and unofficial) is the http://www.linuxhq.com web site." [Rusling02]. Außerdem können

Patches auch von anderen FTP-Servern[18] bezogen werden. [LkmlFaq02] beschreibt den FTP Server wie folgt: "… which [the ftp server] is where Linus [Linus Torvalds] distributes his kernels from. Other notable kernel hackers have directories under the people directory, which is where they keep their kernel patches."

Weiterhin werden sogenannte Pre-Release-Patches veröffentlicht, welche versierte Anwender und Entwickler adressieren und über die Veröffentlichung dem Test-Cycle zugeführt werden: "The testing directory is where Linus puts pre-release patches." [LkmlFaq02]. Diese Patches repräsentieren aktuelle Quellcodeversionen analog zu den instabilen, sogenannten Alpha- bzw. Beta-Releases in anderen OSS-Projekten.

**Quellcode über CVS**

CVS (Controlled Versioning System) ist eine open source SW und dient zum Konfigurationsmanagement und zur Unterstützung der kollaborativen Bearbeitung von Quellcode. Es stellt im Rahmen der Linux Kernel Entwicklung zwar kein offiziell und projektweit verwendetes Werkzeug zur Konfigurations- und Versionsverwaltung dar, kann aber dennoch als elementare Resource im Rahmen des Entwicklungsprozesses angesehen werden.

Trotzdem existieren neben den offiziellen Kernel-Archiven, die zur externen Entwicklung nicht über CVS bereitgestellt werden, auch CVS-Archive: "[...] vger.kernel.org is a mailing list and Web server that faithfully serves the Linux community. A CVS kernel development tree is also available at http://vger.samba.org/ and is an up-to-date mirror of the master tree at vger.kernel.org." [LkmlFaq02]. Weiterhin weist [LkmlFaq02] darauf hin, dass dieser CVS-Baum nicht das offizielle Code-Archiv repräsentiert und dieses ausschließlich über das offizielle Linux Kernel Archiv[19] verfügbar ist. Verschiedene sinnvolle und getestete Patches des CVS Entwicklungsarchives werden nach entsprechenden Tests in das offizielle Kernel Archiv integriert.

#### 2.2.3.4 Softwarefehler berichten

Eine Möglichkeit des aktiven Mitwirkens am Kernel-Entwicklungsprozess ist das Berichten von Software-Bugs, wobei das technologische Artefakt "Bug Report" generiert wird, welches in Kapitel 2.4.2 noch explizit erläutert wird.

18 z.B. von ftp://ftp.CODE.kernel.org/pub/linux/kernel/
19 http://www.kernel.org

Im Kontext der Linux Kernel Entwicklung wird zwischen einem "Bug" und einem "Oops" differenziert, wobei ein "Oops" genau dann auftritt, wenn der Kernel ein Fehlverhalten registriert und dieses durch eine entsprechende Meldung ("Oops") signalisiert. Ein "Bug" bezeichnet ein inhärentes Fehlverhalten des Kernels, das somit auch nicht durch den Kernel, sondern nur durch den Anwender registriert werden kann und daher nicht notwendigerweise ein "Oops" generiert.

Eine gemeinsame, konsequent projektweit verwendete und zentral durch den Maintainer verwaltete Datenbasis für "Bug Reports" existiert nicht. [LkmlFaq02] verweist diesbezüglich auf verschiedene Informationsquellen wie z.B. die Mailing Listen Archive oder die Inhalte einer zentral verwalteten Jitterbug-Bugtracking-Datenbank[20], in denen verschiedene, bereits identifizierte Software-Fehler oder allgemeine Änderungsanforderungen beschrieben werden. Detailliertere Informationen zu Jitterbug enthält Kapitel 2.5. Weiterhin verweist [Gooch02] auf die Datei `REPORTING_BUGS` aus dem Kernel Source Tree und die Datei `oops_tracing.txt` aus dem Dokumentationsverzeichnis des Kernels, in denen das Prozedere des Bug Reporting beschrieben wird.

Obwohl bereits standardisierte Bug Report Formulare entwickelt wurden (siehe [LkmlFaq02]), werden diese nicht konsquent projektweit und homogen verwendet. Daher definieren verschiedene, heterogene Quellen die notwendigen Informationen, welche ein "Bug Report" enthalten sollte.

#### 2.2.3.4.1 Inhalt und Format des Bug Reports

"Perhaps the most important step is to determine under which conditions this misbehaviour can be triggered, and whether it is reproducible." [LkmlFaq02]

[Kernel02] definiert folgende Vorgaben für die Präparation eines "Bug Reports":

- Kompilierungsfehler: Ausgabe des Kompilers (empfohlen: GCC) berichten
- Oops: Decodieren des Oops mit dem Tool ksymoops

Weiterhin definieren [Gooch02] und [Tatham02] neben verschiedene Leitlinien, auf welche referenziert wird, auch ein standardisiertes Format für die Aufbereitung von "Bug Reports". Nach dem Ausführen des Skripts `ver_linux`, welches sich im jeweiligen Scripts-Verzeichnis befindet, sollte es dem Akteur möglich sein, die Ausgabe dieses Skripts zum Ausfüllen des

20 http://samba.anu.edu.au/cgi-bin/linux-patches

folgenden Formulars aus [Gooch02] zu verwenden:

```
[1.]  One line summary of the problem:
[2.]  Full description of the problem/report:
[3.]  Keywords (i.e., modules, networking, kernel):
[4.]  Kernel version (from /proc/version):
[5.]  Output of Oops.. message (if applicable) with symbolic information
      resolved (see Documentation/oops-tracing.txt)
[6.]  A small shell script or example program which triggers the problem (if
      possible)
[7.]  Environment
[7.1] Software (add the output of the ver_linux script here)
[7.2] Processor information (from /proc/cpuinfo):
[7.3] Module information (from /proc/modules):
[7.4] Loaded driver and hardware information (/proc/ioports, /proc/iomem)
[7.5] PCI information ('lspci -vvv' as root)
[7.6] SCSI information (from /proc/scsi/scsi)
[7.7] Other information that might be relevant to the problem (please look
      in /proc and include all information that you think to be relevant):
[X.]  Other notes, patches, fixes, workarounds:
```

#### 2.2.3.4.2 Kommunikation des Bug Reports

Nach [Kernel02] beinhaltet die präferierte Vorgehensweise, zuerst den verantwortlichen Entwickler ausfindig zu machen und falls dies unmöglich sein sollte, dem entsprechenden Subsystem Maintainer den Bug Report direkt zukommen zu lassen: "Small fixes and device driver updates are best directed to subsystem maintainers." Listen der entsprechenden Maintainer werden in [Kernel02] und in den verschiedenen relevanten Informationsquellen, z.B. den Linux Kernel Archiven, zur Verfügung gestellt.

Als sekundäre Adressaten für Bug Reports werden in den Bug Reporting Guidelines [Gooch02] die allgemeinen Mitglieder der Linux Kernel Mailing Liste definiert.

### 2.2.3.5 Anforderung einer funktionalen Systemerweiterung

Neben den "Bug Reports" kann auch der Wunsch nach zusätzlichen Funktionalitäten oder Erweiterungen ("Enhancement Request") durch Anwender formuliert werden, wobei das Prozedere dem des Beitragens von "Bug Reports" prinzipiell ähnelt. Nach der Kategorisierung der Nutzeranforderung als Kernel-spezifische Anforderung empfiehlt [LkmlFaq02] die weitere Recherche nach evtl. äquivalenten oder zumindest ähnlichen Anfragen in den bezeichneten Informationsquellen wie Jitterbug oder den verschiedenen Archiven der Linux Kernel Mailing Liste.

Da diese neue Anforderung durch eine Modifikation des Quellcodes, also durch einen Patch, befriedigt werden soll, wird als optimales Vorgehen die direkte

Entwicklung eines Patches durch den Inititator des Enhancement Requests betrachtet. Daher empfiehlt [LkmlFaq02] diesem Anwender folgende Vorgehensweise:

1. Artikulation der Anforderung
2. Generierung eines ersten Patches zur konkreten Anforderungsspezifikation
3. Kommunikation des Patches über die Linux Kernel Mailing Liste

Der Prozess des Beitragens von Patches wird im entsprechenden Abschnitt dieses Kapitels noch konkretisiert.

Sofern der Anwender nicht in der Lage sein sollte, selbständig ein initiales Patch zu entwickeln, wird als Alternative die verbale und möglichst eindeutige Beschreibung und Kommunikation der Anforderung über die Linux Kernel Mailing Liste nahegelegt. Dieser Prozess hat zur Folge, dass die Community der Linux Kernel Entwickler die neue Systemanforderung diskutiert und bei gegebenem Interesse und weithin akzeptierter Relevanz der Anforderung, diese bis zum nötigen Reifegrad hin implementiert.

#### 2.2.3.6 Beitragen von Dokumentationsartefakten

[LkmlFaq02] definiert das Beitragen von Dokumentationsartefakten als weitere Möglichkeit, am Projekt mitzuwirken. Mögliche Varianten dieses Anwendungsfalles sind nach [LkmlFaq02] das Schreiben oder Überarbeiten von Manpages oder das Verfassen von Dokumentationen, die über die Mailing Liste bzw. einen IRC (Internet Relay Chat) Kanal einem Peer-Review unterzogen werden. Aufgrund der meistens sehr informellen und individuell verschieden ausgeführten Aktivitäten im Rahmen dieses Teilprozesses, wird auf eine weitere Beschreibung der Aktivitäten verzichtet.

#### 2.2.3.7 Beitragen von Patches

"If your patch is something big and new, announce it on LKML and try to attract testers. After it has been tested and discussed, you can expect Linus to consider inclusion in mainline." [Kernel02]

Als primären Kanal für die Kommunikation von Patches wird [LkmlFaq02] die LKML mit folgendem Vorgehen definiert:

1. Entwicklung des initialen Patches
2. Generierung einer begleitenden README-Datei
3. Message für LKML generieren und versenden

4. Test des Patches
5. Integration in ein Quellcode Archiv

#### 2.2.3.7.1 Generierung eines Patches

[LkmlFaq02] empfiehlt für die Entwicklung des Patches das Werkzeug `diff`, welches es erlaubt, aus zwei verschiedenen Quellcode-Verzeichnissen bzw. Dateien lediglich die Quellcodemenge zu extrahieren, welche sich nicht in der Schnittmenge beider Verzeichnisbäume befindet: "The easiest way to do this is to set up two source trees under `/usr/src`, set a symlink '`/usr/src/linux`' to point to the modified tree, and diff one tree against the other." Es ist dabei auch möglich, einen der beiden Verzeichnisbäume lediglich logisch und basierend Verknüpfungen mit dem Originalbaum zu generieren.

`Diff` ist Bestandteil des Kernels und daher für jeden Linuxanwender verfügbar.

Weiterhin wurden verschiedene Leitlinien definiert, wie z.B. die Verwendung des aktuellsten Ausgangsquellcodes und die Vermeidung von unnötigen Formatierungsänderungen.

Vor der Kommunikation des Patches sollte der Autor seinen Quellcode in jeder Datei mit seinem Namen versehen, um die spätere Identifikationsmöglichkeit sicher zu stellen. Weiterhin empfiehlt [Russel00], den eigenen Namen in den entsprechenden `CREDITS`- und `MAINTAINERS`-Dateien zu vermerken: "Put yourself in `CREDITS` if you've done something noteworthy, usually beyond a single file (your name should be at the top of the source files anyway). `MAINTAINERS` means you want to be consulted when changes are made to a subsystem, and hear about bugs; it implies a more-than-passing commitment to some part of the code." [Russel00]

Die korellierende `README`-Datei wird als "an one line `README` file describing it very succinctly" ([LkmlFaq02]) umschrieben, die hauptsächlich dazu dient, den Autor mit dem Patch assoziieren zu können.

#### 2.2.3.7.2 Kommunikation von Patches

"[...] one purpose of the list [LKML] is to get patches peer-reviewed and well-tested." ([LkmlFaq02]) Ziel der Kommunikation des Patches an die Community ist das umfassende Testen und Überarbeiten des Patches um diesen vor der Integration in die zentralen aktuellen Kernel-Verzeichnisbäume zu einem hohen Reifegrad und einer möglichst geringen Fehlerdichte zu führen.

Die Nachricht, mit welcher das Patch und die README-Datei über die LKML kommuniziert wird, soll möglichst kurze und prägnante Informationen zum Patch, wie z.B. die Version des Kernels oder noch zu klärende Konflikte enthalten. Die Betreffzeile sollte nach [LkmlFaq02] wie folgt aufgebaut sein:

```
[PATCH] <the driver name or piece of code patched>, kernel <kernel version>
```

Der Patch wird entweder direkt der Nachricht beigefügt oder im Falle von größeren Patches über eine URL kommuniziert. Im ersten Fall wird prinzipiell die Formatierung als reiner ASCII-Text verwendet, um eine größtmögliche Zielgruppe erreichen zu können.

Als weitere Adressaten des Patches sollten neben den Mitgliedern der Mailing Liste auch direkt die Maintainer des Quellcodes einbezogen werden, den der Patch verbessert oder erweitert. Dazu ist es nach [Russel00] notwendig, anhand des `MAINTAINERS`- und des `CREDITS`-Files des entsprechenden Quellcodes, den Maintainer bzw. die Autoren des Moduls zu identifizieren. Außerdem sind i.d.R. die Autoren des Quellcodes direkt zu Beginn der entsprechenden Datei vermerkt. Falls keine spezifischen Maintainer für den Quellcode identifizierbar sind, sollte nach [LkmlFaq02] der Patch an die Mailing Liste und ebenfalls an Linus Torvalds bzw. Alan Cox als übergreifende Maintenance Instanz gesendet werden.

#### 2.2.3.7.3 Integration von Patches

Die Entwicklung des Linux Kernels ist durch eine sehr restriktive Integrationspolitik gekennzeichnet, bei der ausschließlich Linus Torvalds dazu berechtigt ist, Quellcode direkt in die existierenden Quellcodearchive zu integrieren: "However, in all questions, Linus Torvalds has the final word as the project maintainer, and he is often referred to as a ‘benevolent dictator’." [HeNiHe02]

Der Projekt Maintainer stellt also im Rahmen der Linux-Kernel Entwicklung die höchste Entscheidungsinstanz für die Integration eines Patches in das aktuelle Entwicklungsverzeichnis dar. Dies bedeutet, dass ausschließlich er über die nötigen Privilegien verfügt, Patches in den aktuellen Verzeichnisbaumes des Kernels zu integrieren. Dies stellt eine sehr restriktive Art der OSS-Entwicklung dar, die aufgrund ihrer zentral ausgeführten Quellcodeintegration als eher OSS-untypisch betrachtet werden muss.

Da aber neben dem primären Entwicklungspfad der Linux Kernel Archive auch verschiedene andere Archive existieren, welche z.B. die dezentrale Integration von Patches in frei zugänglich CVS-Archive zulassen, kann die schnelle und flexible Entwicklung nach den Prinzipien von Open Source auch innerhalb dieser Portale stattfinden. Wie bereits angedeutet wurde, findet zwischen

verschiedenen projektbegleitenden Portalen und Archiven und den zentralen Linux Kernel Archiven ein regelmäßiger Austausch statt, bei dem verschiedene Patches auch in das offizielle Quellcodearchiv übernommen werden.

#### 2.2.3.8 Software-Test

Das Testen der Software wird im Kontext der Linux-Kernel Entwicklung durch die gesamte Community und alle Anwender ausgeführt. Es konnten keine dedizierten Akteure im Rahmen dieser Aktivität identifiziert werden, da diese implizit durch alle Anwender des aktuellen Entwicklungsquellcodes (Pre-Release-Test) bzw. durch alle Anwender der Software-Releases (Post-Release-Test) durchgeführt wird.

##### 2.2.3.8.1 Review und Pre-Release-Test

Nachdem der eigentliche Entwickler sein Patch ersten Tests unterzogen hat, kommuniziert er es an die Entwickler-Community und ermöglicht damit der gesamten Entwicklergemeinde, am Prozess des Testens teilzuhaben. "Finally, review the patch file (the format is not that complicated) before posting, and include all relevant information as to the nature of the patch. In particular, specify: why is this patch needed/useful, and what exactly does it fix/improve." [LkmlFaq02] Vor der Integration eines Patches in den Quellcode haben aber lediglich die Empfänger des initialen Patches Zugang zum Quellcode, wodurch die potentiellen Tester auf die Mitglieder der LKML bzw. die Projekt Maintainer eingeschränkt werden. Diese diskutieren bei entsprechender Relevanz des Patches die Funktionalität und Qualität des Quellcodes und verbessern diesen sukzessive, bis er den Ansprüchen des Maintainers genügt und von diesem in eine offizielle Kernel-Version integriert wird.

Im Rahmen der Verwendung des CVS-Archivs konnte eine modifizierte Variante des Pre-Release-Tests identifiziert werden, die sich in die Phasen des Review und des Pre-Release-Tests unterteilt. Der Review kann auch als Pre-Commit-Test bezeichnet werden und findet durch alle Adressaten vor der eigentlichen Integration des Patches in den Entwicklungsquellcode statt. Die Phase des Pre-Release-Tests wird implizit durch alle Anwender des Entwicklungsquellcodes ausgeführt und geht mit dem Release des Quellcodes in Form eines neuen Software Release in den Post-Release-Test über.

##### 2.2.3.8.2 Post-Release Test

Das Testen von Kernel-Elementen, die bereits in ein offizielles und damit allgemein verfügbares Release integriert wurden, stellt einen impliziten Prozess dar, der durch jeden Anwender während der Benutzung der Software

durchgeführt wird. Im Falle der Identifikation eines Fehlverhaltens führt dies zu dem Teilprozess des Berichtens von Fehlern, der in Kapitel 2.2.3 explizit dargestellt wird. Hierbei ist es für das weitere Vorgehen unerheblich, ob der Fehler bei einer stabilen oder einer Beta-Version des Kernels aufgetreten ist.

Damit wird eine größtmögliche Zielgruppe erreicht, welche die verschiedenen Versionen des Kernels eingehenden und einsatzrelevanten Anwendertests unterzieht. Zudem haben Anwender mit Entwicklerkompetenzen die Möglichkeit, den Quellcode selber auf Korrektheit, Effizienz und sonstige qualitative Kriterien zu überprüfen und gegebenfalls durch ein Patch zu verbessern.

#### 2.2.3.9 Anwender- und Entwicklersupport

Prinzipiell stehen sehr viele verschiedene Informationsquellen zu Verfügung, um sich über Kernelrelevante Themen zu informieren oder Probleme im Zusammenhang mit dem Linux Kernel zu lösen. Neben unzähligen themenbezogenen Webseiten, Mailing Listen, Newsgroups oder Anwenderforen existieren unterschiedlichste Dokumente in Form von "Frequently Asked Questions" oder "HowTo"-Dokumenten.

Es existieren verschiedene Archive über alle Inhalte der Mailing Listen, z.B. unter der folgenden URL: http://hypermail.idiosynkrasia.net/linux-kernel/, welche die Recherche nach bereits vorhandenen Einträgen über das spezifische Thema ermöglichen.

Unzählige linux-spezifische Webseiten, deren URLs teilweise im Anhang dargestellt sind, bieten Informationen zu fast allen erdenklichen Fragestellungen. Weiterhin existieren sekundäre Informationsquellen wie Literatur oder Fachzeitungen mit einem entsprechenden Themenschwerpunkt.

Das prinzipielle Vorgehen der meisten Akteure im Rahmen dieses Teilprozesses gliedert sich in die folgenden Aktivitäten auf:

1. Recherche in existierenden Informationsquellen
2. Kommunikation einer Anfrage an die Community

Der zweite Arbeitsschritt wird dabei i.d.R. nur nach erfolglos durchgeführter Recherche in den existierenden Informationsquellen ausgeführt. Der Kommunikation einer Anfrage über die Kommunikationskanäle schließt sich im optimalen Fall die Antwort eines qualifizierten Akteurs an, der einen Lösungsvorschlag an den Absender persönlich oder an die gesamte Community übermittelt und dazu ebenfalls die verschiedenen Kommunikationskanäle des

Projekts verwendet. I.d.R. werden Anfragen bezüglich bereits hinreichend dokumentierter Problemstellungen von den Adressaten ignoriert.

### 2.2.4 Infrastrukturelle Prozesse

Die infrastrukturellen Prozesse beinhalten die Bereitstellung und Wartung der Werkzeuge und der benoetigten Infrastruktur.

"The Linux developer community, therefore, should not only be viewed as a community that develops the Linux kernel. The success of its development model is critically dependent on resources that it appropriates and develops for the various tasks of the community.... In this sense, the Linux community is not only a Linux developer community, but also a tool-developing community." [Tuomi00]

Die infrastrukturellen Prozesse im Rahmen des Linux-Kernel Projektes werden sowohl vom Maintainer des Projektes als auch von der Entwicklergemeinde ausgeführt, wobei die Koordination und Initiation einzelner Aktivitäten ausschließlich durch den Maintainer erfolgt.

#### 2.2.4.1 Kommunikationsinfrastrukturen

Infrastrukturen, die zur Koordination und zur Kommunikation zwischen den Mitgliedern der Entwicklergemeinde notwendig sind, werden primär durch den Maintainer und einen ihm nahestehenden Kern von Entwicklern geschaffen und gepflegt. Hierzu gehören die Website des Projektes, die LKML und sekundäre Mailing Listen und sonstige Tools zur Kommunikation wie Newsgroups oder IRC-Channels.

Die zentrale Website der Linux Kernel Archive [KernelOrg02] stellt lediglich rudimentäre und grundlegende Funktionalitäten zur Verfügung:

- Kommunikation neuer Kernel-Versionen
- Archivierung von Kernel-Versionen
- Kommunikation von Patches
- Kommunikation der primären Informationsquellen (z.B. LKML)
- Verweis auf weiterführende Informationsquellen

Weiterführende Aufgaben zur Koordination der kollaborativen Entwicklung werden durch diese Plattform nicht erfüllt. Die Tatsache, dass keine konsequent projektweit verwendete Website zur Projektkoordination eingesetzt wird, erklärt [LkmlFaq02] mit der heterogenen Struktur der Entwicklergemeinde: "... because there is no Linux Kernel Team, per se. Also because so many people contributed to the Linux kernel that it would be a tough task to setup and

maintain such a page. Finally, although this is not a rule, most Linux kernel contributors prefer to keep a low profile, for various reasons."

Neben den primären Kommunikationskanälen und -plattformen existieren unzählige weitere Plattformen, die nicht offizieller Bestandteil des Projektes sind, aber von den Entwicklern durchaus als ergänzende Plattform und Informationsquelle akzeptiert werden und deren Aktivitäten z.T. auch mit Linus Torvalds, also dem offiziellen Maintainer des Linux Kernel Projektes, koordiniert werden. Ein Beispiel hierfür wäre z.B. das "Linux Documentation Project" [LDP02].

#### 2.2.4.2 Kollaborationsinfrastrukturen

Da für die kollaborative Entwicklung, z.B. für das Konfigurations- und Versions-Management oder das Bug Tracking, kaum Werkzeuge projektweit etabliert sind, werden diese Tools von anderen Mitgliedern der Community verwaltet, sofern sie im Rahmen des Projektes zum Einsatz kommen.

Als Beispiel wäre hier z.B. das zentrale CVS-Archiv (http://vger.samba.org/) zu nennen, welches verschiedene Versionen des Linux-Kernels in einem CVS-Repository verwaltet und damit das Konfigurationsmanagement und die autarke Integration von Modifikationen durch die Entwickler ermöglicht.

Das zentral verwaltete Bugtracking-Tool Jitterbug (http://samba.anu.edu.au/cgi-bin/linux-patches/) stellt ebenfalls ein projektweit akzeptiertes aber nicht durch den Maintainer initiiertes Werkzeug zur Vereinfachung der kollaborativen Arbeit dar. Jitterbug unterstützt die dezentral organisierte Arbeit, indem es ein zentrales Repository für Bug Reports und die dazugehörigen Patches zur Verfügung stellt. Außerdem definiert es verschiedene Vorgaben und Metadaten, welche zur Verwaltung der Bug Reports verwendet werden und dadurch auch den Patchentwicklungsprozess maßgeblich reglementieren. Auf die verschiedenen Werkzeuge wird an adäquater Stelle noch explizit eingegangen.

#### 2.2.4.3 Etablierung dedizierter Entwicklungspfade

Linux bietet die Möglichkeit der direkten Integration von Modifikationen des Quellcodes und des direkten Hinzufügens neuer Funktionalitäten. "For others, this flexibility was a problem. Continuous change intervened with the translation processes and made it difficult to use Linux as a resource." ([Tuomi00]) Das Resultat dieses Interessenskonflikts war die Aufteilung der Entwicklungspfade durch den Projekt Maintainer in einen stabilen Pfad, in den nur essentielle Änderungen integriert wurden und einen Entwicklungspfad, in den kontinuierlich Änderungen und neue Funktionalitäten integriert wurden.

"Since then, the Linux kernel has been maintained along two parallel paths: a development release containing experimental and relatively untested code, and a stable release containing mostly updates and bug fixes relative to the previous stable release." [GoTu00]

Daraus resultieren die Termini "Production Kernel" und "Experimental Kernel", die in [LkmlFaq02] und [Ebling03] wie folgt definiert werden:

- **Production Kernel**: "Production or stable kernels have a well defined feature set, a low number of known bugs, and tried and proven drivers. ... GNU/Linux distributions are usually based on chosen stable kernel versions, not necessarily the latest production version." [LkmlFaq02]
- **Experimental Kernel**: "The experimental series are fast moving versions which are used to test new features, algorithms, device drivers, etc." [LkmlFaq02] "They serve as test beds for new drivers, experimental features and innovative algorithms." [Ebling03]

Die einzelnen Pfade sind anhand der Versionsnummern zu unterscheiden, da Releases des Entwicklungspfades immer mit ungeraden und Releases des stabilen Pfads mit geraden Versionsnummern markiert sind: "By convention, the middle number in a kernel version identifies to which path it belongs: an odd number (e.g., 1.3.49) denotes a development kernel, and an even number (e.g., 2.0.7) denotes a stable kernel." [GoTu00]

Das offizielle Linux Kernel Archiv [KernelOrg02] bezeichnet die experimentellen Entwicklungsversionen als "Beta-Version" und die getesteten Versionen des Kernels als "Stable Version".

## 2.3 Rollen

Dieser Abschnitt thematisiert die diversen Rollen, welche anhand der definierten Teilprozesse im Rahmen der Kernel-Entwicklung identifiziert werden konnten. Diese Rollen sind i.d.R. nicht explizit im Rahmen des Projektes definiert, sondern wurden implizit identifiziert. Dazu wurden jeweils Teilmengen der Akteure, welche gleiche oder ähnliche Teilprozesse ausführen und ähnliche Eigenschaften besitzen, zu Rollen zusammengefasst.

### 2.3.1 User

Anwender der Software stellen die fundamentale Basis der Projektgemeinde dar und zeichnen sich durch den Prozess des Bezugs der Software, der im Kapitel 2.2.3 explizit erläutert wird, und des späteren Einsatzes der Software aus. Diese passiven Anwender repräsentieren einen großen Anteil der Community. Auch [HeNiHe02] identifiziert 51% der Anwendergemeinde als

passive Mitglieder der Mailing Liste, die somit noch nicht aktiv an der Entwicklung mitgewirkt haben, aber bereits an entwicklungsrelevanten Themen interessiert sind.

### 2.3.2 Contributor

Zur Rolle der Contributoren können alle Akteure zusammengefasst werden, welche bereits aktiv am Entwicklungsprozess teilgenommen haben, indem sie beispielsweise einen Softwarefehler berichtet oder eine Nachricht über einen projektinternen Kommunikationskanal kommuniziert haben. Sobald ein Contributor Quellcode modifiziert und kommuniziert, geht er in die automatisch Rolle des Developers über. Für diese Transition zum Entwickler existieren keine formalen Kriterien, da dies durch den Beitrag von Patches bzw. neuem Quellcode implizit vollzogen wird: "Since the subscription to the mailing list is open to everyone, joining and leaving the project is easy. There is no formal criterion for being a Linux developer apart from the contributions (patches) one submits." [HeNiHe02]

### 2.3.3 Developer

Ein Entwickler hat bereits aktiv Quellcode entwickelt oder modifiziert und in Form von Patches an die Community kommuniziert.

Entwickler verfügen im Rahmen der Kernel-Entwicklung über keine weiteren Privilegien und sind auch nicht berechtigt, Patches direkt und selbständig in die offiziellen Kernel-Versionen zu integrieren. Da von Seiten der primären Maintainer auf ein zentrales CVS-Archiv verzichtet wurde und somit nur die Projekt Maintainer in der Lage sind, Aufgaben des Integrationsmanagements zu übernehmen, ist es dem Entwickler nicht möglich, Änderungen am Quellcode nach ausreichenden Tests direkt in die offiziellen Entwicklungspfade zu integrieren.

### 2.3.4 Committer

Die Rolle des Committers repräsentiert alle Akteure, welche direkt Quellcode in das zentrale Quellcode-Archive integrieren können. Bei der ausschließlichen Betrachtung der zentralen Kernel Archive, konnte diese Rolle nicht identifiziert werden. Da aber noch weitere Plattformen existieren, welche nicht durch die offiziellen Projekt Maintainer verwaltet werden und eine demokratischere Entwicklungsmethodologie basierend auf einem CVS-Repository verfolgen, ist es möglich, im Rahmen dieser Subprojekte kollaborativ an verschiedenen Versionen des Kernels zu arbeiten. Sofern man die projektbegleitenden CVS-Archive als weiteren Entwicklungspfad des Projekts definiert, verfügen die

jeweiligen Entwickler auch über entsprechende Commit-Rechte bezüglich dieses Entwicklungspfades und repräsentieren somit die Rolle des Committers.

### 2.3.5 Modul Maintainer

[HeNiHe02] identifiziert die dedizierte Rolle eines Modul Maintainers, welcher für die Entwicklung eines speziellen Kernel-Subsystems verantwortlich ist: "These participants categorized themselves either as "active Linux kernel developer" or as "maintainer of a module", and were members of a kernel subsystem." Ein Modul Maintainer organisiert alle entwicklungsrelevanten Aspekte bezüglich eines speziellen Moduls weitgehend eigenverantwortlich und arbeitet relativ eng mit dem Maintainer des Projekts zusammen.

### 2.3.6 Projekt Maintainer

Wie bereits im vorangegangenen Abschnitt dargestellt, besitzt Linus Torvalds die oberste Entscheidungsgewalt über alle Aspekte, welche die aktuelle Kernel-Version betreffen. Dies entspricht dem klassischen Modell der OSS-Entwicklung, bei dem der Projekt-Initiator meist auch die weitere Maintenance übernimmt.

Linus Torvalds hat die Verwaltung älterer Kernel-Versionen an verschiedene andere Personen delegiert, welche somit auch in die Kategorie der Projekt Maintainer einbezogen werden müssen.

Die Projekt Maintainer besitzen in diesem Kontext die ausschließliche Entscheidungsgewalt und stellen somit die oberste Projektinstanz dar. Nur sie besitzen die Privilegien, um strategische Entscheidungen, z.B. bezüglich neuer funktionaler Ausrichtungen des Kernels oder sonstiger Systemerweiterungen zu treffen. Auch die operationale Verfügungsgewalt bezüglich offizieller Kernel-Versionen liegt vollständig bei den Maintainern.

Im Rahmen der Linux-Kernel-Entwicklung sind nur die Maintainer dazu berechtigt, Modifikationen des Quellcodes in die offiziellen Entwicklungspfade zu integrieren. Hierbei ist es nicht von Belang, ob die Integration in den stabilen oder den experimentellen Entwicklungspfad erfolgen soll. Auch die Entscheidung über Art, Umfang und Zeitpunkt eines neuen Software-Release kann ausschließlich durch die Maintainer getroffen werden, wodurch im Falle der Kernel-Entwicklung also der Maintainer zugleich die Rolle des Release Managers einnimmt.

## 2.4 Artefakte

"This ecology of resources is a complex one, and a main challenge in becoming a competent Linux developer is to learn how to use these resources. Some of the resources can be characterized as organizational or community resources, others can be viewed as technological artifacts or tools, and as information resources." [Tuomi00]

Im folgenden werden die identifizierten Artefakte dargestellt und eine Klassifizierung basierend auf dem definierten Metamodell vorgenommen.

[Tuomi00] räumt ein, dass ein Artefakt im Laufe des Prozesses verschiedene Funktionen zur Verfügung stellen und damit auch in verschiedenen Entwicklungsphasen zum Einsatz kommen kann: "…one technological artifact can have multiple roles in this ecology." Daher ist es oft nicht möglich, einzelne Artefakte absolut eindeutig zuzuordnen, was zur Folge hat, dass die Artefakte lediglich basierend auf ihrem primären Verwendungszweck klassifiziert werden können.

### 2.4.1 Managementartefakte

In diesem Abschnitt werden alle Artefakte thematisiert, welche nach dem Metamodell als Managementartefakt klassifiziert werden können.

#### 2.4.1.1 Strategische Managementartefakte

Die strategischen Artefakte sind nach dem Metamodell in [Dietze03] von langfristiger Bedeutung für das gesamte Projekt und definieren die Rahmenbedingungen für jegliche Entwicklungsaktivität.

##### 2.4.1.1.1 Lizenz - GNU GPL

"Indeed, open source licenses themselves can be seen as standardized translations which simultaneously provide multifaceted interfaces for many different actors. In very concrete terms, you don't have to negotiate licenses for open source: the license creates a universal standard interface that links the system to potential developers and users. This standardized interface limits the increase in complexity when new communities and actors start to use translated resources in their own activities." [Tuomi00]

Die Software Lizenz, in diesem Falle die GNU General Public License [GnuGpl91] stellt ein zentrales Management-Artefakt dar, da das Lizenzmodell die Rahmenbedingungen für jegliche Entwicklung des Linux-Kernels definiert.

Die Wahl der Lizenz ist somit eine der fundamentalsten strategischen Entscheidungen für ein OSS-Projekt, da die Softwarelizenz alle Restriktionen und Rechte definiert, welche den Einsatz, die Modifikation und die Distribution der Software reglementieren.

#### 2.4.1.1.2 Projektweite Guidelines

Neben der Softwarelizenz können auch die verschiedenen Leitlinien und Konventionen des Projekts als strategische Managementartefakte bezeichnet werden.

##### 2.4.1.1.2.1 Programmierkonventionen

Ein einziges, explizit zu verwendendes Artefakt, welches projektweite Programmierkonventionen definiert, existiert im Rahmen der Kernel Entwicklung nicht. Verschiedene Dokumente thematisieren aber die zu verwendenden Standards, wie [Russel00], der "Linux Kernel Module Programming Guide" [Salzmann01] und das "Kernel-Hacking-HowTo" [Ebling03]. Vor allem [Salzmann01] stellt eine elementare Informationsquelle für projektweite Programmierrichtlinien dar.

Einige fundamentale Konventionen beschreibt [Russel00], wie z.B. Anforderungen an die Portabilität: "You should also minimize CPU specific stuff, e.g. inline assembly should be cleanly encapsulated and minimized to ease porting. Generally it should be restricted to the architecture-dependent part of the kernel tree." Weiterhin beschreibt beispielsweise [Ebling03] die Kernel-Struktur, elementare Datenstrukturen und das empfohlene Vorgehen für die Kernel-Entwicklung. Außerdem beschreiben die Guidelines verschiedene Coding Standards, wie. z.B. Namenskonventionen.

##### 2.4.1.1.2.2 Bug Reporting Guidelines

Die zentralen Prozessdefinitionen und Reglementierungen des Prozesses des Beitragens eines "Bug Reports" werden in "Reporting Bugs for the Linux Kernel" [Gooch02] spezifiziert. Außerdem definiert [Tatham02] verschiedene Vorgaben für das Format des Bug Reports und dessen Kommunikation an die Community. Weitere diesbezügliche Informationen enthalten diverse sekundäre Artefakte, wie z.B. [Salzmann01] oder [Ebling03].

#### 2.4.1.1.2.3 Verhaltensrichtlinien

Die bereits genannten Dokumente thematisieren z.T. auch verschiedene Verhaltensrichtlinien, welche projektweit von der Entwicklergemeinde eingehalten werden sollten und somit ebenfalls einen strategischen Wert für das Projekt besitzen.

Weiterhin wird z.B. auf Eric S. Raymonds Verhaltensmanifest "How To Ask Questions The Smart Way" [Raymond01b] verwiesen, welches einige Leitlinien für die projektweite Kommunikation, wie z.B. die Verwendung von Text-basierten E-Mails oder die präzise, faktenbasierte und möglichst kurze Formulierung von Anfragen beschreibt.

### 2.4.1.2 Operationale Management-artefakte

In diesem Abschnitt werden alle Managementartefakte thematisiert, welche die operative Ausführung der Entwicklungsprozesse kontrollieren, steuern und unterstützen und daher im Sinne des Metamodells als operationale Managementartefakte klassifiziert werden.

**MAINTAINERS-File**

"If you don't know who is who in the community, take a look at the `MAINTAINERS` file in the top level of the kernel source tree." [Ebling03]

Das `MAINTAINERS`-File enthält Informationen über die Autoren des Quellcodes und dient somit der Identifikation von Ansprechpartnern für die Kommunikation von Bug Reports oder Patches. Dadurch ist das `MAINTANERS`-File von großer operationaler Bedeutung und ermöglicht es außerdem, den Autoren die entsprechende Reputation zukommen zu lassen.

## 2.4.2 Technologische Artefakte

In diesem Abschnitt werden alle identifizierten Artefakte thematisiert, die den verschiedenen Phasen der Systementwicklung zugeordnet werden und daher im Kontext des Metamodells als technologische Artefakte betrachtet werden können.

### 2.4.2.1 Anforderungsartefakte

Im Prozess der Linux Kernel Entwicklung findet kein expliziter Teilprozess zur Anforderungsanalyse statt. Die primären Anforderungen an den Kernel wurden

durch den initialen Release des Prototypen durch den Projektinitiator bereits definiert und vorgegeben. Dies bedeutet, dass diese initialen Anforderungen implizit im Quellcode der ersten Kernel-Versionen definiert wurden.

Weitere allgemeine Anforderungen und Vorgaben bezüglich neuer Entwicklungen und Modifikationen sind z.B. in [Ebling03] definiert: "Your code should be 64-bit clean, and endian-independent .... You should also minimize CPU specific code, so inline assembly should be cleanly encapsulated and minimized (hence put in the architecture specific parts of the source tree) to ease porting." Neben diesen Anforderungen an die Portabilität des Quellcodes, werden verschiedene weitere generelle Anforderungen in [Ebling03] und [Salzmann01] definiert, da diese Dokumente Richtlinien für die Linux-Kernel-Entwicklung enthalten.

Im weiteren Verlauf des Entwicklungsprozesses werden durch Mitglieder der Community neue Anforderungen formuliert, indem diese z.B. eine Anfrage für eine Funktionalitätserweiterung an die LKML senden, einen Bug Report formulieren oder eine funktionale Erweiterung direkt in Form eines Patches an die LKML kommunizieren. Somit sind also alle Artefakte, welche entsprechende Infomationen enthalten können, als Anforderungsartefakte in Betracht zu ziehen:

- Patch
- Bug Report
- Feature Request

Als Container für die Kommunikation dieser Informationen können die Elemente der gebäuchlichen Kommunikations- und Kollaborationswerkzeuge betrachtet werden, welche im entsprechenden Abschnitt näher erläutert werden:

- Mailing List Message
- Newsgroup Message
- Persönliche E-Mail
- IRC-Channel-Eintrag
- JitterBug-Eintrag

**Feature Request**

Die Anfrage einer funktionalen Systemerweiterungen durch einen Anwender bzw. Entwickler wird im folgenden als "Feature Request" bezeichnet. Ein "Feature Request" kann über alle gebräuchlichen Kommunikationswege des Projektes kommuniziert werden und stellt das einzige, primär zur Anforderungsspezifikation verwendete Artefakt dar.

Der Lebenszyklus des "Feature Requests" wird im entsprechenden Abschnitt der Prozessbeschreibung (Kapitel 2.2.3) näher dargestellt und weist sehr viele Ähnlichkeiten zu dem des "Bug Reports" auf.

#### 2.4.2.2 Entwurfsartefakte

"Linux has neither a documented conceptual nor a documented concrete architecture....There is no architectural documentation that describes the system structure at a high level of abstraction." [BoHoBr99]

Der Teilprozess des Softwareentwurfs basiert im Kontext der Linux Kernel Entwicklung auf keinen dedizierten Aktivitäten, sondern wird i.d.R. implizit während der Implementation ausgeführt. Dies bedeutet, dass alle Dokumente, welche die Softwareanforderungen bzw. deren Implementation thematisieren, und vor allem die entwickelten Patches, implizit auch das Softwaredesign definieren. Ergänzt werden diese Informationen durch etwaige Dokumentationen der Patches, z.B. basierend auf `README`-Dateien. [Tuomi00]

#### 2.4.2.3 Implementationsartefakte

Dieser Abschnitt beschreibt die Artefakte, welche primär in der Phase der Implementierung generiert bzw. verwendet werden.

##### 2.4.2.3.1 Patch

"A patch file (as it refers to the Linux kernel) is an ASCII text file that contains the differences between the original code and the new code, plus some additional information such as filenames and line numbers." [LkmlFaq02]

Wie bereits in den Prozessbeschreibungen dargestellt wurde, werden Quellcodemodifikationen in Form von Patches kommuniziert, die i.d.R. aus genau einer Quellcodedatei bestehen: "Several lines of code, in turn, form a "patch" which is an atomic contribution to a software project, submitted as one file and addressing one single issue." [HeNiHe02]

Der Quellcode setzt sich im Rahmen des Linux-Kernels aus C-Dateien oder den entsprechenden Header-Dateien zusammen: "We use the term "source file" to mean any file whose name ends with ".c" or ".h" ... " [GoTu00]. Patches in Form von Quellcode stellen in diesem Zusammenhang die elementarsten bzw. atomaren Implementationsartefakte dar und spezifizieren implizit auch die Softwareanforderungen und das Softwaredesign.

#### 2.4.2.3.2 Dokumentationsartefakte

"Linux information and technical support is available from a wide variety of locations. There are the "official" routes such as the Linux Software Map, Linux Documentation Project, HOWTOs, and FAQs (Frequently Asked Questions). There are dozens of published reference materials, from books and print journals to electronic "ezines" available by email and/or various web sites.

In additon, there is a wealth of information that is available from other users -- from personal web sites to numerous Linux-oriented Usenet (news) Groups, mailing lists and Internet Relay Chat (IRC) channels dedicated to the discussion of Linux." [LinuxOrg02][21]

Dokumentation kann in die beiden Kategorien entwicklungsrelevante und anwendungsbezogene Dokumentation eingeteilt werden. Zentrales Informationsportal sowohl für technische, als auch für anwendungsbezogene Dokumentation stellt hierbei das Linux Documentation Project[22] dar, welches auf das Gros der Linux- und Linux-Kernel-bezogenen Dokumentationen verweist: "The Linux Documentation Project (LDP) is working on developing good, reliable documentation for the Linux operating system. The overall goal of the LDP is to collaborate in taking care of all of the issues of Linux documentation, ranging from online documentation (man pages, HTML, and so on) to printed manuals covering topics such as installing, using, and running Linux." [LDP02]. LDP verweist auf verschiedene HowTos, FAQs, Manuals und ähnliche Artefakte und bezieht sich sowohl auf entwicklungsrelevante als auch auf anwendungsbezogene Aspekte.

##### 2.4.2.3.2.1 Anwendungsbezogene Dokumentation

Dokumentation, die sich primär an die Anwender der Software richtet, ist in diversen Informationsquellen zu finden, die projektweit durch sehr heterogen verwaltete Werkzeuge zur Verfügung gestellt werden. Neben den verschiedenen, direkt auf einschlägigen Websites und den zentralen Linux Kernel Archiven publizierten Informationen existieren verschiedene Formate, welche für die anwendungsbezogene Dokumentation verwendet werden:

[LinuxOrg02][21] definiert u.a. die folgenden Ausprägungen von Dokumentationsartefakten:

- HOWTOs
- Mini-HOWTOs

21 http://www.linux.org/docs/index.html
22 http://www.ldp.org

- Project Guidelines
- FAQs
- Linux Man Pages (download via FTP)
- Unmaintained HOWTOs
- Unmaintained MINI-HOWTOs

"FAQs" repräsentieren hierbei sowohl allgemeine, als auch spezielle Frequently Asekd Questions, die zentral zur Verfügung gestellt werden. HOWTO-Dokumente beschreiben i.d.R. einen speziellen Anwendungsfall im Rahmen der Anwendung oder Entwicklung und stellen somit ein essentielles Hilfsmittel dar.

Manpages repräsentieren ebenfalls eine weit verbreitete Möglichkeit zur Dokumentation und beinhalten, programm- bzw. modulspezifische Informationen, welche als formatierter ASCII-Text ausgegeben werden können. Sie beinhalten verschiedene grundlegende Informationen über die Syntax des Programmaufrufs, dessen Beschreibung, die Optionen bei der Ausführung, eine Beschreibung der verwendeten Umgebungsvariablen und eine Liste der Autoren.

Zur Kommunikation und Publikation dieser Dokumentationsformate kommen i.d.R. die explizit im Kapitel 2.5.1 beschriebenen Kommunikationswerkzeuge zum Einsatz. Außerdem enthalten diese projektweit eingesetzten Kommunikationswerkzeuge implizit weitere anwendungsbezogene Informationen, welche auch über einen längeren Zeitraum hinweg, z.B. über Archive, zur Verfügung stehen.

#### 2.4.2.3.2.2 Entwicklungsrelevante Dokumentation

Neben diesen allgemeinen Informationsartefakten, die sich sowohl an Anwender, als auch an Entwickler richten, existieren verschiedene Artefakte, welche explizit nur für die Entwickler von Bedeutung sind.

Entwicklungsrelevante Dokumentation kann i.d.R. direkt mit einem spezifischen Modul, Patch oder einer Quellcodedatei assoziiert werden und wird daher häufig direkt in Verbindung mit diesem speziellen Objekt formuliert.

Die Dokumentation des Quellcodes erfolgt z.T. implizit in den Quellcodedateien in Form von Kommentaren, wobei [Godfrey00b] zu dem Schluß kommt, dass der Kernel-Quellcode über einen Kommentar-Anteil von ca. 28-30% verfügt. Dokumentation in dieser Form richtet sich primär an die Entwickler und nicht an die Anwender des Kernels. Weitere modul- oder patchspezifische Dokumentation ist oft in Artefakten zu finden, die direkt mit dem betreffenden Quellcode assoziiert werden, wie z.B. die Mailing List- oder Newsgroup-Messages, über welche Patches kommuniziert und diskutiert wurden.

Eine weitere wichtige Informationsquelle für die Entwickler der Software stellen die `MAINTAINERS`- und `CREDITS`-Files dar, welche modulbezogene Informationen über die Autoren und Maintainer enthalten.

#### 2.4.2.3.2.3 Projektexterne Dokumentationsartefakte

Weiterhin existieren neben diesen, projektintern verwalteten Dokumentationsartefakten verschiedene Informationsobjekte, die ebenfalls eine zentrale Informationsquelle für Akteure im Linux Kernel Kontext darstellen.

Diese können nach [LinuxOrg02][23] wie folgt eingeteilt werden:

- Books
- Videos, Newsletters, Magazines, and Ezines
- Non-Commercial Support Web Sites

[LinuxOrg02][24] führt eine Liste verschiedener, nicht projektgesteuerter Informationsquellen an, die u.a. verschiedene Webseiten, Magazine u.ä. umfasst. Aufgrund der nicht projektinternen Erstellung und Verwaltung dieser Artefakte, werden diese in den weiteren Betrachtungen nicht explizit berücksichtigt.

### 2.4.2.4 Einsatzartefakte

Diesem Abschnitt werden ausschließlich Artefakte zugeordnet, die primär im Kontext der Anwendung der Software von Bedeutung sind.

#### 2.4.2.4.1 Bug Report

Der bereits in der Prozessbeschreibung vorgestellte "Bug Report" stellt ein weiteres elementares Artefakt in diesem Kontext dar, da die Linux-Kernel-Entwicklung auch als Prozess der sukzessiven Fehlerbeseitigung und Systemverbesserung betrachtet werden kann [Tuomi00]. Die Generierung von Patches und sonstigen Quellcodeartefakten basiert somit i.d.R. auf den Anforderungsspezifikationen der "Bug Reports" und "Feature Requests", welche somit einen integralen Bestandteil der Entwicklung darstellen.

23 http://www.linux.org/docs/index.html
24 http://www.linux.org/docs/reference.html

Bug Reports können über alle gebräuchlichen Kommunikationswege (siehe Kapitel 2.5.1) kommuniziert werden und können als zusätzliches Transportmedium über ein zentral verwaltetes Tool zum Bugtracking, Jitterbug, kommuniziert und bearbeitet werden, welches ebenfalls im Kapitel 2.5.1 näher dargestellt wird. Entsprechend dem Jitterbug-System kann ein Bug Report im Rahmen seines Lebenszyklus die folgenden Stati durchlaufen:

- **Incoming**: Ausgangsstatus aller Bug Reports, die neu generiert werden
- **Discussion**: Bug Reports, die momentan durch die Community validiert werden
- **Highpri**: Bug Reports, die noch nicht bearbeitet werden, aber als äußerst wichtig bzw. kritisch eingestuft werden und daher mit einer hohen Priorität bearbeitet werden sollten
- **Discarded**: Bug Reports, die nicht als relevanter Softwarefehler verifiziert und reproduziert werden konnten
- **Deferred**: Softwarefehler, deren Bearbeitung auf einen späteren Zeitpunkt verschoben wurde
- **Pending**: Softwarefehler, die bereits durch Entwickler bearbeitet werden
- **Done**: Softwarefehler, die bereits durch ein Patch behoben wurden

Auch bei der ausschließlichen Kommunikation über klassische Kommunikationskanäle durchläuft ein Bug Report implizit vergleichbare Stati, welche aber nicht explizit über Metadaten dokumentiert werden.

#### 2.4.2.4.2 Binärcode

Basierend auf dem Implementationsartefakt "Patch", also dem Quellcode einer Softwaremodifikation, lässt sich durch Kompilation Binärcode erzeugen, der somit ebenfalls als ein Einsatzartefakt betrachtet werden kann.

Die kleinsten Binärcode-Artefakte sind in diesem Zusammenhang einzelne ausführbare Dateien. Diese können wiederum entweder einem Software Release zugeordnet werden oder basieren auf der Kompilation des aktuellen Entwicklungsquellcodes. Daher kann die folgende Klassifizierung vorgenommen werden:

- Release Build
- Development Build

## 2.5 Werkzeuge

Neben verschiedenen, von der Entwicklergemeinde entwickelten Werkzeugen wird der Entwicklungsprozess auch durch externe und i.d.R. frei verfügbare Tools unterstützt: "At the same time resources that are produced outside the focal community are appropriated by the community. For example, the c-

language compiler produced by the GNU gcc community is generated outside the Linux development community." [Tuomi00]. Fundamentale Basis für die projektweite Nutzung von externen aber freien Hilfsmitteln, stellt die GNU General Public License [GnuGpl91] dar, welche die Nutzung des Werkzeugs langfristig sichert und damit eine strategische Grundlage für das Projekt bildet: "The GNU General Public License obviously plays an important role here. It guarantees that the GNU gcc compiler can be appropriated by the Linux community as a core resource in the development. By relying on the institution of copyright, open source licenses provide an institutional basis for reducing risk and building knowledge based trust .... Without open source license, it would be very risky to build a system that so critically depends on a resource that is produced outside the community." [Tuomi00]

### 2.5.1 Kommunikationswerkzeuge

Wie bereits dargestellt wurde, existieren neben dem primären Kommunikationsmedium, der Website der Linux Kernel Archive[25], viele verschiedene Kommunikationswerkzeuge, welche in [LinuxOrg02][26] wie folgt dargestellt werden:

- Usenet News Groups
- User Groups and Organizations
- Mailing Lists
- Internet Relay Chat (IRC)
- Non-Commercial Support Web Sites

#### 2.5.1.1 Projekt Website

Primäre Plattform für die Kommunikation und Kollaboration im Rahmen des Projektes stellt die Website des Projektes dar. Als offizielle Projekt-Website sind die Linux Kernel Archive[25] anzuführen, wobei verschiedene weitere Plattformen existieren, welche nicht direkt durch die Maintainer des Projektes gewartet werden, aber für die Community trotzdem eine relevante Plattform zur Kollaboration darstellen.

#### 2.5.1.2 Mailing Lists

Zentrale und offizielle Mailing List des Linux Kernel Projektes ist die Linux Kernel Mailing List[27] (LKML), welche zur Kommunikation aller technischer und

25 http://www.kernel.org
26 http://www.linux.org/docs/index.html

organisatorischer Aspekte verwendet wird. Neben dieser primären Mailing Liste existieren verschiedene untergeordnete Mailing Listen, welche als adäquates Medium für die Diskussion von spezifischen Problemstellungen dienen: "Besides the Linux kernel mailing list, there are a couple of mailing lists dealing with more specific technical aspects of the kernel development, such as the development of certain filesystem or hardware drivers. These fields are also called "subsystems" of the kernel." [HeNiHe02]

Die LKML ist das präferierte Kommunikationswerkzeug im Rahmen der Linux Kernel Entwicklung und stellt die Basis für die Diskussion aller Kernel-relevanten Aspekte dar. Sie wird i.d.R. den meisten anderen Kommunikationswerkzeugen vorgezogen.

#### 2.5.1.3 Usenet Newsgroups

"Newsgroups are an excellent resource for new and experienced Linux users. A wide variety of groups ... offer a wealth of information on general and specific topics of discussion." [LinuxOrg02]

[LinuxOrg02] stellt eine Liste der gebräuchlichsten Newsgroups zur Verfügung und definiert die Newsgroup comp.os.linux.misc als adäquate allgemeine Alternative, wenn die probate Newsgroup nicht identifiziert werden kann: "When in doubt, the comp.os.linux.misc group is a good choice." Als zentrale Newsgroup für die Kernel-Entwicklung konnte comp.os.linux.development.system identifiziert werden, deren Inhalte von [LinuxOrg02] wie folgt kategorisiert wurden: "Discussion specific to the Linux kernel, device drivers, and loadable modules."

Weiterhin existieren eine Vielzahl von Sub-Newsgroups, die jeweils verschiedene Teilaspekte der Kernel-Entwicklung und -Anwendung thematisieren oder sich an bestimmte, z.B. regional begrenzte, Zielgruppen richten.

Zur Recherche über die verschiedenen Inhalte verschiedener Newsgroups über einen längeren Zeitraum, um z.B. die Novität eines Software-Fehlers zu verifizieren, stehen verschiedene Archive zur Verfügung, welche die historischen Inhalte der Newsgroups zur Verfügung stellen: "An archive of comp.os.linux.announce is available at ftp://sunsite.unc.edu/pub/Linux/docs/linux-announce.archive. An archive of all Usenet groups is available at http://www.dejanews.com." [LinuxOrg02]

27 linux-kernel@vger.kernel.org

#### 2.5.1.4 Internet Relay Chat

"Internet Relay Chat (IRC) is a method of communicating with other people around the world in a real-time fashion. There are dozens of IRC networks, many of which have channels devoted to Linux…" ([LinuxOrg02]) Durch die Technologie des IRC werden die Möglichkeiten der Echtzeitkommunikation für das Linux Kernel Projekt instrumentalisiert. Außerdem bietet das IRC die Möglichkeit zur zielgerichteten, bidirektionalen Kommunikation oder zur Kommunikation mit einigen, selbständig durch den Initiator eines privaten Kanals bestimmbaren Akteuren.

IRC wird im Rahmen der Kernel-Entwicklung vor allem für die Diskussion technischer Aspekte verwendet und dient nur peripher dem Anwendersupport: "....many channels are not support-oriented and are instead focused on highly technical discussions." ([LinuxOrg02]) Es konnte in diesem Kontext kein explizites, offiziell empfohlener IRC-Network identifiziert werden, welcher primär verwendet wird, zumal der Hauptanteil der Kommunikation auf der LKML beruht. [LinuxOrg02] stellt aber eine Liste vieler Linux-relevanter IRC-Networks zur Verfügung.

#### 2.5.1.5 Nichtkommerzielle Support-Websites

Neben der zentralen Projektplattform, den Linux Kernel Archiven, existiert eine Vielzahl von projektbegleitenden Webseiten, welche auf sehr diverse Art und Weise Anwender- und Entwicklersupport zur Verfügung stellen und z.B. die folgenden Leistungen zur Verfügung stellen: "...discussion group postings, question/answer formats, and individualized support when resources are available." [LinuxOrg02] Diese sekundären Plattformen stellen für die Linux-Gemeinde eine bedeutende Ressource dar, indem sie durch Webforen oder ähnliche Kommunikationskanäle weitere Möglichkeiten zur projektbezogenen Kommunikation zur Verfügung stellen.

Weiterhin weist [LinuxOrg02] auf die nichtkommerzielle Natur dieser Plattformen und die damit verbundene Freiwilligkeit der Dienstleistung hin, indem eingeräumt wird, "that free support is simply that -- a group of devoted Linux users willing to help others. They may not always have all the answers, and the turnaround time may be longer than you wish -- many people in the Linux community are contributing their time and knowledge in addition to full-time jobs and responsibilities unrelated to Linux support."

Da diese Webseiten keinen offiziellen Bestandteil der Projektinfrastruktur darstellen und damit auch nicht strategisch in den Planungshorizont und den Entwicklungsprozess einbezogen werden können, werden diese projektbegleitenden Plattformen im Folgenden nicht näher evaluiert.

### 2.5.2 Kollaborationswerkzeuge

Auch die Werkzeuge zur Unterstützung der kollaborativen Entwicklung basieren ausschließlich auf webbasierten oder zumindest webkompatiblen Tools: "A special characteristic of this collaboration has been that it has almost completely relied on Internet-based tools." [Tuomi00]

Aufgrund der sehr speziellen Charakteristika des Linux Kernel-Projekts ist die Definition von einigen, extern verwalteten Ressourcen als integraler Bestandteil der Kernel-Entwicklung durchaus zulässig, da der Entwicklungsprozess des Linux Kernels ohne verschiedene externe Ressourcen, wie z.B. CVS-Archive oder Bug Tracking-Systeme, welche nicht durch das zentrale Management bereitgestellt werden, nicht in der momentan praktizierten Form möglich wäre.

#### 2.5.2.1 Bug Tracking - Jitterbug

Jitterbug ist ein webbasiertes Bug-Tracking System, welches es ermöglicht Reports über Software-Fehler zu verwalten, zu bearbeiten und zu archivieren und damit die kollaborative Bearbeitung von "Bug Reports" unterstützt: "… the JitterBug system is a web-based database which shows what bugs are known to the community and whether someone is doing something to correct a bug. JitterBug acts as an information resource by allowing people to find out what bugs are known, and as a community resource by coordinating the work needed to solve the problem." ([Tuomi00]) Jitterbug ist kein offiziell verwaltetes Werkzeug im Rahmen der Kernel-Entwicklung, da "Bug Reports" vorzugsweise über die LKML kommuniziert werden, stellt aber trotzdem eine wichtige Ressource für die Gemeinde der Linux-Kernel-Entwickler dar.

Eine häufig verwendete Instanz eines Jitterbug-Systems steht unter [SambaOrg02][28] zur Verfügung. Hier werden "Bug Reports" eingepflegt, verwaltet und bearbeitet und auch die entsprechenden Patches, im ASCII-Format und als Bestandteil eines Datenbankeintrages, veröffentlicht und diskutiert. "JitterBug operates by receiving bug reports via email or a web form. Authenticated users can then reply to the message, move it between different categories or add notes to it. In some ways JitterBug is like a communal web based email system." [SambaOrg02][29]

Weiterhin beschreibt die Website [SambaOrg02][30] das Jitterbug-System mit den folgenden Spezifikationen:

- Vollständig webbasiert (C-Applikation als CGI-Script)

28 http://samba.anu.edu.au/cgi-bin/linux-patches/
29 http://samba.anu.edu.au/cgi-bin/jitterbug

- HTML 3.2-kompatibel (keine Frames, Java o.ä.)
- Nutzer-Authentisierung
- E-Mail-Integration möglich: "Each user can request email notification of changes to messages or directories"
- Verschiedene Metadaten zu jedem Bug Report konfigurierbar
- Nachverfolgung aller Aktivitäten möglich (Logfiles für jeden Bug Report)

Diese Merkmale unterstützen somit vor allem die Anforderung der OSS-Entwicklung, nach einer möglichst großen Reichweite und hohen Kompatibilität, um möglichst alle Akteure der sehr heterogenen Community erreichen zu können. [Vepstas02] beschreibt außerdem, dass im Rahmen des Samba-Projekts bereits über 2500 Bug Reports im System verwaltet werden.

Jitterbug wird unter der GNU GPL veröffentlicht und ermöglicht damit die langfristige, strategische Verwendung des Systems als integraler Bestandteil der Projektinfrastruktur: "It is now available under the GNU General Public License in the hope that it will be useful for other projects." [SambaOrg02][30]

#### 2.5.2.2 Quellcodeverwaltung und Konfigurationsmanagement

"Any development work is an incremental process, particularly so at the debugging stage. It always pays to keep a record of the process using some form of revision control system, so that a bad change can backed out with minimal effort." [Ebling03]

Von offizieller Seite der Projekt Maintainer wird momentan darauf verzichtet, allen Entwicklern den Zugriff auf den Quellcode über ein zentrales Werkzeug für das Konfigurationsmanagement zur Verfügung zu stellen. Das Konfigurationsmanagement inkl. der Verwaltung der verschiedenen Kernel-Versionen und der Integration von Patches erfolgt somit ausschließlich durch die Projekt Maintainer. Es ist aber davon auszugehen, dass diese für die interne Verwaltung des Quellcodes auf ein entsprechendes System zugreifen: "bitkeeper is a powerful, scalable, reproducible and distributed source management system. It is used by some of the high profile kernel hackers (including Linus himself) to try and bring order to a some-what hap-hazard development process." [Ebling03]

Trotzdem existiert ein zentrales CVS-Archiv, welches allen Entwicklern zur Verfügung steht und auf einer inoffiziellen Plattform bereitgestellt wurde. Der Quellcode dieses Archivs wird regelmäßig mit den offiziellen Linux Kernel Archiven koordiniert.

30 http://samba.anu.edu.au/cgi-bin/jitterbug

Weiterhin verwenden individuelle Entwickler und Entwickler-Teams verschiedene Werkzeuge um ihren Quellcode dezentral zu verwalten und die kollaborative Arbeit daran zu ermöglichen.

#### 2.5.2.2.1 RCS - Revision Control System

RCS ermöglicht die Verwaltung und Archivierung von Quellcode und stellt damit ein elementares Werkzeug für die Kernel-Entwicklung dar, welches von vielen Entwicklern und Entwicklergruppen verwendet wird. Damit wird versucht, die Abstinenz eines derartigen zentral organisierten Werkzeugs durch die dezentrale Verwendung bei der lokalen Bearbeitung zu kompensieren.

#### 2.5.2.2.2 CVS - Controlled Versioning System

"While RCS is good for basic projects involving changes to a limited number of files, it quickly becomes unwieldy for larger projects, particularly so with contributions from more than one developer. CVS is particularly good in this area." [Ebling03]

Wie bereits in der Prozessbeschreibung dargestellt wurde, existiert unter der URL: http://vger.samba.org/ ein zentrales CVS-Archiv welches nicht durch die Projekt Maintainer instanziiert wurde, aber dessen Inhalte häufig mit den Inhalten der Linux Kernel Archive abgeglichen werden. Daher repräsentiert dieses System auch die am häufigsten verwendete Ressource zur zentralen Quellcodeverwaltung und stellt neben den offiziellen Kernel-Archiven die primärer Quellcodebasis dar.

#### 2.5.2.2.3 BitKeeper

"Some kernel developers ... have chosen to use BitKeeper to manage their kernel source trees…" [LkmlFaq02]
BitKeeper ist ein Werkzeug zur Verwaltung von Quellcodeverzeichnissen, welches auch die kollaborative Bearbeitung des Quellcodes durch mehrere Entwickler unterstützt: "BitKeeper is a source management system that enables developers to work concurrently on the same project. … BitKeeper was designed to be a scalable system that supports globally distributed development, disconnected operation, change sets, and named lines of development (branches)." [BitkprCom02]

Trotzdem stellt BitKeeper kein projektweit verwendetes Kollaborations-Werkzeug dar, sondern kann nur als individuell verwendetes Tool zur Unterstützung einzelner Entwickler und Entwicklerteams kategorisiert werden, da es lediglich sporadisch zum Einsatz kommt: "Many notable kernel

developers continue to maintain their source trees using other tools and techniques, and continue to send conventional patches." [LkmlFaq02]

### 2.5.3 Entwicklungswerkzeuge

Für die individuelle Bearbeitung des Quellcodes auf Seiten der Entwickler kommt eine sehr diverse Infrastruktur zum Einsatz, da diese Kategorie der Werkzeuge nicht zentral, sondern individuell durch jeden Entwickler selektiert, verwaltet und verwendet werden. Die bereits definierten Projekt-Guides treffen hierfür lediglich verschiedene Empfehlungen, die aber für die Entwickler nicht bindend sind. Dies führt zu einer sehr heterogenen Struktur der Entwicklungswerkzeuge, wobei sich dieser Abschnitt nur auf die primär verwendeten beziehen wird, die nach [Tuomi00] u.a. aus den folgenden Tools bestehen:

- Editoren (vim, emacs)
- gnu gcc
- make
- gdb
- ksymoops
- diff

Neben diesen Entwicklungswerkzeugen konnten noch andere, individuell durch die Entwickler verwendeten Werkzeuge identifiziert werden, die in den folgenden Abschnitten dargestellt werden.

#### 2.5.3.1 Editoren

"There is a wide choice when it comes to choosing an editor. Many heated debates have taken place over which editor(s) is best suited to (kernel) programming." [Ebling03] Primär kommen in diesem Zusammenhang die Editoren-Werkzeuge vim und emacs zum Einsatz. Beide Werkzeuge weisen verschiedene Vor- und Nachteile auf, die an dieser Stelle nicht weiter thematisiert werden sollen.

#### 2.5.3.2 Make

Das Werkzeug Make wird ebenfalls sehr häufig im Rahmen der Kernel-Entwicklung eingesetzt und ermöglicht es, die Dateien eines Quellcodeverezeichnisses zu identifizieren, welche nach der letzten Kompilation modifiziert wurden: "make is a program that is used to determine which parts of a multi-source-file project need to be rebuilt. Only those source files which

have been modified are rebuilt before the final linking step." [Ebling03] Dadurch kann der Kompilationsaufwand auf ein Minimum beschränkt werden.

#### 2.5.3.3 Diff

"To make a patch you use the diff program…" [LkmlFaq02] Wie bereits in der Beschreibung des Teilprozesses "Generierung von Patches" im Kapitel 2.2.3 dargestellt wurde, stellt `diff` ein elementares Werkzeug zur einfachen Generierung von Patches dar, indem es dem Entwickler erlaubt, aus zwei Quellcode-Verzeichnissen die Teilmenge zu extrahieren, die sich nicht in der gemeinsamen Schnittmenge des Quellcodes befindet. Dadurch ist es möglich, ein Patch basierend auf dem Entwicklungsverzeichnisbaum und dem originalen Verzeichnisbaum der zugrundeliegenden Kernel-Version zu extrahieren. `diff` ist Bestandteil des Kernels und daher frei verfügbar.

#### 2.5.3.4 Compiler - GCC

Nach [LkmlFaq02] wurden durch den Maintainer folgende Compiler empfohlen: "On this date, Linus announced that gcc 2.91.66 (aka egcs 1.1.2) is the recommended compiler for 2.4.x kernels up to version 2.4.9. Gcc 2.95.3 is the recommended compiler for kernel 2.4.10 and later."

Weiterhin wird die Verwendung des empfohlenen Compilers als wichtige Bedingung nahegelegt: "... the recommended compiler has been extensively tested and proven to be a very stable compiler. […] Current kernels and compilers are very complex pieces of software. There are just too many ways that the two can interact and cause trouble (a recent example: gcc 2.8.x and kernel 2.0.x). By keeping constant one of the variables - the compiler - kernel developers can concentrate on the kernel." [LkmlFaq02]

Damit stellt der GCC-Compiler eine Ressource von strategischer Bedeutung für die Kernel-Entwicklung dar, deren Verwendung durch die Lizensierung unter der GNU GPL langfristig gesichert ist: "The gcc compiler is, however, a critical resource for the Linux community..." [Tuomi00]

#### 2.5.3.5 Lclint

Lclint kann entsprechend den individuellen Gewohnheiten der Entwickler dazu verwendet werden, häufige Fehler im Quellcode zu entdecken und findet zum Teil Anwendung in der Kernel-Entwicklung: "lclint is a program that can be used to statically check C code, that is to check it before it has even been compiled or executed. Like lint, lclint can be run on unmodified C source code and used to catch classic programming mistakes." [Ebling03]

#### 2.5.3.6 Quellcodenavigation

Zur Navigation durch existierenden Quellcode stehen verschiedene Werkzeuge zur Verfügung, wobei die Entwicklergemeinde i.d.R. auf die im folgenden dargestellten zurückgreift.

##### 2.5.3.6.1 Grep, cscope, ctags

Grep stellt ein Werkzeug zur Suche und Recherche in Quellcodedateien dar und findet auch häufig in der Kernel-Entwicklung Anwendung: "grep is used to find lines matching a given pattern; a useful tool for quickly locating variable or function definitions and use." [Ebling03]

Cscope verfolgt eine ähnliche Zielsetzung wie grep, aber bietet etwas umfassendere Funktionalitäten und eine komfortablere Nutzerschnittstelle.

Ctags verfügt über eine ähnliche Funktionalität und ermöglicht zusätzlich eine bessere Integration mit dem verwendeten Editoren-Werkzeug: "ctags provides similar functionality to cscope, but integrates closely with your editor, allowing look-ups with a few key strokes." [Ebling03]

##### 2.5.3.6.2 Lxr

"lxr enables you to navigate source code via a web browser, where all variables and functions are links to their respective definitions." [Ebling03] Damit stellt lxr eine sehr komfortable Möglichkeit dar, durch zentral oder lokal verwalteten Quellcode zu navigieren und ermöglicht auch die Verlinkung von logisch zusammenhängenden Statements.

#### 2.5.3.7 Ksymoops

Für die Decodierung und Erfassung eines "Oops", der bereits in der Darstellung der Teilprozesse erläutert wurde, wird häufig das Tool ksymoops verwendet. Die Ausgaben dieses Werkzeuges dienen somit der Generierung von notwendigen Informationen für das Artefakt "Bug Report", wie auch in [Kernel02] und [LkmlFaq02] vermerkt ist.

# 3 Fallstudie des OSS-Projekts Apache

Die folgenden Fallstudien analysieren primär die Entwicklungsprozesse im Apache HTTPD-Projekt und berücksichtigen aber zum Teil auch Erkenntnisse aus weiteren Apache-Teilprojekten. Da das HTTPD-Projekt das erste Entwicklungsprojekt im Apache-Kontext repräsentiert, erscheint die ergänzende Betrachtung später initialisierter Entwicklungsprojekte (Apache-Jakarta) sehr sinnvoll, da diese bereits Erfahrungen aus dem Projektverlauf des HTTPD-Projekts berücksichtigen und somit einen weiteren Schritt in der Evolution der Entwicklungsprozesse darstellen. So wurden häufig auftretende Probleme im HTTPD-Kontext bereits durch eine Optimierung der Prozesse und Infrastrukturen in späteren Projekten vermieden: "These types of problems have allowed Jakarta to flourish into a great example of open source software development. Therefore, analyzing this project provides us with valuable open source techniques, which may allow us to form a general open source software development meta-model." [AtGaGe02]

## 3.1 Apache Projektüberblick

"The Apache group was formed in February of 1995 by the 8 core founders." [AtGaGe02] Diese Organisation wurde mit dem Ziel gegründet, die existierenden Weiterentwicklungen eines bereits existierenden Prototypen, des HTTPD-Programms des National Center for Supercomputing Applications (NCSA) der University of Illinois, zu koordinieren und zu integrieren: "Apache began in February 1995 as a combined effort to coordinate existing fixes to the NCSA httpd program developed by Rob McCool." [MoFiHe00]. Die existierende Softwarearchitektur wurde mit dem Ziel der gesteigerten Stabilität und Fehlerfreiheit vollständig überarbeitet und im Juli 1995 in Form einer komplett neuen Quellcode-Basis implementiert. Die bestehenden Funktionalitäten und verschiedene Systemerweiterungen wurden in die neue Architektur integriert und in einer ersten Testphase veröffentlicht, mit der Konsequenz, dass das erste offizielle Release des Apache-HTTPD Projektes im Januar 1996 veröffentlicht werden konnte: "The Apache Group then ported all existing features, and many new ones, to the new architecture and made it available for beta test sites, eventually leading to the formal release of Apache httpd 1.0 in January 1996." [MoFiHe00]. Einige Aspekte der initialen Veröffentlichung und Entwicklung werden im Abschnitt "Initiale Prototypentwicklung und -release" noch explizit dargestellt.

Im Jahr 2000 ging aus der ursprünglichen Apache Group die Apache Software Foundation[31] hervor, welche die Aufgaben der Projektwartung, der

Bereitstellung der notwendigen Infrastruktur und der Entscheidungsfindung bezüglich strategischer Fragen wahrnimmt: "Four years later, the Apache group formed the Apache Software Foundation (ASF) to provide the logistic support (such as donation or contribution from others) and other business-oriented needs (such as any licensing issues) of the project." [AtGaGe02]. Seitdem wurden verschiedene weitere Teilprojekte durch die ASF initiiert, die zum Teil von ähnlich großer Bedeutung wie das Kernprojekt HTTPD sind. Die Teilprojekte verfolgen i.d.R. alle das Ziel, Softwareprodukte zur Unterstützung von webbasierter Kommunikation zu entwickeln und decken mit dem Tomcat-Application Server oder verschiedenen XML- und Java-verwandten Projekten viele Bereiche der Internet-Infrastruktur ab.

### 3.1.1 Teilprojekte

Die ASF bezeichnet sich selbst als stark dezentralisierte Entwicklergemeinde und ist in verschiedene Teilprojekte unterteilt, welche z.T. wiederum diverse Teilprojekte integrieren. Nach [ApacheOrg02] ist die oberste Teilprojektebene wie folgt strukturiert:

- HTTPD: HTTP Server
- APR: Apache Portable Runtime
- Jakarta: Serverseitige Java-bezogene Projekte
- Perl: Teilprojekt mit Fokus auf die Skriptsprache Perl
- PHP: serverseitige, HTML-embedded Skript Sprache
- TCL: Teilprojekte mit Fokus auf der Erstellung dynamischer Webseiten mit TCL
- XML: webbasierte Softwareprodukte basierend auf XML
- Apache Foundation Project: Foundation Administration und infrastrukturelles Management dargestellt

Neben diesen bereits existierenden Projekten strebt die ASF stehts danach, neue OSS-Projekte in ihre Projektorganisation zu integrieren: "The Apache Software Foundation is always looking for new or existing collaborative software development projects that are in need of a home. This includes both projects related and unrelated to the existing ASF projects." [ApacheOrg02]

Der Fokus dieser Arbeit wird dabei auf dem HTTPD-Teilprojekt und einzelnen Teilprojekten des Jakarta-Projekts liegen. Außerdem wird das Foundation-Projekt näher dargestellt, da es als zentrales Projekt zur Verwaltung und Pflege des Projekts viele managementrelevante Aufgaben umfasst.

31 http://www.apache.org

### 3.1.2 Apache HTTPD

Das Apache-HTTPD-Projekt repräsentiert den Ausgangspunkt und die Basis des gesamten Apache-Projekts und definiert damit die Rahmenbedingungen für alle inzwischen involvierte Teilprojekte. Die Zielsetzung des Apache HTTPD-Projekts umfasst die Entwicklung einer professionellen und portablen Server Software, welche hochwertige HTTP-Dienste basierend auf aktuellen http-Protokoll-Standards zur Verfügung stellt: "The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server software product for various modern desktop and server operating systems. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards." [HttpdApOrg02]

Im August 2000 wurde die Entscheidung getroffen, die Software Architektur erneut zu überarbeiten und den Server vollständig neu zu implementieren, mit dem Ziel einen kompakteren und verbesserten Software Entwurf zu realisieren: "Their original aim was to have the new server released by the end of 2000." [AtGaGe02]. Die Beta-Testphase wurde aber erst im April 2001 erreicht, das erste stabile Release wurde erst im April des darauffolgenden Jahres veröffentlicht: "In reality, their effort of creating Apache 2.0 didn't get into first beta testing stage until April of 2001 (i.e. the 'testing'). [...] The ultimate stable public release did not become available until the beginning of April 2002 (i.e. the 'ultimate release'), which is approximately 1.5 years behind their original goal." [AtGaGe02]

Nach [HttpdApOrg02] existieren im Rahmen des HTTPD-Projekts neben dem primären Weiterentwicklungsansatz die folgenden Teilprojekte:

- Documentation: Erstellung, Wartung und Pflege der Dokumentation des Servers
- Test: "Regression and performance testing tools"

#### 3.1.2.1 OSS: HTTPD Server

Der aktuelle Apache Webserver, respektive das Softwareprodukt des Apache HTTPD-Teilprojekts, repräsentiert inzwischen den etabliertesten Webserver weltweit mit einem Marktanteil von über 50% in 1999 und 60 % in 2000: "The Apache Web Server, free and publicly available, is currently in use on over 50% of all Internet web sites." [GrTa99]. "By Beginning of the year 2000, the Netcraft Web survey indicated that Apache was responsible for hosting over 60 % of all Web sites on the Internet." [Pavlicek00]. Der Status des meistverwendeten Webservers unterstreicht damit auch die Qualität und Konkurrenzfähigkeit der entwickelten OSS und die Relevanz des zugrundeliegenden Entwicklungsmodells.

Kompilierte Binärversionen des Servers sind neben dem Quellcode für alle gebräuchlichen Plattformen verfügbar und die Anwender- und Entwicklergemeinde des Projektes stellt inzwischen eine der umfangreichsten und bedeutendsten im OSS-Kontext dar.

[HttpdApOrg02][32] beschreibt die Funktionalitäten des aktuellen Release des httpd-Servers mit den folgenden Eigenschaften:

- HTTP 1.1 kompatibel (RFC2616)
- Konfigurierbar und erweiterbar durch zusätzliche Module
- Apache API
- Plattformkompatibilität zu allen gebräuchlichen Plattformen (Windows NT/9x, Netware 5.x, OS/2, und verschiedenen Unix-Derivate: "Apache web server is known to work on at least 28 different operating systems. It difficult to know exactly how many operating systems the Apache web server has been ported to as there is no known central list of ports." [Jordan01]
- Konfigurierbare Systemfehlermeldungen
- Unbegrenztes URL-Rewriting und Aliasing
- Content Negotiation: Auslieferung von Webseiten basierend auf den Darstellungsmöglichkeiten des Anwenders
- Definierbarkeit von virtuelle Webservern

Die folgende Tabelle definiert die Plattformen, für welche aktuelle, kompilierte Binärversionen des Webservers verfügbar sind.

Tabelle 1 Apache HTTPD: Plattformkompatibilität (Quelle: [Jordan01])

| Operating System | Operating System | Operating System | Operating System |
|---|---|---|---|
| aix | digitalunix | netbsd | reliantunix |
| aux | freebsd | netware | rhapsody |
| beos | hpux | openbsd | sinix |
| bs2000-osd | irix | os2 | solaris |
| bsdi | linux | os390 | sunos |
| darwin | macosx | osf1 | unixware |
| dgux | macosxserver | qnx | win32 |

Der Apache Webserver ist modular aufgebaut und fast vollständig in Form von unabhängigen Modulen organisiert, wodurch die kollaborative Entwicklung und die Modifikation des Quellcodes stark vereinfacht wird: "Most of the functionality that comes as part of the Apache web server distribution is in the form of modules, and can be removed or replaced. [...] Besides those included

32 http://httpd.apache.org/docs/misc/FAQ.html

with the distribution, modules are also written to add functions not already in the code […]." [Jordan01]. Über diese modulare Architektur ist es außerdem möglich, das System an die anwenderspezifischen Anforderungen anzupassen und Schnittstellen zu weiteren Applikationen zu entwickeln: "The ability to easily customize the Apache web server to specific applications is a very important capability of the software. These modules allow the Apache web server software to interact with a wide variety of other computer systems such as databases, scripting languages and specialized hardware." [Jordan01]

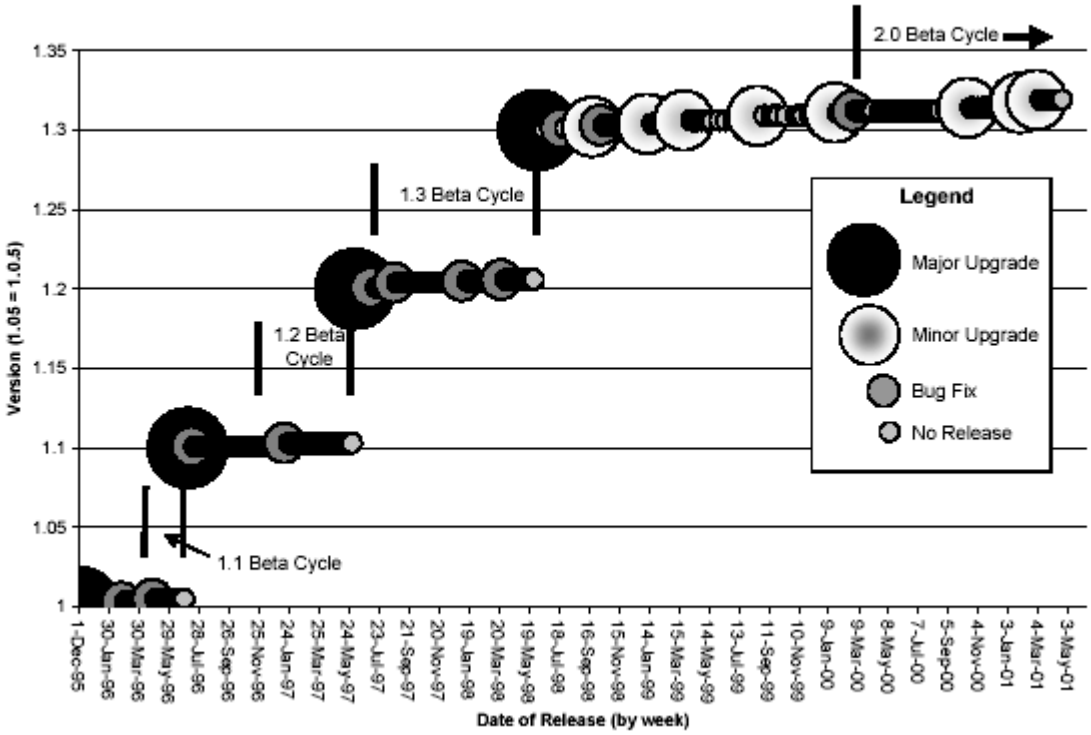
Abgesehen von einigen Komponenten, die durch sogenannte "Core Developer" realisiert wurden, wurde der Hauptanteil der Module durch die Entwicklergemeinde des Projektes implementiert.

#### 3.1.2.2 Release History

Die folgende Grafik aus [Jordan01], visualisiert die Veröffentlichungshistorie der HTTPD-Softwareversionen. Apacheweek [ApWeekCom01] ist eine Online Publikation, die sich speziell mit dem Apache Webserver beschäftigt.

Wöchentlich wird durch die Apacheweek ein Report veröffentlicht, der den aktuellen Status des Apache HTTP- Servers, die aktuelle Version und aktuelle Beta-Aktivitäten thematisiert. [Jordan01]

Bild 7 Apache: Apache Web Server Release History (aus [ApWeekCom01], 2001)




Die folgende Tabelle aus [Jordan01] fasst bezüglich der Releasepraxis im HTTPD-Kontext einige statistische Angaben zusammen, welche die sehr dynamische Veröffentlichungspraxis im OSS-Kontext klar von der in der kommerziellen Software Entwicklung praktizierten Vorgehensweise abgrenzen:

Tabelle 2 Apache: Release History Statistiken (Quelle: [Jordan01])

| Description | Weeks |
|---|---|
| Average time between releases | 11.9 |
| Average time between major releases | 43.7 |
| Average time between minor releases | 17.9 |
| Average time for bug fix releases since last release | 8.8 |
| Frequency of release (any type) | Once every 11.4 weeks |

Die durchschnittliche Zeit zwischen den Releases beträgt ca. 11,4 Wochen, wobei Releases, die lediglich Fehler korrigieren, wesentlich häufiger veröffentlicht werden als umfangreiche Systemupdates.

### 3.1.3 Apache Jakarta

Das Jakarta-Teilprojekt repräsentiert alle serverseitigen Teilprojekte der ASF, welche Applikationen, Bibliotheken oder Werkzeuge auf der Basis von Java entwickeln: "Jakarta is a Project of the Apache Software Foundation, charged with the creation and maintenance of commercial-quality, open-source, server-side solutions for the Java Platform, based on software licensed to the Foundation, for distribution at no charge to the public." [JakartaApOrg02][33]

Das Jakarta-Projekt der ASF kann nach [JakartaApOrg02] in die folgenden Teilprojekte mit den folgenden OSS-Zielsetzungen unterteilt werden, wobei nur einige besonders relevante exemplarisch dargestellt werden:

**Libraries, Tools und APIs:**

BCEL - Byte Code Engineering Library:
Dieses Teilprojekt verfolgt die Zielsetzung, Endanwendern die Möglichkeit zu geben, binäre Java-Class-Dateien zu analysieren, zu erstellen und zu manipulieren.

Commons:
Teilprojekt welches als Container für kleine, wiederverwendbare Code-Komponenten fungiert, die in mehreren Jakarta Teilprojekten verwendet werden.

ECS - Element Construction Set:
Der Inhalt dieses Teilprojektes ist eine Java API für die Generierung von Elementen für verschiedene Markup-Languages, u.a. für HTML 4.0 und XML. ([JakartaApOrg02])

Jmeter:
Jmeter is eine javabasierte Applikation für den Test von Funktionalität und Performance von Applikationen.

Log4J:
Log4J ist eine logging Library für Java

ORO:
ORO stellt eine Bibliothek von Java-Klassen für die Textverarbeitung dar.

33 http://jakarta.apache.org/site/mission.html

Regexp:
Dies ist ein Java Package für die Generierung von regulären Ausdrücken (‚Regular Expressions').

Taglibs:
Taglibs ist eine Zusammenstellung von speziellen Tag-Bibliotheken für Java Server Pages.

Watchdog:
Watchdog beinhaltet Testmodule zur Verifizierung von Servlets bzw. Java Server Pages.

**Frameworks and Engines**

Avalon:
Avalon ist der Name eines komponentenorientierten Programmier-Projektes.

Cactus:
Cactus stellt ein Test Framework für den Test von serverseitigem Java-Code zur Verfügung.

Lucene:
Dieses Teilprojekt befasst sich mit einer javabasierten, plattformunabhängigen Volltext-Such-Maschine.

Struts:
Struts ist ein Framework für die Entwicklung von Web-Applikationen basierend auf Java Servlets und Java Server Pages.

Velocity:
Velocity ist eine allgemeine java-basierte Template-Engine basierend auf XML und XSL und wird oft als Präsentationsschicht in J2EE-Applikationen für die Generierung von dynamischen Inhalten verwendet.

Turbine:
Turbine stellt ein Framework für die Entwicklung von Web-Applikationen basierend auf Velocity oder Java Server Pages zur Verfügung.

**Server Applikationen**

Alexandria:
Versionsverwaltung für das Management von Quellcode, Javadocs und Dokumenten

Avalon:
Komponenten-Framework

James:
E-Mail- bzw. Messaging Server basierend auf Avalon, unterstützt SMTP und POP3

Jetspeed:
Javabasiertes Portalsystem, welches das Turbine Framework instrumentalisiert

Slide:
Content Management System, welches WebDAV unterstützt

Tomcat3:
Java Applikationsserver - Referenzimplementation der Servlet 2.2 und Java Server Pages 1.1 Technologien

Tomcat 4:
Java Applikationsserver - Referenzimplementation der Servlet 2.3 und der Java Server Pages 1.2 Technologien

Die OSS-Produkte des Jakarta Teilprojektes sind inzwischen sehr weit im Bereich von webbasierten Anwendungen verbreitet. Die involvierten Entwickler haben in großem Maße von den Erfahrungen des HTTPD-Projekts profitiert, welche es ihnen ermöglichten, den Entwicklungsprozess zu optimieren und zu automatisieren: "Lessons learned in the Jakarta project, have allowed team members to further refine the software process in order to enhance it and make it more efficient." [AtGaGe02]. Da diese Teilprojekte Eigenschaften aufweisen, die sich zum Teil von denen des HTTPD-Projekts unterscheiden, wurden die zwei Jakarta-Teilprojekte Tomcat und Lucene ergänzend mit in die Analyse einbezogen. Sofern explizit auf diese beiden Teilprojekte referenziert wird, wird dabei in der Regel verallgemeinernd der Terminus Jakarta verwendet. Die Prozesse und Aspekte werden nur dann explizit thematisiert, wenn sie sich in wesentlichen Kriterien von denen des HTTPD-Projekts unterscheiden.

#### 3.1.3.1 Jakarta Tomcat

Tomcat ist ein Teilprojekt von Jakarta, dass die Entwicklung eines Java Applikationsservers zum Ziel hat: "Tomcat is the servlet container used by the official reference implementation for Java Servlets (http://java.sun.com/products/servlet/index.html) and Java Server Pages (http://java.sun.com/products/jsp/)." [AtGaGe02]. Tomcat wird häufig in Verbindung mit dem Apache HTTPD-Server eingesetzt, um serverseitig in Java verarbeitete Applikationen, z.B Java Servlets oder Java Server Pages, in Webseiten zu integrieren. Tomcat wird durch die ASF unter der Apache Software License in

binärer Form und als Quellcode vertrieben: "Tomcat is available for commercial use under the ASL license in both binary and source versions." [AtGaGe02]

#### 3.1.3.2 Jakarta Lucene

Das Teilprojekt Jakarta Lucene stellte javabasierte Volltextsuchfunktionalitäten zur Verfügung. "Lucene is a fully featured, Java-based text search engine, optimized for high performance." [AtGaGe02]

Nach [AtGaGe02] wurde Lucene im September 2001 als Teilprojekt in Jakarta integriert und wird ebenfalls unter der ASL (Apache Software License) veröffentlicht.

### 3.1.4 Apache Foundation Projekt

[ApacheOrg02][34] beschreibt die Zielsetzung des Apache Foundation Projekts wie folgt: "The Foundation Project has the goal of building and sustaining the literal foundation upon which our open-source software projects are based, by managing and staffing the administrative functions of the Apache Software Foundation and providing equipment and communication services to the ASF projects." Das Apache Foundation Projekt dient somit dazu, die nötigen infrastrukturellen Voraussetzungen für die Durchführung der primären Software-Entwicklungsprojekte zu entwickeln und zu verwalten und die Rahmenbedingungen aus der Ebene des Projekt Managements zu schaffen.

### 3.1.5 Organisation

Im folgenden werden die Organisation des Apache Projekts und die impliziten Instanzen thematisiert.

#### 3.1.5.1 Apache Software Foundation (ASF)

"The Apache Software Foundation exists to provide organizational, legal, and financial support for the Apache open-source software projects." [ApacheOrg02][35] Die Apache Software Foundation ist im Jahre 1999 direkt aus der Apache Group hervorgegangen und repräsentiert eine nicht-profitorientierte Organisation mit der zentralen Aufgabe des Project

34 http://www.apache.org/foundation/
35 http://www.apache.org/foundation/faq.html

Managements und der Projektkoordination: "Formerly known as the Apache Group, the Foundation has been incorporated as a membership-based, not-for-profit corporation in order to ensure that the Apache projects continue to exist beyond the participation of individual volunteers, to enable contributions of intellectual property and funds on a sound basis, and to provide a vehicle for limiting legal exposure while participating in open-source software projects." [ApacheOrg02][36]. Die Apache Group wurde damit aber nicht aufgelöst, sondern existiert parallel zur ASF weiter, wobei der Fokus der Apache Group ausschließlich auf dem zentralen Teilprojekt, dem HTTPD-Server liegt, während die ASF mit der gesamtheitlichen Koordinierung der teilprojektübergreifenden Belange des Apache-Projekts betraut ist.

Die primären Aufgaben der Apache Software Foundation liegen nach [Grassmuck02] in der rechtlichen und finanziellen Unterstützung des gesamten Projekts und aller involvierten Teilprojekte. Aufgrund der Rechtsform der ASF, einer jurisische Person in Form eines "nicht profitorientiertes Unternehmens" [Grassmuck02] mit Sitz in Delaware, USA, ist es der ASF möglich, rechtsgültige Verträge mit anderen juristischen Personen einzugehen.

Die einzelnen Teilprojekte des Apache-Projekts werden durch eine Gruppe von selbsternannten ASF-Mitgliedern verwaltet, die ihre Kompetenz durch aktive Mitwirkung am Projekt über einen langen Zeitraum hinweg unter Beweis gestellt haben: "Each project is managed by a self-selected team of technical experts who are active contributors to the project, according to whatever guidelines for collaborative development are best suited to that project." [ApacheOrg02][36]. Neben der Verwaltung der bereits existierenden Teilprojekte, entscheidet die ASF auch über die Initiierung weitere Projekte, d.h. neue Projekte können bei der ASF eingereicht werden.

### Ziele

Die Zielsetzung der ASF wird in den ASF-bezogenen Informationen der FAQ in [ApacheOrg02] wie folgt dargestellt:

- Schaffung der infrastrukturellen Grundlage für offene und kollaborative Softwareentwicklung
- Juristische Person: Schaffung einer unabhängigen, juristischen Entität
- Schutz der Marke "Apache" vor Missbrauch durch andere Organisationen
- Durchführung aller teilprojektübergreifenden Managementaufgaben

36 http://www.apache.org/foundation/faq.html

Die Akteure des Foundation-Projekts repräsentieren somit die ausführende Instanz für Management- und infrastrukturelle Prozesse entsprechend des Metamodells und führt somit aus der Perspektive der Entwicklungsprozesse sekundäre und unterstützende Aufgaben aus.

### Mitgliedschaft

Die Mitgliedschaft in der ASF steht jedem Mitglied der Entwicklergemeinde frei, wobei als Aufnahmevoraussetzung ein außerordentlicher Beitrag zu den Apache-Softwareprojekten und eine überdurchschnittliche aktive Partizipation an den Projekten gegeben sein sollte. Ein individueller Entwickler kann durch die ASF nominiert und durch eine mehrheitliche Zustimmung durch die ASF-Mitglieder im Rahmen einer Abstimmung bestätigt werden: "Individuals who have demonstrated a commitment to collaborative open-source software development, through sustained participation and contributions within the Foundation's projects, are eligible for membership in the ASF. An individual is awarded membership after nomination and approval by a majority of the existing ASF members." [ApacheOrg02][37]. Eine aktuelle Liste der Mitglieder der ASF[38] kann dort ebenfalls eingesehen werden. Der Vorstand der ASF setzt sich nach [Grassmuck02] ausnahmslos aus Mitgliedern der Apache Group zusammen, was vor allem aus der Tatsache resultiert, dass die ASF direkt aus der Apache Group hervorgegangen ist.

### ASF Bylaws

Die Foundation Bylaws[39] reglementieren die Prozesse der ASF und definieren die Prozesse der demokratischen Entscheidungsfindung durch die Mitglieder. Außerdem sind in diesem Artefakt von strategischer Bedeutung für das Projekt die Rechte und Pflichten der einzelnen Akteure bzw. Rollen, die Voraussetzungen für die Ausführung spezifischer Aufgaben und die Organisationsstruktur der ASF definiert.

### Board of Directors

In periodischen Abständen wird durch die Mitglieder der ASF ein Board of Directors gewählt, welche die Aufgabenstellung des organisatorischen Managements der ASF hat: "The ASF members periodically elect a Board of Directors to manage the organizational affairs of the Foundation, as accorded by the ASF Bylaws." [ApacheOrg02][40]. Das Board of Directors (BOF) ernennt

37 http://www.apache.org/foundation/
38 http://www.apache.org/foundation/members.html
39 http://apache.org/foundation/bylaws.html

eine Anzahl sogenannter Officers, welche das Tagesgeschäft der ASF kontrollieren und nötigenfalls regulierend eingreifen.

Die folgende Tabelle stellt die aktuellen Officers des BOF nach [ApacheOrg02][40] dar:

Tabelle 3 Apache: Board of Directors, Stand 11/2002 (Quelle: [ApacheOrg02])

| Function | Name |
|---|---|
| Chairman | Greg Stein |
| President | Dirk-Willem van Gulik |
| Treasurer | Randy Terbush |
| Exec. V.P. and Secretary | Jim Jagielski |
| V.P., APR | Ryan Bloom |
| V.P., Apache Conference Planning | Ken Coar |
| V.P., Apache DB | Jason van Zyl |
| V.P., Apache HTTP Server | Ben Hyde |
| V.P., Apache PHP | Rasmus Lerdorf |
| V.P., Apache TCL | David N. Welton |
| V.P., Apache XML | Ted Leung |
| V.P., Jakarta | Sam Ruby |
| V.P., Java Community Process | Jason Hunter |
| V.P., Perl-Apache | Doug MacEachern |

Wie auch die anderen Projekte, die primär das Ziel der Softwareentwicklung verfolgen, ist auch die Aufgabe des Managements des Apache-Projekts durch das Foundation Teilprojekt stark von der Mitwirkung Freiwilliger abhängig.

### 3.1.6 Community

Dieser Abschnitt thematisiert einige signifikante Merkmale der Community des Apache-Projekts. Die folgenden Characteristika wurden anhand des HTTPD Projekts identifiziert und konnten im Rahmen des Jakarta-Projekts bestätigt bzw. nicht widerlegt werden.

40 http://www.apache.org/foundation/faq.html

### Developer

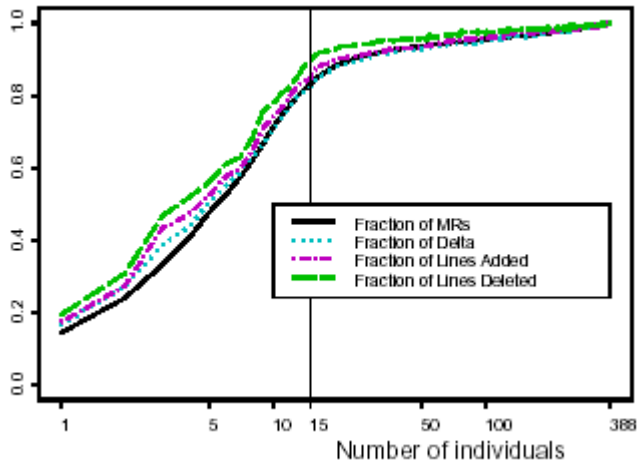
[MoFiHe00] identifiziert neben der Vielzahl von weltweit verteilten Anwendern ca. 400 aktive Entwickler. 182 Entwickler trugen 695 Patches bei, die auf Bug Reports beruhen, während 249 Entwickler 6092 Quellcodemodifikationen beitrugen, die nicht direkt auf der Gundlage eines Fehlerberichts entwickelt wurden. Damit wird offenbar, dass der Entwicklungsprozess nicht im selben Maße auf dem zentralen Bug Tracking System basiert, wie dies z.B. im Mozilla Kontext der Fall ist.

### Core Developer

[MoFiHe00] beweist außerdem die Existenz von sogenannten Core Developers, die für einen Großteil des Quellcodes verantwortlich zeichnen.

Die folgende Grafik stellt den kumulierten Anteil der Quellcodeänderungen der Anzahl der Entwickler gegenüber, wobei die Entwickler gemäß des Umfangs ihres Anteils in absteigender Reihenfolge angeordnet sind. LOC steht hierbei für "Lines of Code". Ein MR repräsentiert in dieser Grafik einen spezifischen Quellcode-Commit in das CVS-Reprository, während ein Delta die extrahierten Quellcodeänderungen an einer spezifischen Quellcodedatei darstellt.

Bild 8 Apache: Kumulierter Anteil an Code Änderungen (aus [MoFiHe00])

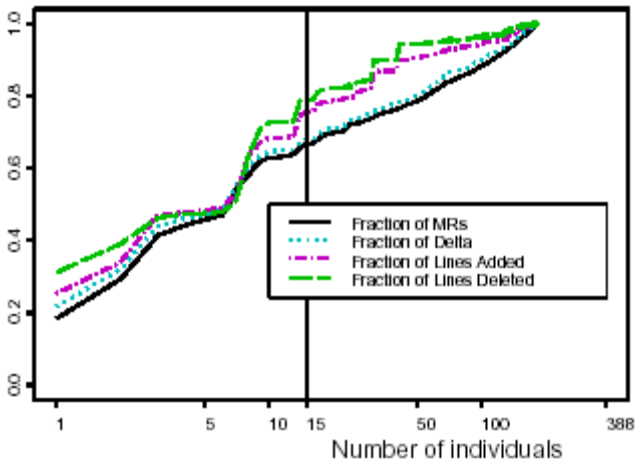



Dabei wird offenbar, dass die 15 produktivsten Entwickler des Projekts für mehr als 83% der MR's und Deltas, für 88% der hinzugefügten und für 91% der gelöschten LOC verantwortlich sind und somit für einen Großteil der Quellcodeänderungen zuständig sind. Obwohl eine solche Rolle nicht explizit im Projekt definiert wurde, kann sie somit als existent betrachtet werden. Auf

diesen Personenkreis wird im Verlauf dieses Kapitels stets durch den Terminus des Core Developers referenziert.

In der folgenden Grafik werden die selben Werte im Kontext von ausschließlich Bug Report-basierten Änderungen gegenübergestellt:

Bild 9 Apache: Kumulierter Anteil an BR-basierten Änderungen (aus [MoFiHe00])




Auf die genau Definition des Terms "Bug Report" im Apache-Kontext wird noch explizit eingegangen. Aus obiger Abbildung kann abgeleitet werden, dass in diesem Fall lediglich 66% der Änderungen durch die Core Developer entwickelt wurden und somit eine wesentlich geringere Kohärenz identifiziert wurde. Da im Rahmen des Apache HTTPD-Projekts ausschließlich Software-Fehler als Bug Report kommuniziert werden, und somit Änderungen, die nicht auf einem entsprechenden Artefakt beruhen i.d.R. eine funktionale Systemerweiterung implementieren, lässt sich ableiten, dass neue Funktionalitäten fast ausschließlich durch die Core Developer hinzugefügt werden (siehe Bild 9) wohingegen Fehlerbehebungen auch zu weiten Teilen durch die anderen Entwickler ausgeführt werden.

Außerdem hat die Fallstudie in [MoFiHe00] ergeben, dass nur wenige Entwickler, die nicht der Kerngruppe angehören, über einen längeren Zeitraum aktiv Quellcode entwickelt haben.

### Bug Reporter

Basierend auf den Informationen des Bug Tracking Systems definiert [MoFiHe00] eine Zahl von 3060 Personen, die bisher 3975 Fehlerberichte beigetragen haben. 458 Personen erstellten 591 Bug Reports, welche direkte

Quellcodeänderungen zur Folge hatten, während 2654 Personen 3384 Fehlerberichte beitrugen, die keinerlei Patch nach sich zogen.

Die Existenz einer Kerngruppe analog zu den Core Developers konnte in diesem Fall nicht belegt werden, da die 15 aktivsten Personen lediglich 213 Bug Reports (5 %) beitrugen. Fast 2600 Personen trugen nach [MoFiHe00] lediglich einen einzigen Fehlerbericht in die Datenbank ein. Nur drei Personen der Core Developer gehörten auch zu den 15 aktivsten Personen im Kontext der Bug Reports. Daher lässt sich nach [MoFiHe00] ableiten, dass die Durchführung von Software Tests annähernd gleichmässig verteilt durch die gesamte Projekt Community wahrgenommen wird.

**Core Group - Apache Group**

Neben dem Entwicklerkern existiert im HTTPD-Projekt auch eine sogenannte Core Group, die sich den Aufgaben der Koordination und des Managements widmet und aus den ursprünglichen Initiatoren des Projekts besteht: "There is a core group of contributors (informally called the 'core') which was formed from the project founders and is augmented from time to time when core members nominate outstanding contributors and the rest of the core members agree… The term 'The Apache Group' technically refers to this core of project contributors." [HttpdApOrg02][41]

Diese Core Group des HTTPD-Projekts wird auch Apache Group genannt und ist von der ASF abzugrenzen, da die ASF teilprojektübergreifend tätig ist und die Apache Group lediglich im Kontext des HTTPD-Projekts agiert.

### 3.1.7 Allgemeine Charakteristika der Prozesse

In diesem Abschnitt werden einige grundlegende Merkmale des Entwicklungsprozesses thematisiert und eine gesamtheitliche Perspektive auf den Entwicklungs- und Releaseprozess ermöglicht, während in den nachfolgenden Abschnitten die einzelnen Prozesse basierend auf dem definierten Metamodell dargestellt werden.

[MoFiHe00] beschreibt die Notwendigkeit eines einheitlichen und z.T. reglementierten Entwicklungsprozesses, welche sich aus der Ausgangssituation der globalen Verteilung und Heterogenität der Entwickler ergibt: "[...] it was clear from the very beginning that a geographically distributed set of

41 http://httpd.apache.org/ABOUT_APACHE.html

volunteers, without any traditional organizational ties, would require a unique development process in order to make decisions."

Die Entwicklungs- und Entscheidungsprozesse müssen demnach die dezentralen Arbeitsräume und die asynchrone Kommunikation der Entwickler unterstützen und ein möglichst demokratisches Abstimmungsverfahren über verschiedene Entscheidungen etablieren, welches die demokratische Struktur des Projekts unterstützt, aber trotzdem in der Lage ist, zeitnah Entscheidungen über kritische, projektrelevante Fragen zu treffen.

#### 3.1.7.1 Anforderungsanalyse

Der Prozess der Anforderungsanalyse wird in großem Maß durch die Mitglieder der Projekt-Community wahrgenommen: " [...] potential features can come from enhancement reports that are submitted to Bugzilla. Second, features can be in the form of patches already available (committed into the CVS already)." [AtGaGe02]. Neben dieser dominierenden Variante der Anforderungsdefinition können Anforderungen auch durch ein sogenanntes Project Management Committee definiert werden, die dabei aber in der Regel lediglich die strategische Ausrichtung und Zielsetzung und entsprechende Anforderungen definieren: "The Project Management Committee details a 'ROADMAP' (http://cvs.apache.org/viewcvs.cgi/httpd-2.0/ROADMAP) file, which is overall direction for the preliminary requirements of HTTPD." [AtGaGe02]

Über diese Anforderungen wird anschließend durch die Community abgestimmt, wobei relevante Anforderungen in einem sogenannten STATUS-File spezifiziert werden, welches für jedes Softwarerelease erstellt wird: "Each item on the 'ROADMAP' file is then detailed out and the results of these votes are recorded, all together to be put into the 'STATUS' (http://cvs.apache.org/viewcvs.cgi/httpd-2.0/STATUS) file." [AtGaGe02]. Dieses STATUS-File beschreibt die Evolution des entsprechenden Release und kann durch die Entwickler eingesehen werden. Dies ermöglicht ihnen die Auswahl von Anforderungen bzw. Funktionalitäten, welche sie in Form eines Patches implementieren möchten.

#### 3.1.7.2 Aktive Mitwirkung dezentraler Akteure

[ApacheOrg02][42] beschreibt die Möglichkeiten zur aktiven Mitwirkung an allen Projekten der ASF wie folgt: "As a volunteer-based organization, the Apache Software Foundation stands most in need of dedicated volunteers who can

42 http://www.apache.org/foundation/contributing.html

work on software, documentation, or administrative issues within each of the ASF projects, including the Foundation project itself."

Besonders das Software Testing, die Generierung von Bug Reports und der Prozess des Bug Triage, der Prüfung eine Bug Reports auf Relevanz, wie es im Mozilla-Kontext dargestellt wurde, werden als fundamentale Aktivitäten auf Seiten der Anwendergemeinde definiert: "In terms of software developers, there is typically a large need for people willing to test the various pieces of software, in various environments, and to track, reproduce, isolate, and fix problems reported by the user community." [ApacheOrg02]

Auch die Managementprozesse können z.T. durch dezentrale Akteure der Community ausgeführt werden, wobei aber die primären Akteure in diesem Kontext durch die verschiedenen Management-Instanzen repräsentiert werden. So wird z.B. die Betreuung der Anwender durch die Organisation regionaler User Groups oder das Bereitstellen von Anwenderinformationen in einschlägigen Foren als hilfreiche Aktivität angesehen: "Organizing local user groups, volunteering to work on user conferences, and helping less experienced users on the various on-line forums are all ways in which you can contribute to the ASF projects." [ApacheOrg02]. Das entwicklungsbegleitende Foundation-Projekt, mit der Zielsetzung der Schaffung von adäquaten Rahmenbedingungen und Infrastrukturen, kann ebenfalls von der Mitwirkung von Freiwilligen profitieren, die etwaige organisatorische Aufgaben wahrnehmen.

#### 3.1.7.3 Kommunikation

Die projektweite Kommunikation erfolgt hauptsächlich über Mailing Listen, Newsgroups und Internet Relay Chat, wobei die verschiedenen fach- und zielgruppenspezifischen Mailing Listen als primäre Kommunikationskanäle betrachtet werden können. "Our primary method of communication is our mailing list. [...] We discuss new features to add, bug fixes, user problems, developments in the web server community, release dates, etc." [HttpdApOrg02][43]

Es steht dabei jeder interessierten Person frei, Mitglied einer Mailing Liste oder Newsgroup zu werden.

43 http://httpd.apache.org/ABOUT_APACHE.html

#### 3.1.7.3.1 Dedizierte Kommunikationskanäle

[AtGaGe02] identifiziert die folgenden Kategorien von Mailing Listen, die ihrerseits noch in weitere Kategorien eingeteilt werden können:

- Announcement Lists:
  Nach [AtGaGe02] dienen diese Listen lediglich der Kommunikation von wichtigen Informationen an die Community, wie z.B. die Veröffentlichung eines finalen Softwarerelease und ist daher nicht sonderlich stark frequentiert.
- User Lists:
  Diese Mailing Listen werden vor allem durch die Anwender der Software verwendet um anwendungsbezogene Themen zu verhandeln. Sie stellen daher eine wichtige Quelle der Information und damit der Substitution von traditionellem Anwendersupport dar.
- Developer Lists:
  Diese Kategorie von Mailing Listen wird primär dazu verwendet, entwicklungsrelevante Themen, wie die Qualität von Patches oder die Relevanz von Fehlerberichten zu kommunizieren und zu diskutieren.
- Commit Lists:
  Diese Klasse der Mailing Listen werden verwendet, um automatisch alle 'Commits', also die Integration von Patches in einen spezifischen CVS-Verzeichnisbaum, zu kommunizieren. I.d.R. sind alle mit Commit-Privilegien ausgestatteten Entwickler Mitglied dieser Mailing Liste.

Die oben aufgeführten Kategorien der Mailing Listen werden weiter in verschiedene fachspezifische Themen unterteilt und existieren außerdem immer im Rahmen eines spezifischen Teilprojekts.

Neben diesen Mailing Listen existieren dedizierte Usenet Newsgroups, Website Foren oder FAQs, die nach ähnlichen Kriterien instanziiert wurden. Außerdem stellt der IRC (Internet Relay Chat) ein weiteres wichtiges Kommunikationsmedium dar.

#### 3.1.7.3.2 Kommunikationsrichtlinien

Analog zu anderen OSS-Projekten wurden auch für das Apache-Projekt verschiedene Richtlinien definiert, welche die Kommunikation reglementieren sollen, um eine gewisse Kontinuität und Standardisierung dieses Prozesses zu ermöglichen.

So sollten Entwickler über die verschiedenen Kommunikationskanäle ausschließlich Themen besprechen, die nicht direkt als verifizierter Softwarefehler oder "Enhancement Request" über das dedizierte Bug Tracking System kommuniziert werden können. Der "Mailing List Guide"[44] des HTTPD-

Projektes verweist z.B. auch auf das diesbezügliche Standardwerk von Eric S. Raymond: "Asking Questions the Smart Way"[45]

Weiterhin werden in diesem Guide die folgenden Anforderungen definiert:

- Verwendung des ASCII-Text-Formats: "Please configure your email client to send plain text so that everyone will be able to read your message."
- Verwendung englischer Sprache
- Verifizierung der Novität eines Themas: "Be sure to read the FAQ and search the archives before you post."
- Adressierung der adäquaten Zielgruppe: Da verschiedene fach- und zielgruppenspezifische Newsgroups bzw. Mailing Listen existieren, schafft die Selektion eines probaten Kommunikationskanals eine wichtige Voraussetzung für die erfolgreiche Bearbeitung eines Themas

## 3.2 Entwicklungsprozesse

In diesem Abschnitt werden die Prozesse dargestellt, welche primär und direkt in den Prozess der Softwareentwicklung involviert sind und den Definitionen des Metamodells entsprechend als Entwicklungsprozess klassifiziert werden können.

### 3.2.1 Bezug der Software

"Downloading application source or binary code is the first step towards application usage or development." [AtGaGe02]

In allen Teil-Projekten der ASF werden sowohl kompilierte, binäre Softwareversionen als auch der Quellcode veröffentlicht und über verschiedene Kanäle zur Verfügung gestellt.

Zum einen besteht die Zugriffsmöglichkeit über die Websites der Teilprojekte als auch über ein zentrales Distributionsarchiv[46], welches auf stabile Softwareversionen aller Teilprojekte referenziert: "If you are looking for a stable release of the source code, you should download it from the distribution directory." [ApacheOrg02][47]

44 http://httpd.apache.org/userslist.html
45 http://www.tuxedo.org/~esr/faqs/smart-questions.html
46 http://www.apache.org/dist
47 http://www.apache.org/foundation/cvs.html

Weiterhin werden alle Quellcodes in einem CVS-Archiv verwaltet, um somit den Ansprüchen der kollaborativen und heterogenen Entwicklungsprozesse gerecht zu werden: "Almost all files relating to Apache, either the actual sources or the files that aren't part of the distribution, are maintained in a CVS repository."[HttpdApOrg02][48]

#### 3.2.1.1 Download des Quellcodes

In diesem Abschnitt werden die Möglichkeiten zum Bezug des Quellcodes sowohl im Rahmen des HTTPD-Projekts als auch im Kontext der Jakarta-Projekte dargestellt. Prinzipiell werden verschiedene Arten von Releases über die Websites der Projekte zur Verfügung gestellt. Diese auch als Snapshots bezeichneten Releases können durch jeden interessierten Anwender bezogen werden: "Everyone has access to the CVS snapshots" [Sandred01]. Dabei repräsentieren sie aber nicht immer den aktuellsten Entwicklungsquellcode, dieser steht lediglich direkt im CVS-Archiv zur Verfügung.

**HTTPD**

Im HTTPD-Projekt werden die Releases nach [HttpdApOrg02][49] entsprechend ihres Verwendungszwecks in die folgenden Klassen unterteilt:

- Alpha Release:
  Ein Alpha Release kann als sehr instabil und ungetestet betrachtet werden und stellt das initiale Release im Rahmen des Release-Prozesses dar: "Alpha indicates that the release is not meant for mainstream usage or may have serious problems that prohibits its use. When a release is initially created, it automatically becomes alpha quality."
- Beta Release:
  Für die Transition in den Beta-Status müssen mindestens drei Entwickler mit Commit-Privileg positiv gestimmt haben, wodurch das Release als kompilierbar und funktional eingestuft wird: "Beta indicates that at least three committers have voted positively for beta status and there were more positive than negative votes for beta designation. This indicates that it is expected to compile and perform basic tasks." Trotzdem gilt ein Beta-Release noch nicht als fehlerfrei und stabil.
- General Availability Release:
  Dieses Release repräsentiert die finale Version des Quellcodes, die auch in binärer Form als offizielle neue Softwareversion veröffentlicht wird. Der Abstimmungsprozess hierfür findet analog zum Beta-Release statt:

48 http://httpd.apache.org/dev/devnotes.html
49 http://httpd.apache.org/dev/release.html

"General Availability (GA) indicates that at least three committers have voted positively for GA status and that there were more positive than negative votes for GA designation."

**Jakarta**

Der Quellcode der Jakarta Teilprojekte kann nach [AtGaGe02] analog dazu ebenfalls in verschiedene Kategorien eingeteilt werden:

- Release Source Drops:
  Release Source Drops genügen hohen Stabtilitätsansprüchen, analog zu den General Availability Releases im HTTPD-Kontext: "This code [...] is intended for high quality products."
- Milestone Source Drops:
  Milestone Source Code wird als relativ zuverlässig, aber nicht fehlerfrei und stabil betrachtet und genügt ähnlichen Anforderungen, wie die Beta Releases des HTTPD-Projekts: "Milestone code is not intended for commercial products because although much of the functionality is acceptable, there still exist many bugs. The advantage of milestone source drops is that they allow their users to explore and use future feature of the product."
- Nightly Snapshots:
  Diese Quellcode Snapshots werden automatisch durch CVS erstellt und dienen dem Testen und der weiteren Erstellung der "Nightly Builds", welche im folgenden noch explizit dargestellt werden. Daher sind sie als absolut instabil und noch nicht verifiziert anzusehen. "These code 'snapshots' are automatically taken by CVS everyday."

#### 3.2.1.2 Download via CVS

Die Project Management Guidelines des HTTPD-Projekts weisen darauf hin, dass jeglicher Quellcode auch über ein CVS-Repository verwaltet wird: "All of the Apache products are maintained in shared information repositories using CVS on httpd.apache.org." [HttpdApOrg02][50] Dies trifft auch auf das Jakarta-Projekt zu.

Lesezugriff auf das Archiv wird allen Entwicklern eingeräumt, wobei der schreibende Zugriff nur entsprechend privilegierten Entwicklern zugewiesen wird, welche im Folgenden als 'Committers' bezeichnet werden: "Only some of the Apache developers have write access to these repositories; everyone has read access via anonymous CVS." [HttpdApOrg02][51]

50 http://httpd.apache.org/dev/guidelines.html

Der Zugriff auf CVS kann dabei über einen CVS-Client, das webbasierte Interface ViewCVS oder verschiedene andere Werkzeuge erfolgen: "This is performed using CVS, WinCVS, ViewCVS, CVSup, and Rsync (http://jakarta.apache.org/site/cvsindex.html)." [AtGaGe02]. Zudem kann der Zugriff über einen Nutzeraccount oder anonymisiert durch jede Person erfolgen: "Using these tools developers and committers may access the data repository in two ways: anonymously or via login access." [AtGaGe02]

#### 3.2.1.2.1 Schreibzugriff auf CVS

Der schreibende Zugriff auf das CVS-Archiv, der zur Integration von Patches benötigt wird, kann nur durch die Instanzen des Projektmanagements des jeweiligen Teilprojekts ermöglicht werden. Dieser Prozess wird durch das Stellen einer Anfrage über eine Mailing Liste initiiert und durch die fallspezifische Entscheidung des Projekt-Managements basierend auf den bisherigen Projektbeiträgen des Entwicklers abgeschlossen: "Developers who want write access need to ask for it on the mailing list and, if approved, need to ask the admin of httpd.apache.org for a user account." [HttpdApOrg02][51]

### 3.2.1.3 Download von Builds / Binärdateien

Die zentrale Apache Website[52] stellt Binaries, sogenannte Builds, aller stabilen Releases der verschiedenen Apache-Projekte zentral zur Verfügung.

**HTTPD**

Im HTTPD-Projekt werden die binären Veröffentlichungen basierend auf den bereits beschriebenen Kategorien des Quellcodes erstellt und veröffentlicht. Dabei entstehen die folgenden Builds:

- Alpha Builds:
  Ein Alpha Build wird regelmäßig durch den Release Manager erzeugt und basiert auf dem jeweils aktuellem Entwicklungsquellcode. Das Alpha-Build richtet sich an Entwickler mit der Zielsetzung der intensiven Softwaretests und Fehlerbehebung.
- Beta Builds:
  Basierend auf den Ergebnissen der Alpha-Testphase wird ein Beta Build durch den Release Manager generiert.und ebenfalls auf der Entwickler-

51 http://httpd.apache.org/dev/guidelines.html
52 http://www.apache.org/dist

Website veröffentlicht: "The output is submitted on the developers website for further testing." [AtGaGe02]

- General Availability Build:
  Nachdem die letzten Fehler der Beta-Version durch Patches behoben wurden, generiert und veröffentlicht der Release Manager ein General Availability Build, welches als offizielle neue Softwareversion der Allgemeinheit auf der öffentlichen Website des Projekts zur Verfügung gestellt wird: "...the release manager can release the GA to the public whenever he/she deems it is ready for release." [AtGaGe02]. Über offizielle Apache Webseiten[53] stehen diese Binärversionen der Software der Allgemeinheit zur Verfügung.

### Jakarta

Im Rahmen des Jakarta-Projekts existieren nach[JakartaApOrg02][54] verschiedene Varianten von Builds, die zu unterschiedlichen Zwecken durch das Projekt Management generiert werden:

- Release Builds:
  Ein 'Release Build' stellt die stabilste Version eines Builds dar und muss dabei ähnlichen Anforderungen wie die 'General Availability Releases' des HTTPD-Projekts genügen
- Milestone Builds:
  Die Milestone Builds werden als relativ zuverlässig, aber nicht als fehlerfrei eingestuft und sollten nur von erfahrenen Anwendern verwendet werden. "For developers, these builds serve as a mechanism to track progress towards a release."
- Nightly Builds: ´
  Die 'Nightly Builds' werden in einem täglichen Prozess erstellt und sind als sehr instabil zu betrachten. Sie dienen lediglich der Durchführung von Software-Tests.
- Demo Builds:
  'Demo Builds'werden von Zeit zu Zeit erstellt, um eine Demonstration eines neuen Software-Releases zu ermöglichen: "Demo Builds are useful for showing demonstrations of our products."

53 http://httpd.apache.org/download.cgi, http://www.apache.org/dist/httpd/#binaries
54 http://jakarta.apache.org/site/binindex.html

### 3.2.2 Erstellung eines Bug Reports

Der Prozess der Erstellung und Kommunikation von Berichten über Softwarefehler ist im Apache-Kontext teilprojektübergreifend generalisierbar, er findet in allen Teilprojekten in einer ähnlichen Form statt.

Das Apache-Projekt instrumentalisiert inzwischen das Bugzilla System, welches ursprünglich im Mozilla-Kontext entwickelt wurde, für die Bearbeitung und Verfolgung von Bug Reports und hat zu diesem Zweck eine Instanz des Systems[55] eingerichtet. Daher werden die Aktivitäten bezüglich der Bug Reports ebenfalls durch die Charakteristika des Bugzilla-Systems determiniert und folgen somit Prozessen, die sich an den im Mozilla-Kontext praktizierten Abläufen orientieren und daher an dieser Stelle nicht mehr erschöpfend und detailliert beschrieben werden.

#### 3.2.2.1 Vorarbeiten

[HttpdApOrg02][56] beschreibt folgende prinzipielle Arbeitsschritte im Vorfeld der Erstellung eines Bug Reports:

1. Verifikation der Existenz des Softwarefehlers in der aktuellen Version der Software
2. Verifikation der korrekten Installation
3. Verifikation, dass nur offiziell veröffentlichter Quellcode und keine zusätzlichen Patches verwendet wurden
4. Recherche in den adäquaten Log-Dateien und Identifikation aller relevanten Informationen
5. Recherche im Bug Tracking System zur Verifikation der Novität des Eintrags und zur Vermeidung von multiplen Bug Reports bezüglich eines spezifischen Softwarefehlers. Hierbei ist es wichtig, sowohl die bereits geschlossenen, als auch die offenen Bug Reports einzubeziehen.
6. Kommunikation des Softwarefehlers über die Anwender-Mailing Liste des Projekts, um sicherzustellen, dass es sich um keinen Anwender- oder Konfigurationsfehler handelt
7. Falls innerhalb einiger Tage kein Feedback auf die Nachricht eintreffen sollte, kann zur Generierung des Bug Reports übergegangen werden

55 http://nagoya.apache.org/bugzilla
56 http://httpd.apache.org/bug_report.html

#### 3.2.2.2 Generierung eines Bugzilla-Eintrags

Zur Generierung eines Eintrags in die Bugzilla Datenbank ist es notwendig, über einen Nutzeraccount zu verfügen. Dieser kann durch jeden Anwender durch Angabe des Namens und der E-Mail-Adresse angelegt werden, wobei das entsprechende Passwort anschließend per E-Mail zugestellt wird.

Weiterhin muss das relevante Teilprojekt ausgewählt werden, was die Präsentation eines projektspezifischen Eingabeformulars zur Folge hat. In diesem Formular werden verschiedene Informationen und Metadaten bezüglich des Softwarefehlers abgefragt, die projekspezifisch durch Maintenance Instanzen der Teilprojekte definiert wurden.

#### 3.2.2.3 Format des Bug Reports

Die wichtigsten Informationen des Bug Reports werden in den folgenden Attributen definiert, die sowohl im Kontext der Jakarta-Projekte, als auch für das HTTPD-Projekt in Form von Metadaten mit jeder Instanz eines Anforderungsartefakts verknüpft sind:

- Version: Versionsnummer der Software
- Component: Modul oder fachlich abgrenzbarer Bereich
- OS: Betriebssystem der betreffenden Software
- Plattform: verwendete Hardware-Plattform
- Severity: Blocker, Critical, Major, Enhancement, Normal, Minor
- State: Status der Bearbeitung des Softwarefehlers
- Assigned: Entwickler dem die Bearbeitung zugeordnet ist
- Summary: Zusammenfassung
- Description: Umfassende Beschreibung
- URL: URL mit weiterführenden Informationen oder zu Reproduktion des Fehlers

Diese Metainformationen wurden analog zu denen der Bug Reports im Mozilla-Projekt definiert. Daher wird auf die spezifischen Inhalte nicht detailliert eingegangen. Weiterhin werden diese Daten durch verschiedene, softwareproduktspezifische Informationen ergänzt.

### 3.2.3 Erstellung eines Enhancement Request

Neben dem Berichten von Softwarefehler kann ein Bug Report auch dazu verwendet werden, einen sogenannten Enhancement Request zu kommunizieren, der eine funktionale Systemerweiterung zum Ziel hat.

"Anybody in the community can submit a new feature request via Bugzilla by indicating the submission as an 'enhancement'" [AtGaGe02]. Diese Kennzeichnung eines Bug Reports erfolgt, indem dem Attribut 'Severity' der Wert 'Enhancement' zugewiesen wird: "To suggest a new feature, set the severity of the bug to 'enhancement'." [JakartaApOrg02][57]. In den folgenden Abschnitten repräsentiert der Begriff Bug Report z. T. sowohl Enhancement Requests als auch Berichte über Software Fehler, da beide Arten der Anforderungsartefakte in den gleichen Prozessen verwendet und im Bugzilla-System über die gleichen Metadaten verwaltet werden.

### 3.2.4 Entwicklung von Patches

Die eigentliche Entwicklungstätigkeit erfolgt dezentral durch die verteilten Entwickler in Form von individuellen und parallel ausgeführten Patchentwicklungszyklen: "The actual code development takes place on the developers' local machines, with proposed changes communicated using a patch [...] and committed to the source repository by one of the core developers using remote CVS." [HttpdApOrg02][58]

In Anlehnung an [MoFiHe00] können die folgenden Arbeitsschritte identifiziert werden, die vom Entwickler und entsprechend privilegierten Personen zur Erstellung eines Patches durchgeführt werden:

1. Recherche nach relevanten Problemen bzw. Anforderungen (Bug Triage)
2. Identifikation etwaiger weiterer Entwickler, die bereits an der gleichen Aufgabenstellung arbeiten
3. Bezug des Quellcodes
4. Generierung eines lokalen Entwicklungsquellcodeverzeichnisses
5. Identifikation einer Lösung
6. Entwicklung und Test des Patches
7. Extraktion des Patches
8. Kommunikation und Diskussion (Review) des Patch durch die Community
9. Integration des Patch in das zentrale Quellcodeverzeichnis (Commit-Privileg erforderlich)

Dieser Prozess wird i.d.R. iterativ wiederholt, bis das Patch den Ansprüchen der Reviewer genügt. Die Integration des Patches wird im Falle von komplexen Änderungen erst nach vollständigem Durchlauf und Review des Patches

57 http://jakarta.apache.org/site/bugs.html
58 http://httpd.apache.org/ABOUT_APACHE.html

ausgeführt: "Depending on the scope of the change, this process may involve many iterations before reaching a conclusion, though it is generally preferred that the entire set of changes needed to solve a particular problem be applied in a single commit." [MoFiHe00]. Als komplexe Patches werden Quellcodemodifikationen betrachtet, die starke Abhängigkeiten zu anderen Modulen oder dem Systemkern aufweisen oder generell ein großes Risiko für die Stabilität des Gesamtsystems darstellen.

Im folgenden werden einige zentrale Aktivitäten des Patchentwicklungszyklus explizit dargestellt.

#### 3.2.4.1 Bug Triage

Die Identifikation eines relevanten Bug Reports erfolgt auf der Basis des Prozesses des Bug Triage, bei dem der Entwickler die relevanten Informationsquellen nach unbearbeiteten und verifizierten Fehlerberichten durchsucht. Neben der primären Informationsquelle, dem Bugzilla System, werden hierbei auch die Mailing Listen oder Newsgroups verwendet.

Das Bugzilla System stellt ein umfangreiches Archiv von strukturiert über Metadaten beschriebenen, aber z.T. auch redundanten und diffus organisierten Bug Reports dar, wodurch die Identifikation aktueller und relevanter Entwicklungsaktivitäten erschwert wird. Daher wird der Prozess des Bug Triage, also der Identifikation und Zuordnung von existierenden und relevanten Bug Reports aus Newsgroups, Mailing Listen und dem Bugzilla System durch einige wenige Entwickler ausgeführt, die somit die Basis für die weitere Entwicklung schaffen: "The project relies on one or two interested developers to perform periodic triage of the new reports: removing mistaken or misdirected reports, answering reports that can be answered quickly, and forwarding items to the developer mailing list if they are considered critical." [MoFiHe00]. Diese Selektion führt dazu, dass einige wenige Entwickler in großem Maße die Bearbeitung von Änderungsanfragen determinieren, indem sie bereits eine erste Sondierung der Bug Reports vornehmen. Dieser Prozess stellt sicher, dass anschließend interessierte Entwickler auf eine Datenbank von weitgehend konsistenten und relevanten Bug Reports zugreifen können.

#### 3.2.4.2 Selektion des zu bearbeitenden Bug Reports

Die Modularität der Apache-Produkte stellt eine wichtige Eigenschaft für den Prozess der Selektion von zu bearbeitenden Problembereichen dar, da dadurch die Software in voneinander abgrenzbare Problemdomänen aufgeteilt wird: "The Apache software architecture is designed to separate the core functionality of the server, which every site needs, from the features, which are

located in modules that can be selectively compiled and configured." [MoFiHe00]

Die Selektion eines zu bearbeitenden Problems erfolgt i.d.R. ausschließlich durch den Entwickler, der über die verschiedenen Metadaten nach für ihn relevanten Bug Reports recherchiert: "Developers scan Bugzilla and decide which of these items they would like to implement." [AtGaGe02]. Dabei tendieren nach [MoFiHe00] die meisten Entwickler dazu, Quellcode zu bearbeiten, der ihnen bekannt ist und den sie bereits bearbeitet haben. Entwickler, welche noch keinerlei Patch entwickelt haben, implementieren im Gegensatz dazu häufig neue Funktionalitäten und neue Module. Diese beiden Tendenzen führen dazu, dass in manchen Bereiche die sogenannte Code Ownership realisiert wurde, durch die einige Module eindeutig den Verantwortungsbereichen spezifischer Entwickler zugeordnet werden können.

#### 3.2.4.3 Lokale Entwicklung

Wenn ein Entwickler einen Bug Report identifiziert und den Quellcode gemäß der entsprechenden Prozessbeschreibung bezogen hat, kommuniziert er zu Beginn der lokalen Entwicklung häufig mit dem Initiator des Bug Report: "Once a developer has committed to coding the new enhancement, he/she interacts with the submitter to sort out any details." [AtGaGe02]

Anschließend erstellt der Entwickler einen lokalen Verzeichnisbaum der ausschließlich zur Entwicklung verwendet wird und ein separates Quellcodeverzeichnis, in welchem stets der aktuelle und unveränderte Quellcode des Projekts oder Moduls verwaltet wird. Im Entwicklungspfad werden die Modifikationen am Quellcode vorgenommen und diese in einem iterativen Prozess lokal entwickelt, kompiliert und getestet. Dieses Environment stellt die Voraussetzung für die spätere Extraktion des Patches mit dem Diff-Werkzeug dar.

[AtGaGe02] unterscheidet hierbei zwischen zwei Arten von Patches:

- Major Changes
- Minor Changes

Minor Changes repräsentieren nach [AtGaGe02] einfache Patches zur Fehlerbehebung oder neue Funktionalitäten, die keine nennenswerten Interdependenzen zu bestehenden Modulen beinhalten, während Major Changes das Gesamtsystem in großem Maße beeinflussen, bestehende API verändern oder allgemein ein Risiko für die gesamtheitliche Systemstabilität darstellen: "Minor changes encompass simple patches to fix bugs, or minor changes to the code that affect little part of the functionality. On the other hand, major changes are new features, large-scale changes that can affect the

semantics of an existing API function, program size or data formats." [AtGaGe02]

Der Prozess des Softwareentwurfs findet in dieser Phase der Entwicklung kaum explizite Berücksichtigung und wird i.d.R. als der Implementation immanenter Teilprozess ausgeführt: "Design is largely done by the contributor of the code, when the feature is small or when just one person is developing a large feature to be submitted latter to the project." [AtGaGe02]. Im Falle von Major Changes wird der zu implementierende Entwurf im Vorfeld der Realisierung durch die Entwicklergemeinde diskutiert und evaluiert: "But when a major piece of functionality is to be developed, committers and developers will exchange ideas and comments on how to go about implementing a major feature." [AtGaGe02]. Zur Entwicklung werden sehr diverse Werkzeuge durch die Entwickler instrumentalisiert: "In the Jakarta project, developers and committers make code changes by using different tools (such as text editors, IDEs, etc)." [AtGaGe02]

Der entwickelte Quellcode sollte dabei den Richtlinien des Apache-Projekts entsprechen: "It is also worth making sure your code follows the Apache style, as described by the GNU indent flags in the .indent.pro file in every source directory." [HttpdApOrg02][59] Weiterhin sollten die allgemeinen Kriterien der Softwarequalität beachtet und berücksichtigt werden: "We also have very high expectations for code quality; and to us this means the avoidance of excessive static buffers, using the memory pool mechanism (which ensures proper cleanup), and otherwise writing thread-safe code." [HttpdApOrg02][60]

#### 3.2.4.4 Format des Patches

Im Anschluss an die lokale Entwicklung kann das Werkzeug diff dazu verwendet werden, ein Patch basierend auf den beiden lokalen Quellcodeverzeichnissen zu extrahieren: "The patch should be created by using the `diff -u` command from the original software file(s) to the modified software file(s)." [JakartaApOrg02][61]. `Diff` generiert dabei die eine Ascii-Text-Datei, die lediglich die Modifikationen des Ausgangsquellcodes beinhaltet.

Wichtig ist hierbei, dass das Patch basierend auf der aktuellsten Version des Quellcodes entwickelt wurde, um somit etwaige Konflikte zu verhindern : "It is recommended that you submit patches against the latest CVS versions of the software in order to avoid conflicts." [JakartaApOrg02][61]

59 http://httpd.apache.org/dev/devnotes.html
60 http://httpd.apache.org/dev/patches.html
61 http://jakarta.apache.org/site/source.html

Obwohl Werkzeuge wie `diff` teilweise lediglich auf Unix- bzw. Linux-Plattformen zur Verfügung stehen, ist es Windows-Anwendern möglich, diese Werkzeuge durch entsprechende Bash-Shell-Emulationen trotzdem lokal zu verwenden, da sie einen elementaren Bestandteil der OSS-Entwicklung bilden: "...you can install `Cygwin` which will enable you to use the bash shell and also installs a lot of other utilities (such as `diff` and `patch`) that will turn your PC into a virtual Unix machine." [JakartaApOrg02][62]

#### 3.2.4.5 Lizensierung des Patches

Zentrales Element in allen Teilprojekten der ASF ist die Apache Software License (ASL), welche die strategischen Rahmenbedingungen für die Softwareentwicklung im Apache-Kontext definiert. Neben der ASL können aber auch verwandte oder abgeleitete Lizenzen, wie z.B. die Mozilla Public License, durch die Entwickler verwendet werden: "All source code committed to the Project's repositories must be covered by the Apache License or contain a copyright and license that allows redistribution under the same conditions as the Apache License." [JakartaApOrg02][62]. Sobald ein Quellcode-Artefakt durch einen Entwickler modifiziert wird, müssen auch die entsprechenden Copyright- bzw. Lizenz-Informationen der Datei aktualisiert werden: "Committers should update the copyright notice on the Apache License to include the current year when they revise a source file." [JakartaApOrg02][62]. Sofern ein Quellcode-Artefakt den projekteigenen Quellcodeverzeichnissen, z.B. zu Demonstrationszwecken hinzugefügt wird, aber nicht vollständig unter der ASL verfügbar sein soll, muss dies eindeutig durch den Entwickler signalisiert werden: "Any code, document, or binary that is committed to the Project's repositories, but not being donated to the ASF, must be clearly marked as such." [JakartaApOrg02][62]

#### 3.2.4.6 Kommunikation des Patches

Sofern der Entwickler nicht über die Privilegien zur direkten Integration des Quellcodes verfügt oder das Patch erst einem Review durch die Entwicklergemeinde unterziehen möchte, kann die Kommunikation eines Patches an die Community auf verschiedenen Wegen erfolgen: "The developer then posts the patch in the form of a diff-file to the mailing list dedicated to new patches (i.e. new-httpd), or by submitting the diff-file to the Bugzilla database." [AtGaGe02]

62 http://jakarta.apache.org/site/source.html

Das Patch wird dazu an eine dedizierte Mailing Liste für entwicklungsrelevante Themen gesendet und durch einen eindeutigen Eintrag in dem Betreffattribut der Nachricht als Patch markiert: "If you are a subscriber to dev@httpd.apache.org, you can simply post your patch there, with the string '[PATCH]' prefixing your subject line, so everyone knows it's a patch." [HttpdApOrg02][63]. Da verschiedene Nachrichten in den sehr diffusen Informationen der Mailing Listen häufig nicht die nötige Beachtung finden, kann es nötig sein, diesen Arbeitsschritt mehrfach auszuführen, bis der Entwickler ein entsprechendes Feedback durch die Entwicklergemeinde erhalten hat: "Feel free to resubmit your patch in a week or so if you have not yet received any feedback from the developers' group." [HttpdApOrg02][63]. Im Falle von sehr umfangreichen Patches werden diese häufig auch auf einem öffentlichen Webserver publiziert und lediglich dessen URL an die Entwicklergemeinde kommuniziert.

Neben den Entwicklungsspezifischen Mailing Listen, stellt auch die Bugzilla-Datenbank einen alternativen Kommunikationskanal zur Verteilung von Patches dar, zumal hier auch die Möglichkeit besteht, ein Patch direkt dem zugrundeliegenden Bug Report zuzuordnen: "Another good way to make sure it gets on our plate is to submit it through the bug database, at http://bugs.apache.org/, marking it as a 'change-request'." [HttpdApOrg02][63]. Diese Zuordnung wird über das Attachment-Attribut eines Bug Reports ermöglicht, welches die direkte Assoziation von externen Dateien mit einem Bug Report ermöglicht und somit z.B. auch ein Patch beinhalten kann.

Falls das Patch bedingt durch den Review-Prozess überarbeitet wird, muss jedesmal die vollständige Version des Patches kommuniziert werden: "If later modification to the patch proves necessary, the entire new patch should be posted and not just the difference between the two patches." [JakartaApOrg02][64]

### 3.2.5 Integration von Quellcode

Der Quellcode der Teilprojekte der ASF wird in dedizierten CVS-Archiven verwaltet, wobei der schreibende Zugriff auf die projektinternen Quellcode-Repositories nur wenigen privilegierten Entwicklern eingeräumt wird und somit auch nur diese in der Lage sind, Patches zu integrieren: "The Project's codebase is maintained in shared information repositories using CVS on the dev.apache.org machine. Only Committers have write access to these repositories." [JakartaApOrg02][64]

63 http://httpd.apache.org/dev/patches.html
64 http://jakarta.apache.org/site/source.html

Die Integration eines Patches in das aktuelle Quellcodeverzeichnis sagt dabei aber noch nichts über eine spätere Integration in ein offizielles Softwarerelease aus und ist daher von dieser strikt abzugrenzen. Eine Integration eines Patches auf dieser Ebene bedeutet lediglich eine Integration des erstellten Quellcodes in den aktuell zu Entwicklungszwecken verwendeten Quellcodeverzeichnisbaum.

Generell ist davon auszugehen, dass externe Module, die keinen Einfluss auf den Programmkern eines Apache-Produkts aufweisen, weitaus häufiger in die Quellcodeverzeichnisse integriert werden, als Patches, welche direkte Abhängigkeiten zu bestehenden Kernfunktionalitäten besitzen: "Be aware that the core developers tend to be very conservative about what makes it into the core of Apache. Apache has a very flexible API, and any advanced functionality is likely to be pushed to be a third-party module." [HttpdApOrg02][65]

#### 3.2.5.1 Review und Commit

In diesem Abschnitt werden die Aktivitäten zum Review und Commit von Patches und die verschiedenen diesbezüglich existierenden Strategien dargestellt.

##### 3.2.5.1.1 Review

Der Prozess des Review basiert auf der projektweiten Evaluierung eines Patches durch die Entwickler, welche das Patch über die etablierten Kommunikationskanäle diskutieren und seine Korrektheit verifizieren: "In this process, developers and committers communicate to verify that the code has been successfully updated to address the bug or to implement the feature request or area of functionality that has changed." [AtGaGe02]. Dazu werden der Quellcode und u.u. zusätzliche Dokumentation und Kommentare über die Mailing Liste versendet und anschließend das Feedback der Entwickler diskutiert und möglichst ein Konsens erreicht. Falls der Quellcode bereits basierend auf der CTR-Strategie "Commit Then Review" (vgl. Kapitel 3.2.5.1.3) integriert wurde, werden die so bereits zentral verfügbaren Quellcodeartefakte evaluiert und gegebenenfalls modifiziert.

Im Rahmen des Reviews kommen verschiedene teilprojektspezifische Abstimmungsverfahren zum Einsatz. Diese Abstimmungsverfahren sind außerdem stark von der jeweiligen Projektphase und der Art und dem Umfang des Patches abhängig. Prinzipiell ist jede Person stimmberechtigt, wobei aber nur die Stimmen bewertet werden, die von aktiven Entwicklern abgegeben

65 http://httpd.apache.org/dev/patches.html

werden: "Anyone on the mailing list can vote on a particular issue, but we only count those made by active members or people who are known to be experts on that part of the server." [HttpdApOrg02][66]

Im HTTPD-Kontext muss eine einstimmige Akzeptanz existieren um ein Patch zu integrieren, wenn sich das Projekt gerade in der Phase des Software-Release befindet: "Changes to the code are proposed on the mailing list and usually voted on by active members -- three +1 (yes votes) and no -1 (no votes, or vetoes) are needed to commit a code change during a release cycle; [...]" [HttpdApOrg02]. Auch im Jakarta-Projekt identifiziert [AtGaGe02], dass eine einstimmige Annahme eines Patches vorhanden sein muss, um als Bestandteil des Quellcodeverzeichnisses integriert zu werden. Falls ein fundiert begründetes Veto nicht durch den Beitrag eines ergänzenden Patches entkräftet werden kann, muss es schnellstmöglich aus dem Quellcodeverzeichnis extrahiert werden, sofern es bereits eingecheckt wurde, oder kann dem Prozess des Commit gar nicht erst zugeführt werden: "A committed change must be reversed if it is vetoed by one of the voting members and the veto conditions cannot be immediately satisfied by the equivalent of a "bug fix" commit." [JakartaApOrg02][67]

Ein Veto kann nur durch den Urheber des Einwurfs widerrufen werden.

#### 3.2.5.1.2 Commit

Der Commit von Patches in den zentralen Entwicklungsquellcode kann nur durch entsprechend privilegierte Entwickler erfolgen: "There is a group of people who have logins on the development server and access to the CVS repository." [Sandred01]

Anforderungen an die Integration von Patches sind nach [JakartaApOrg02][67] die Verbindung von logisch bzw. funktional zusammenhängenden Patches und deren gemeinsame Integration und die erfolgreiche lokale Kompilation des Quellcodes durch den Entwickler: "Related changes should be committed as a group, or very closely together. [...] All code changes must be successfully compiled on the developer's platform before being committed.". Dadurch soll gewährleistet werden, dass auch der zentrale Quellcodeverzeichnisbaum jederzeit erfolgreich kompiliert werden kann.

Jedes Quellcodeverzeichnis der verschiedenen Apache-Teilprojekte enthält eine STATUS-Datei, welche aktuelle Pläne und Vorhaben dokumentiert und

66 http://httpd.apache.org/ABOUT_APACHE.html
67 http://jakarta.apache.org/site/source.html

außerdem durch den Committer zur Kommunikation des Checkins genutzt wird: "Committers can use the STATUS file in the repository, to summarize the code changes submitted since the last release." [AtGaGe02]. Der Committer dokumentiert die Integration eines Patches außerdem durch eine Commit-Message und kommuniziert den Checkin weiterhin an die relevanten Mailing Listen und den Entwickler des Patches: "When submitting patches/code changes to CVS, the person who checked in the patch should send a message to the person who contributed with the patch, as well as the mailing list, to specify that the patch has been submitted, and avoid source code/patch conflicts." [AtGaGe02]

Jeder Developer hat die Möglichkeit, durch eine langfristige, aktive Teilnahme am Projekt seine Kompetenz und Erfahrung zu demonstrieren und infolgedessen mit Commit-Rechten privilegiert zu werden: "Committers have shown that they are competent developers with an understanding of how to write good code. Most committers have a good understanding of software process and best practices in developing software." [AtGaGe02]

#### 3.2.5.1.3 Review- und Commit-Strategien

Bezüglich Review und Commit des Patches werden in den Projekten der ASF zwei Strategien angewandt:

- Review Then Commit (RTC)
- Commit Then Review (CTR)

**Review Then Commit (RTC)**

Bei dieser Variante wird das Patch an die Mailing Liste der Entwickler gesendet, durch die Adressaten der Mailing Liste evaluiert, diskutiert bzw. gegebenenfalls modifiziert und anschließend durch ein entsprechend privilegiertes Mitglied der Gemeinde dem aktuellen Entwicklungsverzeichnisbaum hinzugefügt. Nach [MoFiHe00] wird der Review primär durch die sogenannten Core Developer ausgeführt, wobei diese Aufgabe aber auch durch jeden Adressat des Patches wahrgenommen werden kann. Der RTC-Prozess wurde im HTTPD-Kontext früher weitgehend ausschließlich angewandt: "In the past, committers followed a review-then-commit (RTC) policy." [AtGaGe02]

Im Jakarta-Kontext werden alle komplexen Patches, respektive die sogenannten Major Changes, ausschließlich durch die RTC-Strategie integriert, um dadurch die Qualitätssicherung zu gewährleisten: "Doubtful changes, new features, and large scale overhauls need to be discussed before committing them into the repository. Any change that affects the semantics of an existing API function, the size of the program, configuration data formats, or other major areas must receive consensus approval before being committed." [JakartaApOrg02][68]

**Commit Then Review (CTR)**

Aufgrund der ständig wachsenden Anzahl von Patches im HTTPD-Projekt, wird inzwischen häufig der CTR-Prozess praktiziert: "... due to the large amount of patches submitted currently, committers follow a commit-then-review (CTR) policy for patches." [AtGaGe02]. Dabei wird das Patch direkt in den Entwicklungsquellcode integriert und anschließend das Feedback der Entwickler und Reviewer erwartet. Lediglich eine negative Stimme ist im HTTPD-Kontext notwendig, um das Patch wieder zu extrahieren: "The CTR policy is subject to the lazy consensus rule. They commit the patch and then see if any other committers dispute the patch by sending in a -1 vote or a veto vote. Only one veto is needed to cause a patch to be revoked." [AtGaGe02].

Im Kontext des Jakarta-Projekts findet die CTR-Strategie vor allem für die Integration der sogenannten Minor Changes Anwendung, was auch durch die Projekt Guidelines legitimiert wird: "Simple patches to fix bugs can be committed then reviewed." [JakartaApOrg02][68]

### 3.2.6 Integration von Dokumentation

Dokumentationsartefakte werden analog zum Quellcode teilweise in CVS verwaltet und können ebenfalls nur durch entsprechende privilegierte Committer integriert werden. Der diesbezüglich Reviewprozess unterliegt i.d.R. der CTR-Strategie: "[...] docs are usually committed first and then changed as needed, with conflicts resolved by majority vote." [HttpdApOrg02][69]

### 3.2.7 Software Test basierend auf Test Releases

Im Kontext der Jakarta-Teilprojekte wird der Prozess der Nightly Builds, welche u.a. für Testzwecke verwendet werden, in einer ähnlichen Form wie im Rahmen des Mozilla-Projekts praktiziert: "Every night, a new build is created, that includes the latest code for the day. These nightly builds are very unstable, but they can be used for further testing." [AtGaGe02]. Der hier zugrundeliegende Prozess orientiert sich stark am Mozilla-Projekt und wird daher an dieser Stelle nicht noch einmal explizit dargestellt.

Im Rahmen des HTTPD-Projekts konnte ein solcher Test-Prozess nicht identifiziert werden, da lediglich verschiedene stabile Versionen der Software in binärer Form veröffentlicht werden. Dafür wurden Test Releases (Alpha- und

68 http://jakarta.apache.org/site/source.html
69 http://httpd.apache.org/ABOUT_APACHE.html

Beta Release) identifiziert, welche eine ähnliche Intention wie die Nightly Builds verfolgen, aber in einer anderen Frequenz erstellt werden. Der Prozess der kontinuierlichen Software-Tests wird daher in diesem Rahmen in erster Linie individuell durch die Entwickler ausgeführt, welche sich dafür entscheiden, eine aktuelle Alpha- oder Beta-Version des Quellcodes zu beziehen und mit dieser zu arbeiten.

Die Veröffentlichungsprozesse im Rahmen des Release von Testversionen der Software werden im Abschnitt 'Managementprozesse' in Kapitel 3.3 explizit erläutert.

### 3.2.8 Support

Die Anwenderbetreuung für Softwareprodukte der ASF basiert auf sehr diversen und heterogenen Optionen, wobei kein offizieller Support durch die ASF zu Verfügung gestellt wird: "Apache has no official software support organization. ... However, this does not mean that there is no support for Apache web server." [Jordan01]. Anwender können Informationen bezüglich anwendungsbezogener Problemstellungen in diversen Informationsquellen recherchieren:

- Mailing Listen und Mailing Listen Archive
- Newsgroups
- Explizite Dokumentationsartefakte
- Websites
- Webbasierte Anwenderforen
- Sekundäre Informationsquellen wie produktrelevante Literatur, Magazine oder Webseiten (z.B. www.apacheweek.com)

Die teilprojektspezifischen Anwender Mailing Listen stellen die primären Informationsquellen bezüglich des Anwendersupports dar und werden hauptsächlich durch die Anwender der Software frequentiert und genutzt: "Although some developers may participate in the list, the primary interaction is user-to-user." [HttpdApOrg02][70]. Durch die große Bandbreite an Informationsquellen und den Umfang der Entwickler- und Projektgemeinde können i.d.R. alle Anwender- und Entwicklungprobleme gelöst werden. Neben der Recherche in den Informationsquellen ist es auch möglich Problemstellungen über die verschiedenen Kommunikationskanäle zu transportieren, um einen entsprechenden Lösungsvorschlag durch einen Akteur der Community zu erhalten: "Given the large number of users of the Apache web server, it is very likely that one of the on-line sources of information will contain an answer or will have a contributor that is willing to post an answer to

70 http://httpd.apache.org/userslist.html

a question." [Jordan01]. Neben diesen Informationsquellen stehen für Entwickler verschiedene weitere entwicklungsspezifische Informationsquellen zur Verfügung, wie z.B. dedizierte Entwickler-Mailing Listen o.ä..

Weiterhin steht für weitreichend etablierte Softwareprodukte der ASF wie z.B. den Apache HTTPD-Web Server häufig auch Support durch verschiedene kommerzielle Dienstleister zur Verfügung: "There are a number of companies, including IBM, which will support the Apache web server for a fee." [Jordan01]

## 3.3 Managementprozesse

In diesem Abschnitt werden alle Prozesse dargestellt, die primär der Koordination, Steuerung und Unterstützung der Entwicklungsprozesse dienen und dem Metamodell entsprechend als Managementprozesse klassifiziert werden können. Aufgrund der beschränkten Recherchemöglichkeiten im Rahmen der zentral ausgeführten Managementprozesse, die deswegen eine Partizipation und Einsicht nicht direkt ermöglichen, können diese Prozesse z.T. nur auf einem sehr hohen Abstraktionsniveau dargestellt werden.

### 3.3.1 Initiale Prototypentwicklung und -release

Aufgrund der Tatsache, dass die initiale Entwicklung und der Release des HTTPD-Servers die Initialisierung des gesamten Apache-Projekts repräsentiert, wird im Rahmen dieser Prozessbetrachtung lediglich dieser Prozess untersucht und auf die Analyse der initialen Prozesse der verschiedenen, später integrierten Teilprojekte verzichtet. Zudem werden in diesem Abschnitt verschiedene Entwicklungs- und Veröffentlichungsaktivitäten zusammengefasst, welche in ihrer Gesamtheit einen mehrstufigen Veröffentlichungsprozess repräsentieren, der zwar nicht mehr vollständig in einem geschlossenen Kontext aber durch eine begrenzte Gruppe weniger Akteure und nach einem eher traditionellen Modell ausgeführt wurde.

Der Apache HTTPD-Server basiert auf einem HTTP-Server, welcher von Rob McCool am National Center for Supercomputing Applications an der University of Illinois entwickelt wurde: "Apache began in February 1995 as a combined effort to coordinate existing fixes to the NCSA httpd program developed by Rob McCool." [MoFiHe00]. Diese Software stellte nach [HttpdApOrg02] bereits im Jahr 1995 die populärste Server Software dar, deren zentrale Entwicklung aber nach McCools Austritt aus dem NCSA weitgehend eingestellt wurde, was dazu führte, dass viele Webmaster das System durch eigene Patches individuell verbesserten: "However, development of that httpd had stalled after Rob left NCSA in mid-1994, and many webmasters had developed their own extensions and bug fixes that were in need of a common distribution."

Einige dieser Entwickler, welche sich über E-Mail-Kommunikation zur Koordination ihrer Patches kennengelernt hatten, schlossen sich zusammen und stellten eine rudimentäre, zentrale Infrastruktur zur Weiterentwicklung der Software bereit: "A small group of these webmasters, contacted via private e-mail, gathered together for the purpose of coordinating their changes (in the form of "patches"). Brian Behlendorf and Cliff Skolnick put together a mailing list, shared information space, and logins for the core developers on a machine in the California Bay Area, with bandwidth donated by HotWired." [HttpdApOrg02][71]. Im Februar 1995 bestand diese initiale Apache Group aus den folgenden acht Mitgliedern:

- Brian Behlendorf
- Roy T. Fielding
- Rob Hartill
- Cliff Skolnick
- David Robinson
- Robert S. Thau
- Andrew Wilson
- Randy Terbush

Diese Entwicklergruppe benutzte das letzte Release des HTTP-Servers der NCSA und integrierte alle bereits verfügbaren Patches und Bug Fixes, unterzog die Software ersten Tests und veröffentlichte die dadurch entstandene verbesserte Version der Software als Apache Server: "Using NCSA httpd 1.3 as a base, we added all of the published bug fixes and worthwhile enhancements we could find, tested the result on our own servers, and made the first official public release (0.6.2) of the Apache server in April 1995." [HttpdApOrg02][71]

Das NCSA begann in derselben Periode erneut mit der Arbeit am HTTPD-Server, welche aber mit dem neu entstandenen Apache-Projekt koordiniert wurde. Einige zentrale Entwickler des NCSA-Teams integrierten sich ebenfalls in die Apache-Community: "By coincidence, NCSA restarted their own development during the same period, and Brandon Long and Beth Frank of the NCSA Server Development Team joined the list in March as honorary members so that the two projects could share ideas and fixes." [HttpdApOrg02][71]

Obwohl der Server bereits auf weltweite Akzeptanz stieß und das Projekt auf eine kontinuierlich anwachsende Entwickler- und Anwendergemeinde verweisen konnte, beschloss die Apache Group die Überarbeitung der aktuellen Serverarchitektur: "During May-June 1995, while Rob Hartill and the rest of the group focused on implementing new features for 0.7.x (like pre-forked child processes) and supporting the rapidly growing Apache user community, Robert Thau designed a new server architecture (code-named

71 http://httpd.apache.org/ABOUT_APACHE.html

Shambhala) which included a modular structure and API for better extensibility, pool-based memory allocation, and an adaptive pre-forking process model." [HttpdApOrg02][72]. Diese wurde als Version 0.8.8 implementiert und veröffentlicht.

Nach einer umfangreichen Beta-Testphase und der Portierung auf eine Vielzahl von Plattformen wurde die Version 1.0 im Dezember 1995 veröffentlicht: "After extensive beta testing, many ports to obscure platforms, a new set of documentation (by David Robinson), and the addition of many features in the form of our standard modules, Apache 1.0 was released on December 1, 1995." [HttpdApOrg02][72]

### 3.3.2 Release Planung

Die Planung eines Software Release im Rahmen der OSS-Entwicklung des Apache-Projekts wird primär durch die Committer und Core Developer bzw. den Maintainer des Projekts wahrgenommen: "Committers develop release plans upon which they vote in order to determine what will be contained in a build release." [AtGaGe02]. Ziel hierbei ist es, genau zu definieren, welche Funktionalitäten und Patches in ein 'Milestone'- bzw. 'Release-Build' bei Jakarta oder ein 'General Availability Release' im HTTPD-Projekt integriert werden. Diese Entscheidungen werden maßgeblich durch das Project Management Committee des jeweiligen Teilprojekts mitbestimmt, wobei die endgültige Entscheidungsgewalt dem Release Manager obliegt: "Despite what the Project Management Committee's recommendation is, the final decision of what will be released is made by the release manager." [AtGaGe02]

Das Ergebnis dieser Planungsphase ist das Artefakt Release Plan, welches im Kapitel 3.7.1.2.3 näher beschrieben wird.

Weiterhin werden durch die Committer und den Release Manager sogenannte Showstopper definiert, wie der Release-Guide des HTTPD-Projekts beschreibt: "An item being denoted as a showstopper indicates that the group has come to a consensus that no further releases can be made until the entry is resolved. These items may be bugs, outstanding vetos that have not yet been resolved, or enhancements that must make it into the release." [HttpdApOrg02][73]

72 http://httpd.apache.org/ABOUT_APACHE.html
73 http://httpd.apache.org/dev/release.html

### 3.3.3 Release Prozess

Der Release einer Software-Version stellt eine der fundamentalen Bedingungen für die Durchführung der Entwicklungsprozesse dar: "Therefore, as a prerequisite for the download procedure, source and binary code is made available via the Jakarta website [...]" [AtGaGe02]

Dieser Abschnitt beschreibt die allgemeinen Merkmale des Veröffentlichungsprozesses im Kontext des Apache-Projekts, der in dieser Form auf alle Klassen von Software Releases anwendbar ist. Die konkreten Merkmale des Release von Softwareversionen zu Testzwecken werden in einem expliziten Abschnitt erläutert.

In Anlehnung an [AtGaGe02] können verschiedene Phasen der Release-Prozesse im Rahmen des HTTPD-Projekts identifiziert werden:

1. Design des Release: gleichbedeutend mit der Release Planung
2. Entwicklungsphase zur Entwicklung des Quellcodes zur Realisierung des Release Plans
3. Alpha-Phase: Alle Funktionalitäten wurden implementiert, aber der Code ist noch instabil und ungetestet
4. Beta-Phase: Alle komplexen Softwarefehler wurden behoben und eine erste Testphase wurde durchgeführt. In dieser Phase werden alle Entwickler aufgefordert, ausschließlich dieses Release zu verwenden, um dabei das Produkt weiteren Tests zu unterziehen.
5. Release: Das Produkt wird als stabil angesehen und in binärer Form und als Quellcode auf der Website des Projekts als stabile neue Version veröffentlicht

Im Anschluss an die Alpha-Phase erfolgt i.d.R. ein "Feature Freeze" und im Anschluss an die Beta-Phase ein "Code Freeze", wobei auf diese beiden integralen Bestandteile des Release-Prozesses in Kapitel 3.3.3.1 noch detailliert eingegangen wird.

Der Release-Prozess wird i.d.R. durch eine dedizierte Rolle, die des Release Managers (RM), gesteuert und ausgeführt: "When the project nears a product release, one of the core developers volunteers to be the release manager, responsible for identifying the critical problems (if any) that prevent the release, determining when those problems have been repaired and the software has reached a stable point, and controlling access to the repository so that developers don't inadvertently change things that should not be changed just prior to the release." [MoFiHe00]. Der Release Manager ist vor allem dafür zuständig, die Bearbeitung von relevanten Bug Reports zu forcieren bzw. sicherzustellen und die Umsetzung der zeitlichen Terminierung des Release Plans zu gewährleisten. Außerdem trifft der Release Manager die Entscheidung,

wann die nächste Phase des Release erreicht und initiiert werden kann und realisiert die Schließung des Quellcodes im Rahmen des Feature- und Code Freeze.

Die Rolle des Release Managers wird i.d.R. von verschiedenen Committern wahrgenommen, wobei kein Akteur explizit als RM legitimiert wurde, sondern diese Rolle informell durch die Committer zugeordnet wird: "Since this job requires coordination of the development community (and access to CVS), only committers to the project can be RM. However, there is no set RM." [HttpdApOrg02]. Der Release Manager verfügt bezüglich eines speziellen Release über die oberste Entscheidungsgewalt und stellt somit in diesem Kontext die höchste Autorität dar.

#### 3.3.3.1 Feature - und Code Freeze

Die Schließung des Quellcodearchivs für die Integration neuer Patches wird als Freeze bezeichnet, wobei wie auch im Apache-Kontext zwischen dem Feature Freeze und dem restriktiveren Code Freeze unterschieden wird.

Der Feature Freeze bezeichnet dabei die Schließung des Projekts für das Hinzufügen zusätzlicher Funktionalität, um eine Destabilisierung zu unterbinden und lediglich noch Bug Fixes in den Quellcode aufzunehmen: "Also, there are often times when the core developers are in 'feature freeze', when they are trying to iron out the remaining bugs in the code in preparation for a release." [HttpdApOrg02][74]

Im Gegensatz dazu unterbindet der Code Freeze die Integration von jeglichen Quellcodeänderungen, sofern diese nicht der Behebung von fundamentalen Softwarefehlern dienen: "Once committers have decided that a new build can be considered for final release, a committer will make the build and send a message to all committers, to indicate that no changes are to be made to the repository for a certain period of time [...]. This is known as 'code freeze'." [AtGaGe02]

Da dieser Aspekt bereits im Rahmen der Mozilla- und der Linux-Projektstudie dargestellt wurde, wird dies an dieser Stelle nicht eingehender thematisiert.

74 Apache02, http://httpd.apache.org/dev/patches.html

#### 3.3.3.2 Branching

Um die weitere Entwicklung durch den Release Prozess nicht zu behindern, können Branches im Rahmen des Veröffentlichungsprozesses generiert werden. Dadurch können die einzelnen Phasen des Release-Prozesses durchlaufen werden, während die Entwicklung des Quellcodes weiterhin in einem separaten Quellcode-Branch erfolgt. Dabei ist es von hoher Priorität, dass die verschiedenen Quellcodezweige anschließend wieder zu einem Verzeichnis integriert werden: "Groups are allowed to create a branch for release cycles, etc. They are expected to merge completely back with the main branch as soon as their release cycle is complete." [JakartaApOrg02][75]

In der Praxis scheint diese Methode aber von geringer Bedeutung zu sein und wurde von [AtGaGe02] lediglich im Rahmen des Teilprojektes Jakarta Lucene als mögliche Handlungsalternative identifiziert. In diesem Zusammenhang scheinen der Feature- und Code Freeze von größerer Bedeutung zu sein.

#### 3.3.3.3 Alpha- und Beta Releases

Software Tests finden ständig basierend auf dem aktuellen Quellcodeverzeichnis statt. Im Rahmen des Release-Prozesses werden diese Tests durch explizite Release-Tests basierend auf den erstellten Builds unterstützt.

**Alpha Release**

Ein Alpha Release dient lediglich den Entwicklern zu Testzwecken und ist nicht für den allgemeinen Einsatz verwendbar: "Alpha indicates that the release is not meant for mainstream usage or may have serious problems that prohibits its use." [HttpdApOrg02][76]. Dieses Release wird den Entwicklern und Committern über einen speziellen Bereich zur Verfügung gestellt: "Developers and committers have access to the Alpha release distribution for testing purposes." [AtGaGe02]. Im Rahmen der Alpha-Testphase wird i.d.R. ein Feature Freeze ausgeführt. Die Alpha Releases werden auf der Entwickler-Webseite der Projekte veröffentlicht und teilweise durch die Veröffentlichung von Alpha-Builds ergänzt. Außerdem wird ein Alpha-Release über die relevanten Mailing Listen bekanntgegeben: "This release is provided on the developers' website and announced through the developers' mailing list, where users do not have access to." [AtGaGe02]. Entdeckte Fehler werden durch die Entwickler

75 http://jakarta.apache.org/site/source.html
76 http://httpd.apache.org/dev/release.html

bearbeitet und auf die konventionelle Weise als Patch kommuniziert, um den Code zu stabilisieren.

**Beta Release**

Wenn der Release Manager mit der Qualität des Alpha-Release zufrieden ist, wird das Quellcodearchiv einem Code Freeze unterzogen und der Release Prozess geht damit in die Beta-Phase über. Das Beta Release wird den Entwicklern zur Verfügung gestellt, welche es weiteren Tests unterziehen und letzte Patches beitragen bis der Release Manager das Beta-Release als stabil definiert: "They test and fix bugs until the release manager decides that the code is finally ready to become a general availability (GA) release." [AtGaGe02]. Nachdem diese Phase abgeschlossen wurde, wird die Binärdistribution des Softwareprodukts häufig letzten Tests durch die ASF in einem Zeitraum von zwei bis drei Tagen unterzogen.

Diese prototypischen Prozessphasen stellen einen empfohlenen Ablauf dar, der in den einzelnen Teilprojekten individuell und marginal verschieden praktiziert und vom Release Manager maßgeblich determiniert wird: "This is only a suggested guideline and again the release manager has the ultimate control over when the new release is ready and will be available." [AtGaGe02]

#### 3.3.3.4 Release einer offiziellen Version

Sofern die verschiedenen Testprozesse durchgeführt wurden, kann die Veröffentlichung eines neuen Software Release erfolgen.

In Anlehnung an den Release Guide des HTTPD-Projekts[77] kann die Vorgehensweise zur Veröffentlichung eines getesteten Release als sogenanntes "General Availability Release" wie folgt beschrieben werden:

1. Integration der PGP-Schlüssel des Release Managers in das sogenannte KEYS-File im Distributionsverzeichnis des Projekts
2. Generierung eines Packages, welches alle relevanten Quellcode-Dateien und nötige Dokumentationen enthält
3. Kompilierung des Quellcodes für die zu unterstützenden Plattformen und Paketierung der entstehenden Dateien zu plattformspezifischen Paketen
4. Veröffentlichung des Quellcodes und der Binärdateien als neue Softwareversion auf der allgemeinen Projekt-Website

77 http://httpd.apache.org/dev/release.html

5. Kommunikation des Release über die allgemeinen Kommunikationskanäle

Der Arbeitsschritt 3 zur Generierung der binären Produktversionen ist nicht in allen Teilprojekten erforderlich: "The only files that are required for a public release are the source tarballs (`.tar.Z`, `.tar.gz`)." [HttpdApOrg02][78]

#### 3.3.3.4.1 Veröffentlichung und Kommunikation des Release

Die Veröffentlichung des offiziellen Softwarerelease erfolgt sowohl auf den projektspezifischen Seiten als auch im Softwareverzeichnis der Website der ASF: "Once the release has reached the highest-available designation (as deemed by the RM), the release can be moved to the httpd distribution directory on apache.org." [HttpdApOrg02][78]

Nach einer Frist von ca. zwei bis drei Tagen, in der das Release letzten Tests durch die Entwickler unterzogen wurde und auf den Mirror-Websites der Projekte veröffentlicht wird, kann das neue Release über eine öffentliche Ankündigung auch an die Anwender kommuniziert werden, was primär über die Mailing Listen und die Website erfolgt. Eine Option, um ein Release Apache-weit und somit nicht nur über teilprojektspezifische Kanäle zu kommunizieren, ist die Mailing Liste announce@apache.org.

### 3.3.3.5 Weitere Fehlerbehebung

Nach der Veröffentlichung des Release können weitere identifizierte Softwarefehler nur durch die Behebung in einem späteren Release oder durch die Veröffentlichung eines Patches behandelt werden: "If a problem is found, it will have to be addressed in the next release or a patch can be made available." [HttpdApOrg02][78]. Daher sollte ein Softwarerelease vor der Veröffentlichung möglichst umfangreichen Tests durch eine große Community unterzogen werden.

## 3.3.4 Definition teilprojektweiter Regularien

Projektweite Richtlinien werden durch die Management-Instanzen der Teilprojekte definiert und stellen die Basis für die Reglementierung und Koordinierung der dezentralen Entwicklungsprozesse dar: "The group founders

78 http://httpd.apache.org/dev/release.html

set the original rules, but they can be changed by vote of the active members." [HttpdApOrg02][79]

### 3.3.5 Aktualisierung der STATUS-Files

Die STATUS-Dateien stellen ein wichtiges Werkzeug für das Projekt Management der ASF-Projekte dar. STATUS-Dateien werden für jedes Quellcodeverzeichnis definiert und gewartet und thematisieren die geplanten und durchgeführten Modifikationen am Quellcode und aktuelle Arbeiten. Dadurch werden die STATUS-Dateien für Management-orientierte Aufgaben und zur operativen Koordination der Entwicklungsprozesse instrumentalisiert. In vielen Teilprojekten werden sie außerdem regelmäßig an die Entwickler-Mailing Liste kommuniziert, um den Informationsstand der beteiligten Entwickler zu homogenisieren. Eine detailliertere Beschreibung der STATUS-Datei ist im Kapitel 3.7.1.2.1 zu finden.

### 3.3.6 Zuweisung von Privilegien

Privilegien, wie z.B. Commit-Rechte bezüglich des CVS-Archivs werden durch die verschiedenen Projektmanagement-Instanzen der Teilprojekte zentral vergeben. Somit stellen sie die oberste Entscheidungsgewalt bezüglich der Transition der Akteure zwischen den verschiedenen Rollen dar. Die Vergabe von Commit-Privilegien wurde bereits im Rahmen der Entwicklungsprozesse dargestellt.

#### 3.3.6.1 Aufnahme von Mitgliedern in die Apache Group (HTTPD)

[FoBa02] identifiziert einen Aufnahmeprozess in die Apache Group, welche als zentrale Projektmanagement-Instanz im HTTPD-Kontext betrachtet werden kann. Diesbezüglich werden potentielle Mitglieder durch die Apache Group nominiert und anschließend einstimmig durch die stimmberechtigten Mitglieder bestätigt oder abgelehnt: "New members of the Apache Group are added when a frequent contributor is nominated by one member and unanimously approved by the voting members." [HttpdApOrg02][79]. I.d.R. hat ein so gewählter Akteur bereits über einen längeren Zeitraum als Entwickler, Core Developer oder Contributor aktiv am Projekt mitgewirkt.

79 http://httpd.apache.org/ABOUT_APACHE.html

### 3.3.7 Generierung eines neuen Teilprojekts

Die Teilprojekte der ASF können stets durch neue ergänzt werden, wobei dieser Prozess über ein Antragsverfahren zentral organisiert wird. Die Entscheidungsgewalt über die Integration neuer Teilprojekte wird durch das Project Management Committee (PMC) ausgeübt, welches eine zentrale Organisation repräsentiert, die Entscheidungen von strategischer Bedeutung für die ASF trifft. Entsprechende Anträge können bei der ASF bzw. dem PMC eingereicht werden und werden anschließend vom PMC verhandelt und diskutiert. Einer Bewilligung des Antrags muss durch eine ¾ Mehrheit des PMC zugestimmt werden, um eine Integration in die ASF-Organisation zu ermöglichen, wie die Project Management Committee Bylaws definieren: "Creation of a new subproject requires approval by ¾ vote of the PMC." [JakartaApOrg02][80]

## 3.4 Infrastrukturelle Prozesse

Im Rahmen der infrastrukturellen Prozesse wird die Infrastruktur des Apache-Projekts und der verschiedenen Teilprojekte erstellt und gewartet. Damit repräsentieren die infrastrukturellen Prozesse verschiedene sekundäre Aktivitäten zur Ermöglichung der primären Systementwicklung. Aufgrund der peripheren Bedeutung dieser Prozesse für die eigentlichen Entwicklungsprozesse werden die infrastrukturellen Prozesse nur sehr rudimentär dargestellt.

Die infrastrukturellen Prozesse werden primär durch die ASF ausgeführt, die es als eine ihrer Hauptaufgaben betrachtet, eine adäquate Entwicklungsumgebung für die Entwicklergemeinde zur Verfügung zu stellen. So werden zum Beispiel einige Kollaborationswerkzeuge zentral durch die ASF für die teilprojektübergreifende Nutzung zur Verfügung gestellt. Ergänzt werden diese teilprojektübergreifenden Aktivitäten durch die einzelnen Aktivitäten der Projektmanagement-Instanzen der Teilprojekte, welche vor allem die Kommunikationskanäle individuell im Rahmen der Teilprojekte organisieren.

### 3.4.1 Kommunikationsinfrastrukturen

In diesem Abschnitt werden die infrastrukturellen Prozesse zur Etablierung der projektweiten Kommunikationsinfrastruktur dargestellt.

80 http://jakarta.apache.org/site/management.html

#### 3.4.1.1 Website Maintenance

Die Website Maintenance wird primär durch verschiedene Contributoren unter der Leitung der Projektmanagement-Instanzen wahrgenommen. Als primäre Anforderungen an die Website sind die Funktionalität als projektweites Kommunikationsmedium und die Ermöglichung einer einfachen Wartung zu betrachten: "The Web site is functional and easy to maintain, since that is what the project requires." Die Prozesse zur Website Maintenance basieren primär auf den Aktivitäten der Management-Instanzen und werden ergänzt durch die aktive Mitwirkung von Freiwilligen: "All of our teams are always looking for people to help with the documentation and Web site." [JakartaApOrg02][81]

Im Kontext des Jakarta-Projekts wird für die redaktionelle Pflege der Website ein Werkzeug, das sogenannte Jakarta-Site2 Modul genutzt, welches durch die Transformation von XML-Files in statische HTML-Dateien die XML-basierte Verwaltung der Inhalte ermöglicht und somit die redaktionellen Prozesse vereinfacht: "The Jakarta-Site2 module is where we store the code for building our static HTML website. We use XML files as our input and transform them into static HTML files [...] ." [JakartaApOrg02][81]

#### 3.4.1.2 Maintenance der Kommunikationskanäle

Neben der Website existieren verschiedene Kommunikationskanäle wie Mailing Listen, Newsgroups oder IRC, welche in Kapitel 3.8.1 genauer dargestellt werden. Diese Kommunikationskanäle werden zentral durch das Projektmanagement instrumentalisiert und etabliert und daher auch von diesen Instanzen verwaltet und gewartet.

Neben der initialen Implementierung dieser Werkzeuge sind im Rahmen der Maintenance vor allem Aufgabe wie die Einrichtung dedizierter Sub-Kanäle oder die Aufnahme von neuen Anwendern als Adressaten der Kanäle von Bedeutung.

### 3.4.2 Kollaborationsinfrastrukturen

Neben den Kommunikationskanälen sind die Werkzeuge von besonderer Bedeutung, welche die kollaborative Arbeit an gemeinsam genutzten Ressourcen ermöglichen und unterstützen. Die Kollaborationsinfrastrukturen werden im Apache-Kontext primär durch die ASF eingerichtet und verwaltet und gemeinsam durch alle Teilprojekte genutzt.

81 http://jakarta.apache.org/site/jakarta-site2.html

#### 3.4.2.1 Bugzilla Maintenance

Für die Aufgaben des Bug Tracking wird das Bugzilla-System verwendet. Dafür wurde durch die ASF zentral für alle Teilprojekte eine Instanz des Systems implementiert, welche auch durch die ASF verwaltet wird. Somit ist ausschließlich die ASF zur Etablierung neuer Klassen von Bug Reports für ein explizites Teilprojekt und zur Definition bzw. Modifikation der jeweiligen Metadaten berechtigt. Der Prozess der Userverwaltung ist weitgehend automatisiert und benötigt daher kaum zentrale Aktivitäten der ASF.

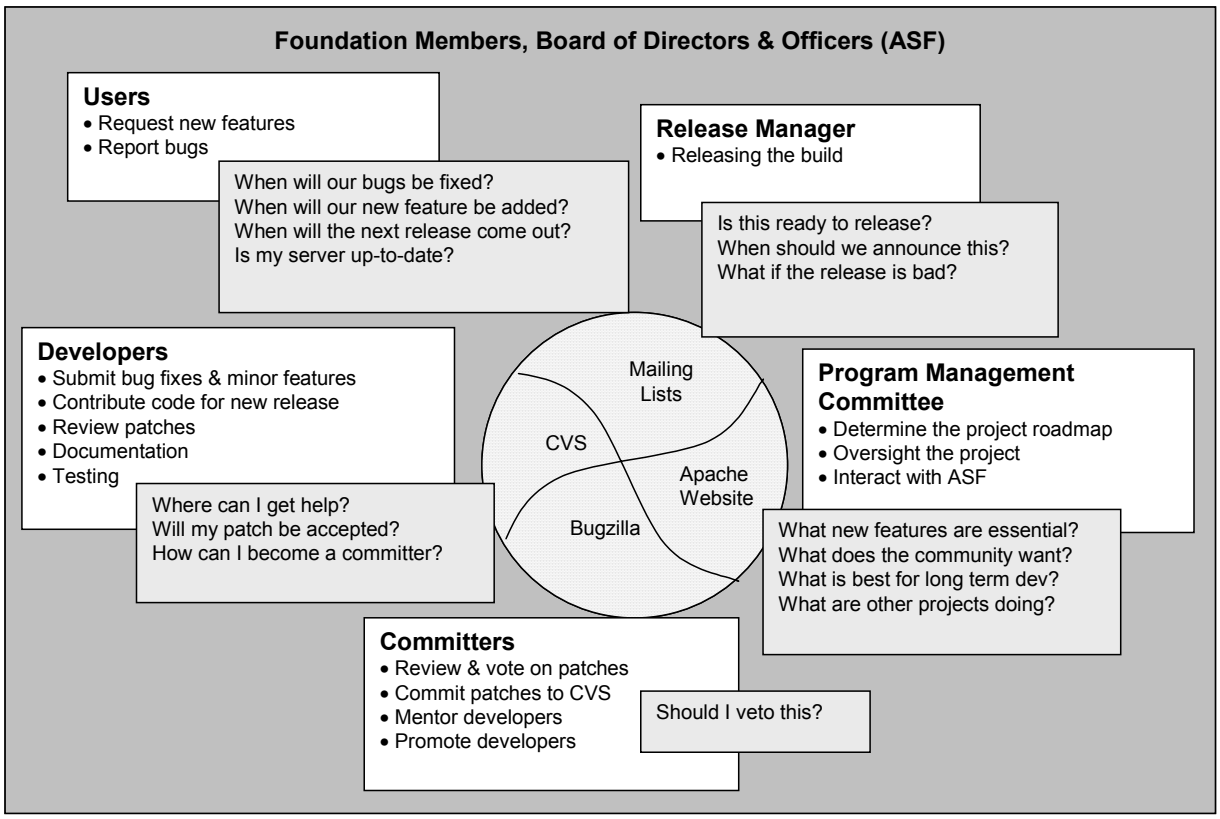
#### 3.4.2.2 CVS Maintenance

Analog zu Bugzilla wird auch für das Konfigurationsmanagement aller Teilprojekte eine zentrale Instanz eines entsprechenden Systems, respektive CVS genutzt. Diese wurde ebenfalls durch die ASF implementiert und wird auch durch die Foundation verwaltet. Somit ist die Generierung von neuen Archiven für spezifische Teilprojekte nur durch die ASF möglich. Da die operative Ausführung von Release Prozessen und die Vergabe von Commit-Privilegien durch die Akteure eines spezifischen Teilprojekts durchgeführt wird, wird aber als Prämisse angenommen, dass den Management-Instanzen der verschiedenen Teilprojekte, respektive den Project Management Commitees entsprechende Privilegien durch die ASF erteilt wurden.

## 3.5 Gesamtheitliches Prozessmodell

Die folgende Grafik aus [AtGaGe02] visualisiert eine gesamtheitliche Perspektive auf die elementaren Prozesse, Rollen und Werkzeuge des Apache HTTPD-Projekts, die in ähnlicher Form auch im Jakarta-Teilprojekt angewendet werden.

Bild 10 Apache: Gesamtheitliches Prozessmodell (nach [AtGaGe02])




## 3.6 Rollen

Dieser Abschnitt stellt die identifizierten Rollen im Rahmen der Apache-Prozesse dar. Zu Beginn werden die verschiedenen, lediglich teilprojektspezifisch agierenden Rollen beschrieben und anschließend die teilprojektübergreifend existierenden Rollen thematisiert.

Von Seiten der ASF werden in [ApacheOrg02][82] die folgenden Rollen definiert und beschrieben:

- Anwender
- Entwickler
- Committer
- Project Management Committee
- ASF Mitglieder

82 http://www.apache.org/foundation/roles.html

- Board Of Directors
- Officers

Neben diesen Rollen konnten aufgrund der eigenen Recherche noch die folgenden Rollen identifiziert werden, welche implizit durch die geeignete Aggregation von Akteuren mit ähnlichen Merkmalen und Verantwortlichkeiten entstanden sind:

- Contributor
- Core Developer
- Release Manager

Diese Rollen sind zwar nicht in eine konkrete Organisationshierarchie eingebunden, lassen sich aber aufgrund von spezifischen Rechten und Aufgabenbereichen der einzelnen Akteure identifizieren: "We do not have a hierarchical structure. Rather, different groups of contributors have different rights and responsibilities in the organization." [ApacheOrg02][83]. Außerdem erfüllt ein Akteur nicht statisch ausschließlich eine bestimmte Rolle, sondern kann entsprechend seiner Fähigkeiten und Beiträge flexibel diverse Rollen einnehmen: "The Foundation is a meritocracy where these rights and responsibilities follow from the skills and contributions of participants." [ApacheOrg02][83]

Weiterhin repräsentieren diese Rollen lediglich eine Möglichkeit der Aggregation von Aufgabenbereichen und Privilegien zu bestimmten verallgemeinerbaren Teilmengen und stellt daher nicht die einzig mögliche Rollendefinition dar. Dies schließt somit die implizite Existenz von weiteren Rollen, wie z.B. die eines Bug Reporters, Reviewers oder Software Testers nicht aus, deren Aufgaben in diesem Fall als immanenter Bestandteil der Rolle des Contributors betrachtet werden können.

### 3.6.1 User

Ein Anwender repräsentiert die minimalste Rolle in den Entwicklungsprozessen der ASF-Projekte und bildet den Einstieg für alle folgenden Rollen: " The majority of our developers start out as users and guide their development efforts from the user's perspective." [ApacheOrg02][83]. Ein Anwender zeichnet sich dadurch aus, dass er die Software lokal betreibt, aber noch keinerlei aktive Mitwirkung am Projekt von ihm ausgegangen ist.

83 http://www.apache.org/foundation/roles.html

### 3.6.2 Contributor

Ein Contributor repräsentiert die nächste Stufe auf der 'Evolution' des Akteurs und zeichnet sich durch aktive Beiträge zum Entwicklungsprozess aus. Diese können vom Akteur generierte Dokumentationsartefakte, Bug Reports, Enhancement Requests oder andere Artefakte beinhalten: "Users contribute to the Apache projects by providing feedback to developers in the the form of bug reports and feature suggestions. As well, users participate in the Apache community by helping other users on mailing lists and user support forums." [ApacheOrg02][84]

### 3.6.3 Developer

Ein Entwickler zeichnet sich dadurch aus, dass er aktive Modifikationen am Quellcode vornimmt und diese in Form von Patches dem Projekt zur Verfügung stellt: "Developers usually subscribe to the project development mailing list and contribute by sending patches to the list." [ApacheOrg02][84]. Ein Developer kann Quellcode über das CVS-Archiv beziehen, verfügt aber noch nicht über die Rechte für eine Integration der Patches: " Developers do not have capabilities to commit changes into the CVS." [AtGaGe02]

#### 3.6.3.1 Core Developer

Wie bereits in der Beschreibung der Organisation des HTTPD Pojektes dargestellt wurde, konnte im Rahmen des HTTPD-Projektes eine Gruppe von Entwicklern identifiziert werden, welche für eine überwiegende Mehrheit der Quellcodemodifikationen verantwortlich zeichnet. Diese repräsentieren die Gruppe der Core Developper.

### 3.6.4 Committers

Ein Entwickler, der seine Qualifikation und Kompetenz durch langfristige Mitarbeit und wertvolle Beiträge zum Projekt unter Beweis gestellt hat, kann durch die Management Instanzen das Privileg des schreibenden Zugriffs auf das CVS-Archiv erhalten: "Frequent and valuable contributers to a project are granted write access to the source code repository." [ApacheOrg02][84]. Dadurch nimmt ein Entwickler automatisch die Position eines Committers ein und trägt somit die Verantwortung über die Integration von Patches in den Quellcode des Projekts. Dabei können sowohl herkömmliche Patches als auch

84 http://www.apache.org/foundation/roles.html

Dokumentationsartefakte in das CVS-Archiv integriert werden. Über die Integration von Patches und auch die Aufnahme weiterer Committer wird durch die bestehenden Committer abgestimmt, wobei ein neuer Committer nur durch eine einstimmige Akzeptanz aufgenommen werden kann: "After a unanimous voting, a developer will advance to a committer status." [AtGaGe02]

Committer bekommen nach [ApacheOrg02][85] außerdem durch die ASF einen E-Mail-Account der Apache Organisation zugewiesen (@apache.org) und können nach [AtGaGe02] Mitglied des Project Management Committees werden.

#### 3.6.4.1 HTTPD-Committers

Die Committers des HTTPD-Server Projekts werden ebenfalls durch andere Committer nominiert und anschließend durch eine einstimmige Akzeptanz der aktiven Mitglieder des Project Management Committees ernannt, wie die Projekt Management Richtlinien des Projektes belegen: "Membership as a Committer is by invitation only and must be approved by consensus of the active Apache PMC members." [HttpdApOrg02][86]. Die Mitgliedschaft in der Gruppe der Committers kann dem Committer nur durch eine einstimmige Abstimmung des Project Management Committees entzogen werden: "Membership can be revoked by a unanimous vote of all the active PMC members... " [HttpdApOrg02][86]

### 3.6.5 Release Manager

Wie bereits im Prozess des Software Release dargestellt wurde, existiert in den einzelnen Teilprojekten die nicht formal spezifizierte Rolle des Release Managers: "When the project nears a product release, one of the core developers volunteers to be the release manager [...] " [MoFiHe00]

Die primären Aufgabenbereiche sind die Identifikation von kritischen Problemen, welche der Generierung des Release gegenüberstehen, die Transition der Software durch die einzelnen Testphasen des Veröffentlichungsprozesses und die Koordination der Freeze-Prozesse. Der Release Manager repräsentiert die uneingeschränkte Autorität im Rahmen der Veröffentlichung eines neuen Softwarerelease: "This individual has the ultimate authority over what makes into the release." [AtGaGe02]

85 http://www.apache.org/foundation/roles.html
86 http://httpd.apache.org/dev/guidelines.html

### 3.6.6 Project Management Committee

Committer, welche für die strategische Ausrichtung und Koordination der Teilprojekte verantwortlich sind, bilden das Project Management Committee (PMC) eines Teilprojekts. Für jedes Teilprojekt der obersten Projekthierarchie im Apache-Kontext (z.B. Jakarta und HTTPD) existiert ein PMC, wobei die jeweiligen PMC u.u. für mehrere untergeordnete Teilprojekte verantwortlich sind. Das PMC ist ein selbstgewähltes Gremium, welches direkt durch das Board of Directors der ASF delegiert wird: "Members of the PMC are self-selected committers. [...] Although the Board of Directors ultimately has the final decision making power on any project, they delegate this responsibility to the PMC of each project." [AtGaGe02]. Im Rahmen des Release Prozesses verfügt zwar der Release Manager über die oberste Entscheidungsgewalt, das PMC kann nach [AtGaGe02] aber dennoch Vorschläge bezüglich der Integration von Patches in das Release machen kann.

#### 3.6.6.1 HTTPD - Project Management Committee

Das Management der projektweiten Ressourcen und die Entscheidung über den Inhalt einer spezifischen Softwaredistribution wird durch das PMC des HTTPD-Projekts ausgeübt. Ein PMC-Mitglied kann nur durch existierende PMC-Mitglieder nominiert werden und muss durch einen vollständigen Konsens der aktiven Mitglieder bestätigt werden, wie in den Projekt-Guidelines des HTTPD-Projekts definiert wird: "Membership in the Apache PMC is by invitation only and must be approved by consensus of the active Apache PMC members." [HttpdApOrg02][87]

### 3.6.7 Teilprojektübergreifende Rollen

Die folgenden Rollen üben teilprojektübergreifende Aufgabenbereiche über die gesamte Apache Organisation hinweg aus.

#### 3.6.7.1 ASF Members

Die Aufnahme als Mitglied in die ASF kann nur durch eine Nominierung und Abstimmung durch die ASF erfolgen und setzt außergewöhnlich aktive und konstante Mitwirkung an einem oder mehreren Teilprojekten der ASF über einen langen Zeitraum voraus: "Individuals who have made sustained and important contributions to one or more of the foundation's projects can be

87 http://httpd.apache.org/dev/guidelines.html

invited to become Members of the Apache Software Foundation." [ApacheOrg02][88]. Die Mitglieder sind nicht an die Mitwirkung an einem bestimmten Projekt gebunden und rekrutieren sich daher aus verschiedenen Projekten der Apache Organisation. Die Mitglieder der ASF sind in erster Linie für die Wahl des Board of Directors verantwortlich. Die Kommunikation der ASF-Mitglieder basiert auf einer dedizierten Mailing Liste und einem jährlich stattfindenden Meeting.

#### 3.6.7.2 Board of Directors

Das Board of Directors der ASF ist verantwortlich für die Verwaltung, das Management und die geschäftlichen bzw. Belange der Apache Organisation und ist daher klar von den Projektmanagement-Instanzen der einzelnen Teilprojekte abzugrenzen: "The Apache Software Foundation Board of Directors is responsible for management and oversight of the business and affairs of the corporation." [ApacheOrg02][88]

Das Board of Directors ist somit ausschließlich für die gesamtheitlichen Belange der ASF verantwortlich und delegiert die Systementwicklungs- und teilprojektspezifischen Management-Aufgaben an die verschiedenen Project Management Committees der einzelnen Teilprojekte: " While the Board retains ultimate responsibility for the foundation, it delegates decision-making authority for the technical direction of projects to the Project Management Committees." [ApacheOrg02][88]

#### 3.6.7.3 Officers

Die sogenannten Officers repräsentieren die operativen Instanzen des Board of Directors und sind somit verantwortlich für die Kontrolle und Steuerung der operativen Geschäfte und Aufgaben der ASF: "The Officers of the Apache Software Foundation oversee the day-to-day affairs of the Foundation." [ApacheOrg02][88]. Sie werden durch das Board of Directors gewählt und ernannt.

## 3.7 Artefakte

Im folgenden werden die elementarsten Artefakte der identifizierten Prozesse beschrieben und exemplarisch verschiedene Artefakte der einzelnen Artefakt-Kategorien vorgestellt.

88 http://www.apache.org/foundation/roles.html

### 3.7.1 Managementartefakte

Die Managementartefakte dienen der Koordination und dem Management der Entwicklungsprozesse und werden basierend auf dem definierten Metamodell in die Kategorien strategische und technologische Managementartefakte eingeteilt.

#### 3.7.1.1 Strategische Managementartefakte

Die strategischen Managementartefakte sind von zentraler Bedeutung für die strategische Ausrichtung des Projekts und determinieren den Entwicklungsprozess in entscheidendem Maß.

##### 3.7.1.1.1 Lizenz - Apache Software License

Wie in den meisten OS-Projekten schafft die Lizenz, die für die Verbreitung des Quellcodes genutzt wird, die elementaren Voraussetzungen und strategischen Rahmenbedingungen für die Weiterentwicklung der Apache-Produkte. Alle Software-Releases der ASF werden unter der sogenannten Apache Software License (ASL) vertrieben, welche sich in der aktuellen Version sehr eng an den Vorgaben der GNU GPL ausrichtet und damit die Bedingungen für eine echte OS-Lizenz erfüllt.

Die vollständige, aktuell verwendete Lizenz ist unter [ApacheOrg02][89] verfügbar.

##### 3.7.1.1.2 ASF-Bylaws: Organisationsgrundsätze

Die sogenannten Foundation Bylaws[90] determinieren die Aktivitäten der ASF, definieren die Organisationsstruktur, die Privilegien und Pflichten der einzelnen Mitglieder und reglementieren die verschiedenen Teilprozesse, z.B. zur Ernennung von sogenannten 'Officers' oder zur Durchführung von Meetings der Mitglieder. Weiterhin werden in diesem Grundsatzdokument verschiedene strategische Vorgaben für die ASF definiert, woraus sich auch Implikationen für das Management der einzelnen Teilprojekte ableiten lassen. So wird z.B. in den Bylaws der Non-Profit-Status der Apache Organisation definiert: "The corporation is organized and shall be operated as a not-for-profit membership corporation organized under Delaware law." [ApacheOrg02][91]

89 http://www.apache.org/LICENSE
90 http://www.apache.org/foundation/bylaws.html

#### 3.7.1.1.3 Bug Reporting Guideline

Da für die Erstellung und Bearbeitung von Fehlerberichten eine projektweite Infrastruktur in Form einer Instanz des Bug Tracking Systems ‚Bugzilla' genutzt wird, wurden diesbezügliche Richtlinien auch teilprojektübergreifend über die gesamte Apache-Organisation hinweg formuliert. Diese Artefakte[92] beschreiben und reglementieren die Verwendung des Bugzilla-Systems[93]. Indem diese Artefakte zur Reglementierung aller Prozesse dienen, die auf das zentrale Bugzilla-System zurückgreifen, stellen sie wichtige Managementartefakte dar, da sie zur Steuerung und Koordinierung der primären Entwicklungsprozesse beitragen.

#### 3.7.1.1.4 Teilprojektweite Guidelines

Neben den teilprojektübergreifenden Grundsätzen der ASF existieren in den einzelnen Teilprojekten Richtlinien, welche nur im Kontext des individuellen Teilprojekts von Bedeutung sind.

**Project Management Committee Bylaws**

In den einzelnen Teilprojekten existieren z.T. explizite Richtlinien der Project Management Committees. Diese thematisieren auf Teilprojektebene ähnliche Inhalte analog zu den ASF-Bylaws. Ein Beispiel sind z.B. die Bylaws des PMC des Jakarta Projekts[94].

**Mailing List Guideline**

In den Teilprojekten wurden Guidelines für die Verwendung der projektinternen Kommunikationswerkzeuge, wie z.B der Mailing Listen, definiert. Im Rahmen des HTTPD-Projekts und im Kontext der Jakarta-Teilprojekte sind derartige Dokumente im Web[95] verfügbar.

91 http://www.apache.org/foundation/bylaws.html
92 http://issues.apache.org/bugzilla/bugwritinghelp.html, http://issues.apache.org/bugzilla/docs/html/Bugzilla-Guide.html
93 http://issues.apache.org/bugzilla
94 http://jakarta.apache.org/site/management.html
95 http://httpd.apache.org/userslist.html, http://jakarta.apache.org/site/mail.html

### Project Management Guidelines

In allen evaluierten Teilprojekten existieren Projekt-Guidelines, welche den Entwicklungsprozess beschreiben und durch dieses Modell die Vorgaben für jegliche weitere Entwicklung definieren. Somit können sie als präskriptives Modell der teilprojektspezifischen Entwicklungsprozesse betrachtet werden und determinieren die praktizierten Entwicklungsprozesse in entscheidendem Umfang. Wichtige Bestandteile sind die Definition von Rollen und Verantwortlichkeiten, die Beschreibung der Kommunikationskanäle und deren Verwendung, der Quellcodeverwaltung und -organisation, die Definition der grundlegenden Entscheidungsprozesse zur Lösung von Konflikten und eine Darstellung der Projektmanagement-Organe und -Instanzen.

Diese Guidelines dienen als fundamentale Dokumentation der optimalen Entwicklungsprozesse und sind somit ein wichtiges Artefakt im Prozess der Integration von neuen Entwicklern in die Community. Für das HTTPD-Projekt und im Jakarta-Kontext sind diese Dokumente im Web[96] verfügbar.

### Release Guidelines

Weiterhin exitistieren Guidelines, welche den Veröffentlichungsprozess neuer Softwarereleases beschreiben und reglementieren und als Vorgabe für den Release Manager des Teilprojekts dienen. Dieser hat aber in begrenztem Umfang die Möglichkeit, diese Vorgaben individuell anzupassen und deren Realisierung abweichend umzusetzen, wie dies der Release Guide des HTTPD-Projekts belegt: "As described herein, this policy is not set in stone and may be adjusted by the Release Manager." [HttpdApOrg02][97]

### Programmierkonventionen

Weiterhin wird in den Teilprojekten auf verschiedene Programmierrichtlinien verwiesen die z.T. explizit für das Projekt definiert wurden. Im Rahmen des Jakarta-Projekts wird z.B. auf die 'Code Conventions for the Java Programming Language'[98] verwiesen, welche durch die Java-Initiatoren Sun Microsystems definert wurden: "All Java Language source code in the repository must be written in conformance to the "Code Conventions for the Java Programming Language" as published by Sun, or in conformance with another well-defined convention specified by the subproject." [JakartaApOrg02][99]

96 http://httpd.apache.org/dev/guidelines.html, http://jakarta.apache.org/site/guidelines.html
97 http://httpd.apache.org/dev/release.html
98 http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
99 http://jakarta.apache.org/site/source.html

Parallel dazu verweisen die FAQ des Jakarta-Projekts[100] auf verschiedene teilprojektspezifische Programmierrichtlinien. Im Kontext des HTTPD-Projekts existiert ein Style-Guide für die Erstellung von C-Quellcode, der 'C-Language Styleguide'[101], welcher verschiedene Vorgaben zur Verbesserung der Lesbarkeit, der Modularisierung und anderen Anforderungen der Softwarequalität definiert.

#### 3.7.1.2 Operationale Managementartefakte

Neben den beschriebenen strategischen Managementartefakte existieren die folgenden Managementartefakte, die basierend auf dem Metamodell eher operationalen Charakter besitzen.

##### 3.7.1.2.1 STATUS-File

In jedem Quellcode-Archiv der Teilprojekte existiert ein sogenanntes STATUS-File, welches Inhalte thematisiert, die in direktem Zusammenhang mit dem Quellcode stehen: "Each of the Project's active source code repositories contain a file named STATUS which is used to keep track of the agenda and plans for work within that repository." [JakartaApOrg02][102]

Diese Datei beinhaltet Informationen über geplante Release-Prozesse, aktuelle Änderungen des Quellcodes, geplante Modifikationen und kurze Beschreibungen aktueller Entwicklungsaktivitäten: "The status file includes information about release plans, a summary of code changes committed since the last release, a list of proposed changes that are under discussion, brief notes about items that individual developers are working on or want discussion about, and anything else that may be useful to help the group track progress." [JakartaApOrg02][102]. Es wird von allen Entwicklern als Informationsquelle genutzt und durch die Committer von Patches und die Mitglieder des PMC aktualisiert. Damit stellt es ein wichtiges Werkzeug zur operativen Kontrolle und Steuerung der Entwicklunsprozesse dar.

Außerdem empfiehlt [JakartaApOrg02][102] die Versendung der aktuellen STATUS-Files über die Entwickler Mailing Listen in einem mehrmals wöchentlichem Rhythmus, wobei dieser Prozess im HTTPD-Kontext automatisiert wurde. [AtGaGe02] identifiziert außerdem, dass das STATUS-File vom PMC instrumentalisiert wird, um prinzipielle Anforderungen zu

100 http://jakarta.apache.org/site/faqs.html
101 http://httpd.apache.org/dev/styleguide.html
102 http://jakarta.apache.org/site/source.html

dokumentieren, die von Anwendern vorgeschlagen und durch das PMC akzeptiert wurden und die somit in Form von geplanten Entwicklungsaktivitäten in das STATUS-File aufgenommen werden.

#### 3.7.1.2.2 Project Plan

Weiterhin konnte ein sogenannter Project Plan identifiziert werden, der im HTTPD-Kontext verwendet wird, um Zielsetzungen, Anforderungen und Arbeitsschritte für die Erstellung eines weiteren Release zu dokumentieren und somit als operationales Projekt Management Artefakt betrachtet werden kann.

Dieser Projekt Plan thematisiert die operationelle Zielsetzung für das nächste Softwarerelease und stellt eine absolut unverbindliche und ständig aktualisierte Zusammenfassung der prinzipiellen Anforderungen dar: "This document aims to encapsulate our plans for world domination, [...] I mean, further work on the Apache web server. This is simply a statement of goals, is always subject to change, and should not be relied upon for business or mission-critical purposes." [HttpdApOrg02][103]

#### 3.7.1.2.3 Release Plan

[AtGaGe02] identifiziert einen impliziten Release Plan, der durch die Inhalte des Projekt Plans und des STATUS-Files repräsentiert wird und nach [AtGaGe02] die folgenden Aspekte definiert:

- Zeitplan für neue Releases
- Meilenstein zur Realisierung eines neuen Release
- Namenskonventionen für das Release
- Funktionale Anforderungen an das Release
- Technologische Anforderungen (z.B. Kompatibilität zu bestimmten Hardwareplattformen und Betriebssystemen)

Dieses Artefakt wurde aber nicht explizit als eigenständiges Informationsobjekt, sondern implizit als Bestandteil der beiden genannten Artefakte identifiziert.

### 3.7.2 Technologische Artefakte

In diesem Abschnitt werden die technologischen Artefakte dargestellt, welche primär einer bestimmten Phase des Software-Lifecycle zugeordnet werden können.

103 http://httpd.apache.org/dev/project-plan.html

#### 3.7.2.1 Anforderungsartefakte

Die Anforderungsartefakte repräsentieren technologische Artefakte, welche primär der Phase der Anforderungsanalyse zugeordnet werden können oder in diesem Kontext generiert werden.

##### 3.7.2.1.1 Bug Report

Das bedeutendste Anforderungsartefakt repräsentiert der Bug Report, welcher im Bugzilla System erstellt und bearbeitet wird und die Anforderungen für die Entwicklung von Bug Fixes in Form von Patches definiert. Aufgrund der Tatsache, dass jeglicher Patchentwicklungsprozess auf einem spezifischen Change Request basiert, kann dieses Artefakt als fundamentaler Bestandteil des gesamten Entwicklungsprozess betrachtet werden. Für das Bug Tracking wurde eine Instanz der Bugzilla Software zur teilprojektübergreifenden Nutzung implementiert[104].

Existierende Bug Reports können über ein webbasiertes Interface[105] eingesehen und anhand ihrer Attribute bzw. Metadaten durchsucht werden.

**Format eines Bug Reports**

Die Struktur eines Bug Reports wird in großem Maße durch die Vorgaben des Bugzilla-Systems determiniert und orientiert sich daher eng an den Attributen der Bug Reports aus dem Mozilla-Kontext, in welchem Bugzilla ursprünglich entwickelt wurde. Daher wird auch in diesem Kontext auf die folgende Mozilla-Projektstudie verwiesen.

Die wichtigsten Attribute eines Bug Reports:

- Status
- Resolution
- Platform
- OpSys
- Priority
- Severity

Die Attribute Status und Resolution repräsentieren und steuern den Lebenszyklus des Bug Reports und stellen daher einen elementaren Bestandteil

104 http://nagoya.apache.org/bugzilla/
105 http://issues.apache.org/bugzilla/query.cgi

eines Bug Reports dar. [IssuesApOrg02][106] beschreibt die möglichen Ausprägungen des Status-Attributs im Apache-Kontext wie folgt:

- *Unconfirmed*: Bug Report wurde noch nicht durch einen entsprechend privilegierten Anwender validiert und als relevanter Fehlerbericht verifiziert.
- *New*: Der beschriebene Fehler wurde verifiziert und soll durch einen evtl. zugeordneten ‚Assignee' bearbeitet werden, der dies aber noch nicht bestätigt hat.
- *Assigned*: Der ursprünglich Assignee oder ein anderer Entwickler haben die Bearbeitung durch die Vergabe von diesem Status bestätigt.
- *Resolved*: Ein erstes Patch zur Behebung des Fehlers wurde durch den Entwickler entwickelt und kommuniziert.
- *Verified*: Die Reviewer des Projekts haben das Patch evaluiert und bestätigt.
- *Closed*: Das Patch wurde in den Quellcode und in ein offizielles Release integriert.
- *Reopened*: Das Patch wurde aufgrund von Schwächen als nicht umfassend problemlösend eingestuft und der Bug Report muss erneut bearbeitet werden.

Das Resolution-Attribut dient dazu, eine evtl. zur Verfügung gestellte Lösung zu beschreiben und enthält somit keinen Wert, wenn sich der Bug Report noch in einem der geöffneten Stati befindet. Andernfalls kann dieses Attribut die folgenden Ausprägungen annehmen:

- *Fixed*: Ein Patch wurde bereits evaluiert und in den Quellcode integriert.
- *Invalid*: Das beschriebene Problem ist kein verifizierbarer Softwarefehler.
- *Wontfix*: Das beschriebene Problem kann nicht in Form eines Patches behoben werden.
- *Later*: Der Bug Report kann nur in einer späteren Version bearbeitet werden.
- *Remind*: Der Bug Report wird wahrscheinlich nicht in der aktuellen Version der Software bearbeitet.
- *Duplicate*: Der Bug Report existierte bereits im Bugzilla System.
- *Worksforme*: Der Fehler konnte nicht reproduziert werden.

Das Plattform-Attribut dient der Definition der Hardware-Plattform, auf welcher der Softwarefehler eruiert wurde und das Feld OpSys dient der Spezifikation des verwendeten Betriebssystems.

Das Attribut Priority dient der Zuweisung einer Priorität und kann die Ausprägungen P1 für höchste Priorität bis P5 für die niedrigste Priorität

106 http://issues.apache.org/bugzilla/queryhelp.cgi

einnehmen: "This field is utilized by the programmers/engineers to prioritize their work." [IssuesApOrg02][107]

Das Attribut Severity dient der Definition der Auswirkungen eines Softwarefehlers und kann die folgenden Werte mit den angeführten, relativ unpräzise konnotierten Bedeutungen einnehmen:

- *Blocker*: behindert weitere Entwicklungsarbeiten
- *Critical*: bewirkt Systemabstürze oder Datenverlust
- *Major*: grundlegende Funktionalität wird verhindert
- *Normal*: durchschnittlicher Softwarefehler
- *Minor*: das Problem kann mit minimalem Aufwand umgangen werden
- *Trivial*: trivialer Fehler, wie z.B. Rechtschreibfehler
- *Enhancement*: Der Bug Report beschreibt keinen Fehler, sondern die Anforderung einer neuen Funktionalität oder Systemerweiterung.

#### 3.7.2.1.2 Enhancement Requests

Wie bereits dargestellt wurde, werden Enhancement Requests ebenfalls in Form von Bug Reports kommuniziert und besitzen daher dieselben Eigenschaften und Attribute, mit der Einschränkung, dass das Attribut ‘Severity' stets den Wert ‘Enhancement' besitzt.

#### 3.7.2.1.3 Proposed Features

[AtGaGe02] identifiziert die sogenannten 'Proposed Features' welche durch das PMC eines Teilprojekts definiert werden und verschiedene strategisch wichtige Anforderungen und Funktionalitäten definieren: "The PMC develops a list of proposed features based on their personal judgment, requested feature enhancements in Bugzilla, and from the enhancement coded as patches (which is submitted to the developers accessible portion of the website)." [AtGaGe02]. Diese Anforderungen werden in Form einer Project Roadmap zusammengefasst und an die Entwicklergemeinde kommuniziert [CvsApOrg02][108]. Nachdem die so spezifizierten Anforderungen durch die Community evaluiert wurden, werden sie in Form des bereits beschriebenen STATUS-Files operationalisiert.

107 http://issues.apache.org/bugzilla/queryhelp.cgi
108 http://cvs.apache.org/viewcvs.cgi/httpd-2.0/ROADMAP

#### 3.7.2.2 Entwurfsartefakte

Im Kontext der Apacheprojekte konnten keine technologischen Artefakte identifiziert werden, die explizit dem Softwareentwurf dienen und somit dieser Kategorie zugeordnet werden können. Wie in den meisten OSS-Projekten sind die Informationen bezüglich des Softwaredesigns i.d.R. implizit in den Anforderungs- und Implementationsartefakten definiert. So determinieren die Bug Reports den Entwurf eines Patches entscheidend, welches wiederum als primäres Entwurfsartefakt betrachtet werden kann, da es den Softwareentwurf implementiert und somit inhärent definiert.

#### 3.7.2.3 Implementationsartefakte

Die Implementationsartefakte werden i.d.R. in der Phase der Implementation generiert und können somit dieser Kategorie zugeordnet werden.

##### 3.7.2.3.1 Patch

Das bedeutendste Artefakt in der Phase der Implementation wird durch das Patch repräsentiert. Ein Patch entsteht im bereits dargestellten Prozess der Patch-Entwicklung und stellt die Quellcode-Differenz zwischen dem aktuellen Quellcode-Archiv und dem lokal modifizierten Entwicklungsquellcode, des jeweiligen Patch-Entwicklers dar. Ein Patch basiert stets auf einem Bug Report oder einem Enhancement Request und wird basierend auf den dort formulierten Anforderungen generiert. Es setzt sich aus einer oder mehreren Quellcode-Dateien in Form von ASCII-Text zusammen, die mit dem Werkzeug diff lokal durch den Entwickler generiert wurden.

Patches stellen somit den Output der Entwickler dar, welche diese über die Kommunikationskanäle oder das Bugzilla-System an die Reviewer kommunizieren oder selbständig in den Quellcode integrieren, falls sie zugleich auch die Rolle eines Committers ausüben: "Patches are output from developers and input into the communication channels (e.g. mailing list or Bugzilla)." [AtGaGe02]

**Released Patches**

[AtGaGe02] identifiziert außerdem eine Unterkategorie der Patches, die bereits minimal evaluiert und durch einen Committer in den Entwicklungsquellcode integriert wurden. Über eine offizielle Distributionswebsite für Patches werden diese veröffentlicht und stehen somit bereits vor der Integration in ein offizielles Softwarerelease und dessen Veröffentlichung der Allgemeinheit zur Verfügung. Dies grenzt die Released Patches von anderen Patches ab, die lediglich einer

begrenzten Personengruppe über die Entwickler-Mailing Listen oder den aktuellen Entwicklungsquellcode zur Verfügung stehen.

#### 3.7.2.3.2 Softwarereleases

Die verschiedenen Klassen der Softwarereleases können ebenfalls als Implementationsartefakt betrachtet werden, da deren Quellcodes die Basis für die weitere Implementationstätigkeit darstellen. Wie bereits im Abschnitt ‚Bezug des Quellcodes' verdeutlicht wurde existieren je nach Produktreife, Zielgruppe und Verwendungszweck der Softwarereleases verschieden Kategorien, denen eine Softwareversion zugeordnet werden kann.

Im Jakarta-Projekt wird hierfür die folgende Terminologie verwendet:

- Nightly Snapshots
- Milestone Source Drops
- Release Source Drops

Analog dazu werden im HTTPD-Kontext die folgenden Begriffe eingesetzt:

- Alpha Release
- Beta Release
- General Availability Release

Diese verschiedenen Klassen von Artefakten wurden bereits in den entsprechenden Prozessbeschreibungen umfassend dargestellt und werden daher in diesem Kapitel nicht weiter thematisiert.

### 3.7.2.4 Einsatzartefakte

Die Kategorie der Einsatzartefakte dient primär der Verwendung der Software bzw. wird in diesem Kontext generiert.

#### 3.7.2.4.1 Builds - Kompilierte Binärversionen

Wie bereits im Abschnitt 'Download von Builds' (Kapitel 3.2.1.3) ausführlich dargestellt wurde, existieren im Rahmen der Apache-Teilprojekte verschiedene Kategorien von Builds, welche basierend auf den bereits dargestellten Klassen des Quellcodes und der Source Releases erstellt wurden. Da die kompilierten Binärversionen im Kontext der Entwicklung nicht von Bedeutung und eher im Kontext der Anwendung der Software eingesetzt werden, können sie entsprechend dem Metamodell als Einsatzartefakte klassifiziert werden.

Im Kontext des HTTPD-Projekts werden die folgenden Builds erstellt:

- Alpha Build
- Beta Build
- General Availability Build

Im Rahmen des Jakarta Projekts werden analog dazu die folgenden Builds generiert:

- Demo Builds
- Nightly Builds
- Milestone Builds
- Release Builds

#### 3.7.2.4.2 Dokumentationsartefakte

Neben den verschiedenen impliziten Dokumentationsartefakten, welche als inhärente Informationen der verschiedenen themenspezifischen Usenet Newsgroups und Mailing Listen existieren, werden auch diverse explizite Dokumentationsartefakte generiert. Diese Dokumente werden z.T. auch über das CVS-Archiv verwaltet. Webbasierte Dokumentation existiert in Form von expliziten Manuals, Frequently Asked Questions, HowTo-Guides u. ä. Informationsquellen.

Diese Dokumente aggregieren i.d.R. Informationen zu einem bestimmten Themengebiet für eine bestimmte Zielgruppe. So existieren z.B. allgemeine FAQ über die Apache Organisation[109] und FAQ bezüglich der verschiedenen Teilprojekte des Jakarta-Teilprojekts[110]. Ein Merkmal, welches alle Dokumentationsartefakte aufweisen, ist die Verwendung und Formatierung in Form von etablierten und standardisierten Formaten, wie z.B. HTML oder ASCII-Text. Diese Eigenschaft ist eine direkte Konsequenz aus den verteilten und heterogenen Entwicklungsprozessen.

## 3.8 Werkzeuge

Wie auch in anderen OSS-Projekten determinieren die Charakteristika der instrumentalisierten Infrastruktur den Entwicklungsprozess in entscheidendem Maße: "Although there are not many tools, these tools are powerful and effective in handling the large number of members in the community and the distribution of those members." [AtGaGe02]. Diese Beeinflussung kann aber als bilaterale Wechselwirkung betrachtet werden, da auch die

109 http://www.apache.org/foundation/faq.html
110 http://jakarta.apache.org/site/faqs.html

Entwicklungsprozesse direkt die Infrastruktur des Projekts prägen. Aufgrund des zentralen und teilprojektübergreifenden Managements des Hauptbestandteils der Infrastruktur, muss bezüglich einiger Werkzeuge, respektive den Kollaborationsinfrastrukturen, keine Unterscheidung zwischen den zentralen Werkzeugen der einzelnen Teilprojekte vorgenommen werden: "The tools and network infrastructure in the Jakarta project are the same as those in the HTTPD project." [AtGaGe02]

### 3.8.1 Kommunikationswerkzeuge

Die wichtigsten Werkzeuge zur projektweiten Kommunikation der Entwickler und des Projektmanagements mit den Entwicklern stellen die Mailing Listen und die Web Portale der Teilprojekte dar: "The entire community coordinates with each other mainly by communicating through mailing list [...].They also rely on information posted on the project web portal." [AtGaGe02]. Daneben existieren verschiedene weitere Werkzeuge. Auch die direkte persönliche Kommunikation findet in sehr eingeschränktem Rahmen, z.B. bei Meetings der ASF oder auf Entwickler-Treffen statt: "For Apache in particular, there is an annual conference for all the developers to gather and to discuss the project." [AtGaGe02]

#### 3.8.1.1 Projekt Website

Neben der zentralen Website der Apache Organisation [ApacheOrg02] existieren für jedes Teilprojekt explizite Web Portale.

- HTTPD: http://www.httpd.apache.org [HttpdApOrg02]
- Jakarta: http://www.jakarta.apache.org [JakartaApOrg02]

Für jedes Teilprojekt des Apache Jakarta Projekts existieren weitere teilprojektspezifische Websites: "The Jakarta Web site is a container for the various products hosted by the project. The development team for each product maintains their area of the Web site, along with rest of the product's documentation." [JakartaApOrg02][111]. Jede Website dient dabei verschiedenen Verwendungszwecken. So wird die zentrale Website der ASF primär zur Kommunikation von allgemeinen Grundsätzen, teilprojektübergreifenden Informationen und zur Distribution der stabilen und offiziell publizierten Anwenderversionen der einzelnen Softwareprodukte genutzt und dient als Einstiegsportal für die weitere Recherche in den verschiedenen Teilprojekten, während die Entwicklungsprozesse der diversen

111 http://jakarta.apache.org/site/jakarta-site2.html

Teilprojekte ausschließlich über deren Web Portale koordiniert und organisiert werden.

#### 3.8.1.2 Mailing Lists

"Mailing lists are crucial to the operation of the Apache Software Foundation. The Foundation, as well as each of the ASF Projects, uses these lists to coordinate development of the software and administration of the organization." [ApacheOrg02][112]

Neben dem primären Zweck der Koordination der Entwicklergemeinde und der Ermöglichung der Kommunikation zwischen den einzelnen Entwicklern, fungieren verschiedene dedizierte Mailing Listen als Medium zur Durchführung von Anwendersupport: "Mailing lists also serve as a primary support channel where users can help each other learn to use the software." [ApacheOrg02][112]. Die Listen sind für jeden zugänglich und werden durch die Projektmanagement-Instanzen der ASF bzw. der Teilprojekte, respektive den verschiedenen Project Management Committees erstellt und verwaltet.

Zur Recherche in bereits vergangenen Beiträgen, welche über die Mailing Listen kommuniziert wurden, existieren außerdem verschiedene Mailing Listen Archive: "Archives for public mailing lists are available at a number of locations, [...]" [ApacheOrg02][112]

Nach [AtGaGe02] können die Mailing Listen in vier verschiedene Kategorien unterteilt werden, die bereits im Kapitel 3.1.7.3.1 eingehender dargestellt wurden:

- Announcement Lists
- User Lists
- Developer Lists
- Commit Lists

Außerdem existieren verschiedene Mailing Listen, welche der Kommunikation von internen Problemen bezüglich des Projekt Managements oder strategischer Entscheidungen dienen und nicht für die Öffentlichkeit bzw. die gesamte Community zugänglich sind. So beschreiben die Projektmanagement-Guidelines des HTTPD-Projekts z.B. eine sogenannte 'Private List': "The Apache Group's private mailing list for discussion of issues that are inappropriate for public discussion, such as legal, personal, or security issues prior to a published fix. Subscription to the list is only open to Apache Group members." [HttpdApOrg02][113]. Auf Ebene der ASF bestehen ebenfalls

112 http://www.apache.org/foundation/mailinglists.html

zugangsbeschränkte Mailing Listen für die Kommunikation interner Aspekte, wie z.B. dedizierte Listen, die nur für Mitglieder der ASF oder des Board of Directors zugänglich sind.

#### 3.8.1.3 Usenet Newsgroups

Analog zu den Mailing Listen werden im Apache-Projekt verschiedene Usenet Newsgroups instrumentalisiert, wobei die Mailing Listen aber die primären Werkzeuge für die projektinterne Kommunikation darstellen und über eine weitaus größere Reichweite verfügen.

#### 3.8.1.4 Internet Relay Chat

Auch die Kanäle des Internet Relay Chat werden durch die Community des Apache-Projekts zur Kommunikation instrumentalisiert: "Announcements are not only performed via email. They are also communicated via IRC chats […]" [AtGaGe02]

#### 3.8.1.5 Web-Foren

Außerdem konnten verschiedene Web-Foren identifiziert werden, welche zur direkten Kommunikation innerhalb der Anwender- und Entwicklergemeinde genutzt werden und dadurch als ergänzende Kommunikationskanäle betrachtet werden können. Von offiziellen Organen der Apache Organisation wird dieser Kanal in keinem relevanten Umfang verwendet oder koordiniert.

#### 3.8.1.6 Projektbezogene Websites

Neben den Plattformen, welche durch Projektmanagement-Instanzen der Apache Organisation bereitgestellt werden, existieren verschiedene inoffizielle projektbezogene Plattformen, die ergänzende Informationen, Anwendersupport oder Entwicklerforen zur Verfügung stellen. Die Entwicklungsprozesse werden aber ausschließlich über die projektimmanente Infrastruktur koordiniert und realisiert.

113 http://httpd.apache.org/dev/guidelines.html

### 3.8.2 Kollaborationswerkzeuge

Die im folgenden dargestellten Kollaborationswerkzeuge werden zentral durch die ASF bereitgestellt und verwaltet und durch die einzelnen Teilprojekte verwendet.

#### 3.8.2.1 Konfigurationsmanagement - CVS

Wie in den meisten OSS-Projekten, wird auch im Rahmen des Apache Projekts eine Instanz eines CVS-Repository instrumentalisiert und zur teilprojektübergreifenden Durchführung des Konfigurationsmanagements bzw. zur Koordination des heterogenen Entwicklungsprozesses verwendet: "Concurrent Versioning System (CVS) is the source repository used by all the Apache Software Foundation projects. Each project has their own branch of source code and documentation within the CVS." [AtGaGe02].

Der lesende Zugriff auf den Quellcode des Archivs wird jedem Entwickler ermöglicht, während der Schreibzugriff, respektive die Integration von Patches in das Repository nur entsprechend privilegierten Committers ermöglicht wird: "Apache committers have write access to the CVS repositories, enabling them to make changes to the source code. Everyone has read access to the repositories, […]." [ApacheOrg02][114]

Für den http-basierten Zugriff auf das zentrale Archiv steht eine webbasierte Benutzerschnittstelle zur Verfügung: "If you just wish to browse around or download a few individual files, the best tool is the web-based ViewCVS interface." [ApacheOrg02][114]

#### 3.8.2.2 Bug Tracking - Bugzilla

Für das Bug Tracking wurde analog zu CVS eine zentrale Instanz des Bugzilla-Systems installiert, welche von allen Teilprojekten der Apache-Organisation genutzt wird. Die konkreten Prozesse zur Nutzung des Systems wurden in der Prozessbeschreibung eingehend dargestellt, wobei [AtGaGe02] das primäre Einsatzgebiet im Rahmen der Fehler- und Anforderungserfassung identifiziert: "Users use this tool to report bugs found, and they also submit suggestions for new features and enhancements through this tool as well." [AtGaGe02]. Damit stellen die Informationen in Bugzilla den Ausgangspunkt für jegliche weitere Entwickungsaktivität dar und geben dem Project Management Committee eines Teilprojekts die Möglichkeit, sinnvolle Erweiterungen und Modifikationen

114 http://www.apache.org/foundation/cvs.html

des Systems zu identifizieren: "Committers, as well as the Project Management Committee, use this tool to track the general interests of the community, and based on that to decide the direction for the project." [AtGaGe02]. Da das Bugzilla-System bzw. die Metadaten der mit ihm verwalteten Anforderungsartefakte in erheblichem Umfang den Prozess der Patchentwicklung steuern bzw. abbilden, kommt dem Bugzilla-System auch eine koordinierende Aufgabe im Sinne eines Managementwerkzeugs zu.

Bugzilla ist ebenfalls eine Open Source Software und wurde ursprünglich in TCL entwickelt und später auf Perl portiert: "Bugzilla was originally written by Terry Weissman in a programming language called 'TCL', to replace a crappy bug-tracking database used internally for Netscape Communications. Terry later ported Bugzilla to Perl from TCL, [...]." [IssuesApOrg02][115]. Das System stellt im Kontext von Open Source eine weitreichend eingesetzte Ressource dar, welches die spezifischen Anforderungen des OSS-Entwicklungsprozesses in besonders probater Weise unterstützt: "It is now the de-facto standard defect-tracking system against which all others are measured." [IssuesApOrg02][115]

[IssuesApOrg02][115] beschreibt die folgenden Systemmerkmale von Bugzilla:

- Verwaltung der Bug Reports und Metadaten in einem RDBMS
- Erweiterte Funktionalitäten zur Generierung von Reports und Statistiken
- Flexible Konfigurierbarkeit und Adaptierbarkeit an die individuellen Anforderungen eines Projekts
- APIs
- Integrierbarkeit von verschiedenen Software Konfigurations Systemen wie CVS

Aufgrund der kontinuierlichen Weiterentwicklung des Systems durch die OS-Gemeinde, unterliegen diese Charakteristika einer ständigen Evolution.

### 3.8.3 Entwicklungswerkzeuge

Wie in den meisten OS-Projekten existieren keine projektweit eingesetzten Tools zur Unterstützung des lokalen Entwicklungsprozesses, da über diese Werkzeuge individuell durch die Entwickler und in Abhängigkeit von der jeweiligen Plattform entschieden wird: "There is no specific development tool recommended by the community for this project." [AtGaGe02]

115 http://issues.apache.org/bugzilla/docs/html/Bugzilla-Guide.html

#### 3.8.3.1 Compiler

Durch die Entwickler werden indivuell verschieden Compiler zu Erstellung von lokalen Builds ihrer Quellcodes instrumentalisiert. Diesbezüglich konnten keine Vorgaben identifiziert werden, wobei aber der GNU C-Compiler gcc auch in diesem Kontext von großer Bedeutung ist.

#### 3.8.3.2 Diff

Das Werkzeug `Diff` wird auch im Apachekontext als elementare Ressource durch die Entwickler zur Extraktion der lokalen Quellcodemodifikationen zu einem kommunizierbaren Patch verwendet. Es stellt damit eine wichtige Ressource dar und ermöglicht die direkte Integration eines Patches in das zentrale CVS-Archiv.

#### 3.8.3.3 GNU Debugger

Ein Debugger-Werkzeug dient zur systematischen Identifikation und Behebung von Softwarefehlern und wird individuell durch die Entwickler instrumentalisiert. Der GNU Debugger (`gdb`) stellt dabei eine weithin verbreitete Ressource dar, wie auch [AtGaGe02] im Kontext des HTTPD-Projekts identifiziert: "GNU debugger (`gdb`) is the recommended tool that the Apache HTTPD community uses to trace the problem within the software." Außerdem wird dies auch in den Debugging-Guidelines des HTTPD-Projekts thematisiert: "If you use the gcc or egcs compilers, it is likely that the best debugger for your system is gdb." [HttpdApOrg02][116]

#### 3.8.3.4 Editoren

Die Modifikation der Quellcodes wird durch die Entwickler mit den unterschiedlichsten Werkzeugen vorgenommen, wobei die etablierten Texteditioren die verbreitetsten Werkzeuge darstellen: "In the Jakarta project, developers and committers make code changes by using different tools (such as text editors, IDEs, etc)." [AtGaGe02]

116 http://httpd.apache.org/dev/debugging.html

# 4 Fallstudie des OSS-Projekts Mozilla

Das Mozilla-Projekt zeichnet sich durch einen relativ vorangeschrittenen Projektstatus aus und bietet damit einen adäquaten Rahmen, um die Strukturen und Prozesse, welche sich evolutionär im Verlauf des Projekts herausgebildet haben, zu identifizieren. Es wurde außerdem für eine weitere Evaluation ausgewählt, da sich das Mozilla-Projekt durch eine besondere Ausgangsbedingung auszeichnet. Die Initialisierung des Projektes begann mit der Offenlegung und Veröffentlichung des Quellcodes eines bisher proprietär geführten und geschlossenem Quellcode. Dies stellt ein im OSS-Kontext durchaus gebräuchliches Vorgehen dar, weswegen dieses Projekt als Repräsentant eines OSS-Projektes unter diesen Vorzeichen selektiert wurde.

Wie auch bei den anderen evaluierten OSS-Projekten, wird im folgenden zwischen dem initialen Entwicklungs- und Releaseprozess und dem sich daran anschließenden evolutionären Entwicklungsprozess durch die Projekt-Gemeinde unterschieden.

## 4.1 Mozilla-Projektüberblick

Das Mozilla-Projekt wurde durch die Offenlegung des Quellcodes des Netscape Communicators durch die Netscape Communications initialisiert: "Netscape Communications made two important announcements on January 23rd, 1998:

- First, that the Netscape Communicator product would be available free of charge;
- Second, that the source code for Communicator would also be free.

On March 31st, the first developer release of the source code to Communicator was made available." [MozillaOrg02][117]. Zielsetzung von Netscape war es hierbei, den Markanteil des Browsers zu vergrößern und dessen Softwarequalität durch die Integration neuer Entwickler in den Entwicklungsprozess zu verbessern: "When Netscape chose to turn the Communicator product into a open-source development project, the goal was to increase the development pace by incorporating contributions from the many developers present on the Internet, as well as to seed market for Netscape's enterprise solutions." [NoSt98]

117 http://www.mozilla.org/mission.html

Mit einer Vielzahl von Entwicklern gehört das Mozilla-Projekt inzwischen zu den größten im OSS-Kontext: "Since its creation in 1998, the project has attracted thousands of participants, and has arguably one of the largest communities working on an OSS project today." [ReMa02]. Als standard-kompatibler Web-Browser stellt die Mozilla-Suite ein Produkt von strategischer Wichtigkeit für die OSS-Entwicklergemeinde dar und füllt damit eine Marktlücke im OSS-Kontext: "Mozilla is very important to the Web development and free software communities, mainly because it fills a perceived gap for an Open Source, standards-compliant Web browser [...] " [ReMa02]. Mozilla wurde in C bzw. C++ entwickelt und besteht inzwischen aus einer vergleichsweise umfangreichen Code-Basis: "The Mozilla project is written in the C and C++ programming languages [...] and currently has more than two millions lines of code." [CaLeCh02]

### 4.1.1 Die Mozilla-Software

Der Name Mozilla wurde früher als Code-Name für den Netscape Navigator und den Netscape Communicator verwendet und repräsentiert heute, die aus diesen Produkten hervorgegangene Software-Suite: "Now, we intend to use the name Mozilla as the generic term referring to internet client software developed through our open source project." [MozillaOrg02][118]. Der Mozilla-Browser basiert auf verschiedenen, offenen Technologien, welche zu dem Mozilla Application Framework zusammengefasst werden können und nach [ReMa02] eine plattformunabhängige Suite bestehend aus Bibliotheken und Entwicklungssprachen darstellen. Die wichtigsten werden im folgenden kurz dargestellt:

- *XUL* - XML User Interface Language: XUL steht für "XML User Interface Language" und stellt eine plattformübergreifende und auf XML basierende Beschreibungssprache zur Spezifikation von Anwender Schnittstellen dar.
- *XBL* - eXtensible Binding Language: XBL dient der Modifikation von Elementen innerhalb von Dokumenten: "... a language used to modify the behavior of elements in documents." [ReMa02]
- *JavaScript*: Javascript ist ECMA-kompatibel und dient der Integration von skriptbasierter Applikationslogik in HTML-Dokumente
- *Gecko*: Gecko repräsentiert eine "cross-platform, embeddable layout engine" [ReMa02]

Dieses Framework repräsentiert nicht nur die Grundlage für Mozilla sondern auch für eine Anzahl anderer, z.T. auch kommerzieller Projekte: "The framework is used as the basis for a number of commercial products [...],

118 http://www.mozilla.org/mission.html

including CiTEC Doczilla, Tuxia Nanozilla, IBM Warpzilla and the OEone Operating Environment [...] " [ReMa02]

### 4.1.2 Spezifische Merkmale des Mozilla-Projekts

Neben der bereits dargestellten Ausgangssituation, dass das Projekt auf eine ehemals proprietäre und unter kommerziellen Gesichtspunkten entwickelte Quellcodebasis zurückgeht, stellen sich an die Software des Produktes verschiedene Anforderungen, deren Fokus als eher untypisch für den OSS-Kontext zu betrachten ist. Die Zielgruppe der Endanwender sollte beispielsweise von Initialisierung des Projekts an durch eine einfache Usability erreicht werden: "The Mozilla Project aims to create a polished, easy-to-use application for end-users of widely varying computer skills, whereas the majority of OSS projects concentrate on applications where the developer is also a domain expert." [ReMa02]. Diese explizite Anforderung kann als eher unüblich für OSS-Projekte betrachtet werden und stellt auch für viele Entwickler im Kontext der Mozilla-Entwicklungsprozesse eine Kontroverse dar: "This presents requirements that most OSS projects never face, and developers must often achieve consensus on controversial issues with which they may not be familiar. Easy installation and usability, for instance, are two examples of potentially unfamiliar territory to developers." [ReMa02]

Weiterhin zeichnet sich das Mozilla-Projekt dadurch aus, dass neben der primären Entwicklung der Mozilla-Suite auch verschiedene periphere infrastrukturelle Software-Tools entwickelt wurden und werden, welche den Prozess der Software-Entwicklung unterstützen und z.T. auch schon in vielen anderen OSS-Projekten zum Einsatz kommen. Hierbei ist vor allem das weit verbreitete Bug-Tracking-System "Bugzilla" zu nennen. Der Abschnitt "Werkzeuge" wird diese Thematik weiter konkretisieren.

Im Rahmen des Mozilla-Projekts wird außerdem eine individuelle Terminologie verwendet. So wird der ursprüngliche Quellcodepfad im CVS-Archiv als 'trunk' bezeichnet, während die davon ausgehenden Quellcodeverzweigungen traditionell 'branches' genannt werden: "In CVS there are several versions of code; the base code is referred to as the 'trunk', while spin off versions are called 'branches'." [CaLeCh02]. Außerdem verwaltet Tinderbox, ein Werkzeug welches an adäquater Stelle explizit erläutert wird, Quellcode-Verzeichnisse zur Kompilation als sogenannten 'Tree'. Daher wird das Konfigurationsmanagement des Quellcodes auch als 'Tree Management' bezeichnet: "In Mozilla, configuration management is therefore often referred to as 'tree management.'" [CaLeCh02]

Ein weiteres sehr bedeutendes Merkmal der praktizierten Prozesse im Mozilla-Projekt ist der sehr hohe Grad der Formalisierung und Reglementierung der Prozesse durch die Mozilla Organisation. Um den Ansprüchen der

Qualitätssicherung gerecht zu werden, wurden von der Mozilla Organisation eine Vielzahl von Rollen und Prozessen definiert, die durch einen Kern von Akteuren ausgeführt werden, die der Mozilla Organisation nahe stehen oder direkt entstammen. Diese Determinierung der freien Prozessevolution ist für die Softwareentwicklung im OSS-Kontext sehr untypisch und daher ein klares Distinktionsmerkmal des Mozilla-Projekts von vielen anderen OSS-Projekten. Trotzdem ist auch diese Reglementierung in vielen anderen OSS-Projekten, die ebenfalls ursprünglich in einem kommerziellen Kontext entwickelt wurden, in ähnlicher Form präsent.

### 4.1.3 Organisation

Nach dem Release des Source Codes wurde ein Großteil der Entwicklung anfänglich weiter innerhalb von Netscape ausgeführt, während die Mozilla Organisation gegründet wurde, um die Entwicklung im OSS-Kontext zu koordinieren: "When the source code was released the product development continued within Netscape, and a small group of people were assigned to coordinate the development project, called the Mozilla organization." [NoSt98]. Im folgenden soll zum besseren Verständnis der Entwicklungsprozesse die grobe Struktur der Mozilla Organisation dargestellt werden.

#### 4.1.3.1 Die Mozilla Organisation

Netscape gründete die Mozilla-Organisation, welche als zentrale Instanz den Prozess der Softwareentwicklung unterstützen soll: "The Mozilla Organization (mozilla.org) is a group which exists to support the development of the browser suite. mozilla.org is responsible for managing, planning and providing server resources to support the development of Mozilla." [ReMa02]

Primär sind die Mitglieder für Aufgaben verantwortlich, die das gesamte Projekt betreffen, und delegieren anderweitige Zuständigkeiten an explizit ausgewählte Mitglieder der Community, wie dies z.B. anhand der dedizierten Rollen "Driver" oder "Modul Owner" praktiziert wird, welche in Kapitel 4.5 "Rollen" explizit erläutert werden: "Issues that affect the entire project include such things as the roadmap and milestone planning, getting to Mozilla 1.0, code review processes, handling of security-related bugs, reorganization of the Mozilla website (where we are way behind), etc. [...] For issues whose scope is less broad, staff tries to delegate authority to community members who have expertise with a particular area [...]" [MozillaOrg02][119]. Dabei sind die

119 http://www.mozilla.org/about/roles.html

einzelnen Mitglieder nach [ReMa02] für verschiedene Aufgabenbereiche wie Website Maintenance, Dokumentation, Architektur-Design oder Release Management zuständig. "To a large extent, it [the Mozilla Organisation] is also responsible for developing and enforcing the Mozilla process, and acts as the final arbitrator between disputes over changes to the codebase." [ReMa02]

Über die Website und ein Werkzeug für das kollaborative Konfigurationsmanagement wird der Quellcode durch die Organisation zur Verfügung gestellt: "The Mozilla organization provides access to the source code by means of a version control system, and by releasing code snapshots on a regular basis." [NoSt98]. Weiterhin definiert die Mozilla Organisation die strategische Ausrichtung und prinzipielle Anforderungen an die Software und koordiniert die Integration von Patches in die offiziell zu veröffentlichenden Versionen des Softwareprodukts: "The role of this organization is to provide directions for further improvements to the software and to integrate contributions into the source-tree, and also to help open-source developers reach consensus in order to coordinate development." [NoSt98]

Die Mozilla Organisation definiert sich außerdem [MozillaOrg02][120] selbst als Code Integrator: "We are code integrators. And, through our forums, we will try to help people reach consensus, and thereby provide direction and coordination for future. improvements."

Die Mozilla Organisation setzt sich aus ausgewählten Mitgliedern der Entwicklergemeinde zusammen, welche die verschiedenen Projekte leiten: "which act as managers and technical lead for the various Mozilla projects." [ReMa02]. Momentan besteht die Mozilla Organisation aus 14 Mitgliedern, die zum Teil für diverse Unternehmen tätig sind [MozillaOrg02][121] "Staff members work for different companies and represent different interests, but we share the common task of making the organization and project infrastructure work well so that everyone (ourselves included) can focus on making the code work better. Many but not all of us are paid by our respective employers to work on mozilla.org activities." [MozillaOrg02][122]. Eine optimale Anzahl der Mitglieder der Organisation konnte bisher noch nicht eruiert werden: "We have not determined whether there is an ideal size for staff." [MozillaOrg02][122]. Von potentiellen neuen Mitgliedern wird erwartet, dass diese bereits für das Projekt tätig waren und bereits Schlüsselrollen im Rahmen des Projektes eingenommen haben, welche an geeigneter Stelle genauer dargestellt werden: "We anticipate that future staff members will have been Staff Associate members and

120 http://www.mozilla.org/mission.html
121 http://www.mozilla.org/about/stafflist.html#Staff-Members
122 http://www.mozilla.org/about/roles.html

probably a Module Owner or Activity Owner as well before being asked to joining mozilla.org staff." [MozillaOrg02][123]

#### 4.1.3.1.1 Entscheidungsprozess

Die Entscheidungsfindung innerhalb der Organisation basiert auf einem Prinzip der Konsensbildung und nicht auf definierten Entscheidungshierarchien: "Since mozilla.org is a virtual organization, there is no formal employment or authority relationship among the staff. Nevertheless, we operate with a certain sense of order, shared responsibility, consensus building, and trust. We seek a general consensus among staff, and we expect a sense of give and take." [MozillaOrg02][123]. Falls ein übergreifender Konsens nicht erreicht werden kann, wird je nach den erforderlichen Qualifikationen eine von zwei zentralen Personen mit klaren Kompetenzen konsultiert: "When consensus is difficult or an individual decision-maker is needed, we look to Brendan Eich for technical matters, and Mitchell Baker for policy and organizational matters." [MozillaOrg02][123]

#### 4.1.3.1.2 Staff Associates

"Staff Associates provide specialized support to the Mozilla community through their involvement with and contribution to mozilla.org staff." [MozillaOrg02][123]. Diese Rolle wurde von der Mozilla Organisation im Frühjahr 2001 eingeführt, umfasst nach [MozillaOrg02][124] momentan nur ein Mitglied und ist daher nur von marginaler Bedeutung. Die Aufgaben betreffen vor allem sekundäre Beiträge zur Entscheidungsfindung der Mozilla-Organisation: "An Associate supports mozilla.org staff by providing input into staff discussions and decision-making process. An Associate may also investigate issues and provide background materials to staff." [MozillaOrg02][123]. Außerdem sollte sich nach [MozillaOrg02][123] ein Inhaber dieser Position durch weitreichende Erfahrung im Rahmen des Projektes und ein ganzheitliches Interesse an dessen strategischer Ausrichtung auszeichnen.

### 4.1.3.2 Community

"Another important aspect of the project are the people that make it happen. The Mozilla web browser project is successful because it is a community that

123 http://www.mozilla.org/about/roles.html
124 http://www.mozilla.org/about/stafflist.html

includes many volunteers, paid contributors, and mozilla.org staff (to name a few roles)." [CaLeCh02]

Indem die Akteure der OSS-Community für alle konstruktiven Beiträge zum OSS-Entwicklungsprozess verantwortlich zeichnen, determinieren sie die Entwicklung der OSS und des gesamten Projekts in entscheidendem Maße: "The Mozilla community includes all those who contribute to Mozilla: writing code, testing software, writing documentation, developing web pages and applications, advocating on behalf of Mozilla, or doing any of the multitude of other things that help make Mozilla useful and successful." [MozillaOrg02][125]. Dabei setzen sich die Akteure aus individuellen Freiwilligen, Mitarbeitern des Bildungssektors oder Mitarbeitern von kommerziellen Unternehmen zusammen: "Some participate as individual volunteers, some through their educational institution, and others work at commercial companies." [MozillaOrg02][125]

Ein weiteres Differenzierungsmerkmal der Mozilla-Community von anderen OSS-Communities stellt die kontinuierliche weitere Mitwirkung verschiedener Netscape-Entwickler als Akteure im OS-Entwicklungsprozess dar: "As the Communicator product is still being developed by Netscape, the different teams of programmers now act as contributors to the Mozilla organization in precisely the same manner as any other contributor." [NoSt98]

Die Entwicklung des Projektes wird durch eine sehr umfangreiche Community vorangetrieben, wobei verschiedene Entwickler z.T. direkt von Unternehmen finanziert werden, welche die Entwicklung von Mozilla unterstützen: "The number of developers is high, many of them being directly paid by Netscape, OEone, Sun, IBM and other companies that fund development of the browser suite and framework." [ReMa02]. Außerdem umfasst die Projektgemeinde inzwischen eine, relativ zu anderen OSS-Projekten extrem große Zahl von freiwillig Mitwirkenden: "Bugzilla, the bug-tracking software used for Mozilla registers over 16,000 active users, over 14,000 unique bug reporters, and over 1,000 'trusted' people who can perform tasks such as confirming and editing bugs." [ReMa02]. Die folgende Tabelle zeigt die aktiv Mitwirkenden, welche von Juli 2001 bis Februar 2002 Modifikationen am Quellcode vorgenommen haben.

125 http://www.mozilla.org/about/roles.html

Tabelle 4 Mozilla: Statistik der Code-Beiträge (Quelle: [ReMa02])

| | Jul 01 | Aug 01 | Sep 01 | Oct 01 | Nov 01 | Dec 01 | Jan 02 | Feb 02 (incomplete) |
|---|---|---|---|---|---|---|---|---|
| **Developers submitting code** | 143 | 160 | 152 | 157 | 158 | 150 | 159 | 104 |
| **New developers submitting code** | 2 | 11 | 7 | 5 | 16 | 5 | 12 | 1 |
| **Total code submissions** | 1577 | 1892 | 1997 | 2355 | 2348 | 1594 | 2083 | 466 |

Weiterhin ist es möglich, neben der Mozilla-Organisation weitere Kern-Mitglieder der Entwicklergemeinde zu identifizieren, welche sich durch besonders häufige Mitwirkung am Projekt auszeichnen und auch über erweiterte Privilegien verfügen: "However, common contributors may be granted certain privileges over time (such as write access to the source control repositories) which can help identify the core members." [ReMa02]. Diese spezifischen Privilegien können zu verschiedenen Rollen aggregiert werden, die durch die Mozilla Organisation definiert wurden: "mozilla.org may also define a special role for developers and community members that are particularly skilled or committed to the project. These roles include super reviewer, module owner, Bugzilla component owner and driver." [ReMa02]. In Kapitel 4.5 werden diese Rollen noch detailliert dargestellt.

### 4.1.4 Softwarelizenz

Speziell für dieses Projekt wurden zwei neue Lizenzen entwickelt, respektive die Mozilla Public License (MPL) und die Netscape Public License (NPL), wobei die MPL nach [FeFi02] in der aktuellen Version durch die OSI zertifiziert ist und somit eine echte OSS-Lizenz darstellt, während die NPL spezielle Klauseln enthält, welche ausschließlich der Netscape Organisation die kommerzielle Verwendung der Software und die Relizensierung der Software einräumt: "... the NPL is not OSI-certified, and contains specific clauses which allow Netscape (and only Netscape) to re-licenss third-party Mozilla developments to create a proprietary product." [FeFi02]

## 4.2 Entwicklungsprozesse

In diesem Abschnitt werden alle identifizierten Prozesse, Teilprozesse und Aktivitäten dargestellt, die im Rahmen des Mozilla-Projekts identifiziert werden konnten und basierend auf dem Metamodell als Entwicklungsprozesse klassifiziert werden können.

### 4.2.1 Allgemeine Merkmale

Die Mozilla-Entwicklung basiert auf den typischen Charakteristika der OSS-Entwicklung: "Mozilla software project follows the model of distributed development and centralized integration." [CaLeCh02]. Weiterhin wird darauf hingewiesen, dass im Mozilla-Kontext der Terminus 'Bug' als gleichbedeutend mit jeglicher Art von Anfrage zur Modifikation des Quellcodes zu betrachten ist und diese Art der Entwicklung auch als 'Bug-driven' bezeichnet wird: "Every code change in the Mozilla codebase is made as part of a 'fix' for a uniquely numbered 'bug'." [ReMa02]. Damit umfasst der Begriff 'Bug' in der Terminologie des Mozilla-Projekts jegliche Anforderungsartefakte, welche einen Änderungsvorschlag bezüglich einer Fehlerbeseitigung oder einer funktionalen Systemerweiterung definieren: "Though it commonly has a pejorative connotation, in the Mozilla Project the term bug is used to refer to any filed request for modification in the software, be it an actual defect, an enhancement, or a change in functionality." [ReMa02]

Die folgenden Abschnitte beschreiben den allgemeinen Prozess der Entwicklung durch die Entwicklergemeinde und stellen die relevanten Teilprozesse explizit dar, wobei diesem klassischen OSS-Entwicklungsprozess aber der Prozess der initialen Prototypentwicklung und -veröffentlichung vorausgegangen ist, der in den Managementprozessen näher dargestellt wird.

#### 4.2.1.1 Re-Engineering entsprechend der OSS-Anforderungen

Um den eigenen Anforderungen entsprechen und eine heterogene und kollaborative Bearbeitung des Quellcodes und die damit verbundene Evolution der Software-Architektur auf adäquate Weise gewährleisten zu können, wurden einige Quellcode-Module komplett überarbeitet. Diese Re-Engineering-Phase dient dazu, den Anforderungen des OSS-Entwicklung gerecht zu werden, indem eine modulare Architektur realisiert und soweit möglich ausschließlich standardisierte Komponenten und Methoden verwendet werden.

Ein Beispiel für eine grundlegende Überarbeitung des Quellcodes ist nach [ReMa02] z.B. die Layout-Engine, die an die Anforderungen des W3C adaptiert werden musste. Durch diesen Mehraufwand konnte die erste stabile Version 1.0 des Mozilla-Application-Frameworks und der integrierten Technologien wie Gecko, XBL oder XUL erst im Juni 2002 nach einer Entwicklungsphase von vier Jahren veröffentlicht werden: "Release 1.0 is expected to finally consolidate these technologies into a stable platform for development." [ReMa02]

Außerdem veränderte der Prozess der grundlegenden Überarbeitung des Quellcodes den Fokus des Projektes im Rahmen dieser Re-Engineering-Phase: "The rewriting activity also changed the project's effective maturity. It shifted from being a project devoted to browser maintenance, with gradual

enhancement and refactoring, to an 'initial implementation', where new features are specified and designed from scratch." [ReMa02]

Diese neuen Anforderungen der Kompatibilität zu existierenden Standards und vor allem der hohen Modularität zur Ermöglichung von verteilten kollaborativen Entwicklungsprozessen, führten außerdem zu der Entwicklung der verschiedenen neuen Technologien wie z.B. Gecko, XUL oder XBL.

#### 4.2.1.2 Prozedurale Anforderungen für kollaborative Entwicklung

Dem zugrundeliegenden Entwicklungsprozess, der z.T. eine direkte Weiterentwicklung der Netscape-Praktiken darstellt, wurde im Rahmen des Mozilla-Projekts stets eine hohe Bedeutung zugemessen. Diese Evolution des Entwicklungsprozesses wurde vor allem von den Anforderungen der OSS-Gemeinde geleitet, die nach [ReMa02] sowohl Anforderungen an die Prozesse als auch an die unterstützenden Werkzeuge umfassen. [ReMa02] definiert diesbezüglich z.B. eine effiziente Versionskontrolle des Quellcodes und klar definierte Prozesse und Verantwortlichkeiten für die Integration von Quellcodemodifikationen in den zentralen Entwicklungsquellcode.

Aufgrund der hohen qualitativen Ansprüche der Mozilla Organisation, ergeben sich ebenfalls direkte Konsequenzen für die Entwicklungs- und Managementprozesse und die Formalisierung und Reglementierung der Systementwicklung. Aus diesen Anforderungen lassen sich außerdem verschiedene Anforderungen an die Infrastruktur des Projektes ableiten, auf welche die Abschnitte "Infrastrukturelle Prozesse" und "Werkzeuge" (Kapitel 4.4 und Kapitel 4.7) noch konkret eingehen werden.

#### 4.2.1.3 Anforderungen an die Softwarearchitektur

Um die plattformübergreifende Verwendbarkeit und einen möglichst hohen Modularisierungsgrad erreichen zu können, wurde das bereits beschriebene Mozilla Application Framework entwickelt. Dadurch basiert der eigentliche Browser auf simplen Front-End Technologien, deren Verarbeitung durch das Framework ermöglicht wird: "The browser itself is a front end written mainly in Javascript, XUL and XBL, and interpreted at runtime. The framework constitutes the back end code and is written in XPCOM and C++" [ReMa02].

Dieses Framework ist plattformunabhängig, was durch die Verwendung der plattformübergreifenden Komponentenbibliothek XPCOM gewährleistet wird, und entspricht den Anforderungen der Modularität, wodurch eine verteilte Entwicklung im Sinne von Open Source möglich ist: "This design is by nature modular, and parts such as the Javascript engine, the runtime libraries, and the framework itself can be (and have been) reused independently of the browser

to develop other products. Modularity, finally, is what makes possible development in spite of the natural concurrency presented by a distributed developer community." [ReMa02]

### 4.2.2 Allgemeiner Entwicklungsprozess

Dieser Abschnitt beschreibt die im Mozilla-Projekt praktizierten Entwicklungsprozesse und stellt diese im Kontext der klassischen Systementwicklungsphasen dar. Die elementaren Teilprozesse werden im Laufe dieser Studien noch an späterer Stelle explizit dargestellt.

#### 4.2.2.1 Anforderungsanalyse

Die High-Level-Anforderungen der Mozilla Organisation bilden lediglich die Rahmenbedingungen für den praktizierten OSS-typischen Prozess der Ad-Hoc-Anforderungsanalyse basierend auf den bereits beschriebenen Bug Reports: "High-level requirements are laid down by mozilla.org management, but since these are few and very abstract, such as 'complete Web Standards compliance', most of the decisions on functionality inclusion and change are discussed piecemeal by the community and module owners through bug and newsgroup comments." [ReMa02]

Die Anforderung von Änderungen am Quelltext werden i.d.R. über einen der folgenden Kommunikationskanäle (Mailing List, Newsgroup) kommuniziert oder direkt im verwendeten Bug Tracking System (Bugzilla) generiert. Sofern die Kommunikation der Anforderung kein negatives Feedback nach sich gezogen hat, führen auch diese Aktivitäten zur Generierung des eigentlichen Bug Reports im Bug Tracking System: "Other people will usually comment on relevance and discuss advantages and disadvantages of the approach. The essence of the debate is often quite technical and will usually culminate in a bug being filed on the change, or the idea being abandoned." [ReMa02]

Im Falle der direkten Erstellung eines 'Bug Reports' erfolgt die Diskussion der Relevanz der Anforderung mit dem Ziel der Verifikation meist implizit nach der Generierung: "Discussion often occurs in the bug itself, and the vote system, community members, module owner, and ultimately mozilla.org staff will help determine if it is a relevant change or not." [ReMa02]. Eine derartig erstellte und kommunizierte Anforderung bietet aber auch nach ihrer Verifizierung durch die Entwicklergemeinde keinerlei Garantie für eine spätere Implementierung bzw. die Integration der implementierten Änderung in das zentrale Quellcode-Archiv. Die Implementierung kann nur durch die Entwickler ausgeführt und durch die Mozilla Organisation forciert werden. Bezüglich der Integration in das offizielle Quellcodeverzeichnis stellen entsprechend privilegierte Entwickler, wie z.B. die Modul Owner und die Mitglieder der

Mozilla Organisation die höchste Entscheidungsgewalt dar: ".... module owners and mozilla.org staff are the final authorities for determining the relevance of a change..." [ReMa02]

#### 4.2.2.2 Entwurfsprozess

Die Vorgaben der Anforderungsdefinitionen bestimmen in großem Maße die Modifikationen des Softwaredesigns. Die meisten Entwurfsentscheidungen werden daher nicht im Rahmen eines expliziten Prozesses sondern im Rahmen der Diskussion der Bug Reports bzw. der darauf basierenden Patches getroffen: "As reflected by the largely ad-hoc requirements process, most design issues are tackled as they come: bugs are filed for changing the design or API of a certain component or class, and the ordinary discussion ensues over bug comments." [ReMa02]

Das Design der Software wird häufig nur implizit durch den Code spezifiziert, wodurch sich an die Strukturierung der Quelltexte sehr hohe Anforderungen stellen: "Because of the 'continuous design adjustment', there is a constant need to perform refactoring of the legacy code." [ReMa02]. Das Refactoring, also die Überarbeitung und Verbesserung der Quellcodestruktur und -qualität findet nach [ReMa02] u.a. durch die Vereinfachung von APIs, die Entfernung veralteter Code-Elemente und die Verbesserung der Modularität und Lesbarkeit statt, wobei dies wiederum nicht durch einen expliziten Teilprozess sondern als inhärenter Bestandteil der schrittweise Softwareverbesserung im Rahmen der Patchentwicklung ausgeführt wird.

Die Verantwortlichkeit für Änderungen am Design spezieller Module liegt prinzipiell bei allen aktiven Entwicklern, wobei in diesem Kontext die Rollen des "Super Reviewers" bzw. "Modul Owners" einen besonderen Stellenwert einnehmen: "Both are ultimately responsible for judging and allowing the design to evolve in a certain direction." [ReMa02]

Ein großes Problem dieses freien Entwicklungsprozesses und der damit verbundenen stetigen Evolution des Designs und des Quellcodes, stellt die Aktualisierung der Dokumentation dar, die häufig nicht mit der Geschwindigkeit der Quellcode-Modifikationen ausgeführt wird: "One issue this constant change produces is the burdensome task of keeping documentation up to date, and the documents on mozilla.org have aged considerably over these years." [ReMa02]

#### 4.2.2.3 Implementation

Die Quellcodeverwaltung erfolgt zentral über ein Revisions Control System, respektive über CVS (Controlled Versioning System), welches an geeigneter

Stelle explizit erläutert wird: "All developers work using revision control [...] on a common, centralized, codebase, which allows changes to be developed concurrently and independently 'checked in'." [ReMa02] In CVS wird ein zentrales Verzeichniss des Quellcodes verwaltet und dadurch jedem Entwickler der Quellcode zur Verfügung gestellt. Interessierte Entwickler können sich zur Bearbeitung den gesamten Quellcode oder Teile davon auf ihre lokalen Rechner kopieren, wobei CVS die spätere Integration und das Management von miteinander konkurrierenden Änderungen steuert und koordiniert.

Werkzeuge zur Echtzeit-Kommunikation stellen wichtige Kommunikationskanäle für den Implementations- und Reviewprozess dar. Die Implementationsphase wird daher primär im Rahmen vieler dezentraler, parallel durch einzelne Entwickler ausgeführter Patchentwicklungszyklen durchgeführt.

Die so generierten Patches beinhalten alle Änderungen im Quelltext des entsprechenden Code-Moduls relativ zum Ausgangsquelltext und werden als Text-Datei im ASCII-Format extrahiert und kommuniziert.

#### 4.2.2.3.1 Review

Den verschiedenen Review-Prozessen kommt im Rahmen der OSS-Entwicklung im allgemeinen und im Mozilla-Kontext im Speziellen eine sehr hohe Bedeutung zu: "Perhaps one of the most striking features of the process is the strict code review and approval system that code changes go through before integration." [ReMa02].

Der Qualitätssicherung wird im Mozilla-Kontext eine besondere Bedeutung beigemessen, wobei dies durch die Etablierung eines werkzeuggesteuerten formalen Review-Prozesses erreicht werden soll: "While other projects do include review as one of their base activities (Eric Raymond's 'eyeballs' hypothesis) the Mozilla Project is one of the first to systematically implement tool-supported formal review." [ReMa02]. Diese starke Reglementierung ist ein im OSS-Kontext eher untypischer Umstand und soll die Integration und Veröffentlichung von fehlerhaftem Quellcode vermeiden und ein hohes Qualitätslevel sicherstellen. Neben den bereits frühzeitig instrumentalisierten üblichen Code Reviews wurden sogenannte 'Super Reviews' eingeführt, welche einen Review durch einen Entwickler beinhalten, der mit dem entsprechenden Quellcode-Artefakt besonders bekannt ist. Diese speziellen Reviewprozesse wurden für besonders kritische Entwurfs-Entscheidungen etabliert und haben sich sehr positiv auf die Softwarequalität ausgewirkt: "[...] developers acknowledge that code quality went up significantly once it started." [ReMa02]

Der Review-Prozess wird in [ReMa02] wie folgt beschrieben:

1. Entwickler generiert Patch

2. Assoziation des Patches mit einem 'Bug Report' im Bugzilla System
3. Entwickler initialisiert Review-Prozess
4. Review

Der erste Schritt umfasst die Entwicklung eines Patches basierend auf einer Änderungsanfrage in einem 'Bug Report'. Anschließend kommuniziert der Entwickler dieses Patch über die Bugzilla-Datenbank, indem er das Patch dem korrellierenden 'Bug Report' hinzufügt und kommuniziert eine Anfrage um Review des Codes über die gebräuchlichen Kommunikationskanäle des Projekts. Der Reviewer bestätigt die Änderungen oder bittet den Entwickler um weitere Änderungen: "A reviewer, which can be the module owner or anyone else familiar with the code, will then read the code critically and either grant review or ask for changes." [ReMa02]. Einige Module erfordern den Prozess des 'Super Review', der eine Überarbeitung des Codes durch einen Entwickler erfordert, der über spezielle Kenntnisse des entsprechenden Quellcode-Artefakts verfügt.

#### 4.2.2.4 Einsatz

Nach erfolgter Integration von Patches in ein offizielles Release steht die modifizierte Software für die Verwendung durch alle interessierten Akteure zur Verfügung. Die Releaseprozesse werden nur durch entsprechend privilegierte Akteure ausgeführt. Dieser Phase können somit auch die eigentliche Anwendung der OSS und die der Anwendung immanenten Testprozesse zugeordnet werden.

### 4.2.3 Aktive Mitwirkung

Aus den dargestellten Spezifika der Mozilla-Entwicklung lassen sich die folgenden Teilprozesse ableiten, welche mögliche Anwendungsszenarien im Rahmen der Mozilla-Entwicklung repräsentieren und in den folgenden Abschnitten konkret beschrieben werden.

Das prinzipielle Vorgehen für interessierte Entwickler beschreibt [MozillaOrg02][126] wie folgt:

- Bezug der Software
- Anlegen eines Bugzilla-Accounts
- Softwarefehler berichten
- Entwicklung von Patches

126 http://www.mozilla.org/get-involved.html

Weiterhin definiert [MozillaOrg02][127] verschiedene Szenarien, welche Handlungsoptionen für interessierte Anwender darstellen, die am Projekt mitwirken möchten:

- Verfassen von Dokumentationsartefakten
- Beitrag von Bug Reports und Enhancement Requests
- Review der Browserbezogenen Bug Reports
- Durchführung von Test Cases
- Test der Mail-Engine
- Verbesserung der Performance

Im folgenden werden die wichtigsten Anwendungsfälle detaillierter dargestellt. Wichtige Bedingung für die Durchführung von zeitaufwendigen Aufgaben ist die initiale Kommunikation des Vorhabens über geeignete Kommunikationskanäle: "Whenever you begin work on something that's more than a day's work, we strongly recommend that you let people know about it. Post your plans to an appropriate newsgroup..." [MozillaOrg02][127]

### 4.2.4 Bezug der Software

Dieser Abschnitt beschreibt verschiedene Möglichkeiten, um sowohl den Quellcode, als auch ausführbare Binärdateien der Mozilla-Software zu beziehen.

#### 4.2.4.1 Bezug der Binaries

Die so bereitgestellten binären Releases basieren stets auf dem Quellcode des jeweiligen Release und werden in einem Kompilationsprozess erzeugt, um den Endanwendern die individuelle und lokale Kompilation abzunehmen und somit eine größtmögliche Zielgruppe erreichen zu können.

Von der URL http://www.mozilla.org/releases/ können die Binärdateien verschiedener Releases für die folgenden Plattformen heruntergeladen werden:

- Win32
- MacOS 9.x
- MacOS X
- Linux
- AIX
- HPUX
- OS/2

127 http://www.mozilla.org/get-involved.html

Weiterhin ist es auch möglich die Dateien über den projekteigenen FTP-Server zu beziehen: http://ftp.mozilla.org/pub/mozilla/releases.

#### 4.2.4.2 Bezug des Quelltextes

Der Quelltext kann nach [MozillaOrg02][128] auf verschiedene Weise bezogen werden:

- Online-Quellcodenavigation über LXR
- Download via FTP
- Download via CVS

Auf dem zur Verfügung stehenden FTP-Server wird im monatlichen Intervall der Quellcode aktualisiert wird: "Source drops to the FTP server usually occur monthly." [MozillaOrg02][128]. Für den FTP-Download des Quellcodes kann die gleiche URL wie für den Download der binären Versionen verwendet werden.

Natürlich steht auch der Download über den HTTP-Server des Projekts zur Verfügung: http://www.mozilla.org/download-mozilla.html .

Über das Tool LXR, welches in Kapitel 4.7.2.4 explizit erläutert wird, kann direkt durch den verlinkten Quellcode navigiert werden: "Browse the up-to-the-minute latest version of the source code online, through a massively-hyperlinked source code browser that lets you cross-reference function and variable names." [MozillaOrg02][128]

##### 4.2.4.2.1 Quellcode via CVS

Das Concurrent Versions System ermöglicht die parallele und kollaborative Bearbeitung von Quellcode und koordiniert die dazu nötigen Aktivitäten.

Über das Revisionskontrollsystem werden ebenfalls alle Quellcodeversionen zur Verfügung gestellt. CVS fungiert als zentrale Quellcodebasis für die verteilte Entwicklung und repräsentiert somit im Mozilla-Kontext das stets aktuellste Quellcoderepository: "Those doing active development can check out the latest source using CVS. This is the preferred method, as it lets you get up-to-the-minute changes and merge them with your own." [MozillaOrg02][128]. Der Quellcode ist für jeden Akteur über CVS verfügbar, wobei aber nur bestimmte Mitglieder der Community das Privileg besitzen, den Quellcode über CVS direkt in die aktuellen Module zu integrieren: "Anyone can check out the sources via

128 http://www.mozilla.org/source.html

CVS, but only certain people have the ability to check in. Those people, basically, are the module owners and their delegates." [MozillaOrg02][129]

### 4.2.5 Subscribe to Mailing List/Newsgroup

Die Möglichkeit, Mitglied einer Mailing Liste zu werden, steht generell jedem offen: "If you choose to subscribe to one of the mailing lists, rather than reading it as news, do this by sending mail to the appropriate -request address." [MozillaOrg02][130]. Gleiches gilt für die Lektüre von Newsgroups.

### 4.2.6 Fehlergetriebene Entwicklung

Da die Kategorisierung der Patchentwicklung als Bug Report-getrieben bezeichnet werden kann, werden in diesem Abschnitt einige Charakteristika der Bug Reports vorgestellt, um ein Verständnis der einzelnen Entwicklungsprozesse zu ermöglichen.

Als zentrales Repository zur strukturierten Verwaltung der Bug Reports fungiert das Bugzilla-System, welches auch durch die Community des Mozilla-Projekts entwickelt wurde: "The Mozilla.org community (including volunteer testers and QA engineers) uses Mozilla Bugzilla to document existing bugs in the form of bug reports." [CaLeCh02]

Wie bereits dargestellt, kann die Quellcode-Entwicklung auch als "Fehlergetriebene Entwicklung" bezeichnet werden, da jegliche Quellcodeänderung i.d.R. auf einem 'Bug Report', also einem Eintrag im Bugzilla-System basiert. Dabei kann ein Bugzilla-Eintrag entweder einen Bericht über einen Softwarefehler oder eine allgemeine Anforderung für eine funktionale Systemerweiterung enthalten: "Though it commonly has a pejorative connotation, in the Mozilla Project the term bug is used to refer to any filed request for modification in the software, be it an actual defect, an enhancement, or a change in functionality." [ReMa02]. Da ein Bug Report im Laufe seines Lebenszyklus ständig modifiziert wird und durch seine Metadaten auch das darauf basierende Patch und den jeweiligen Patchentwicklungszyklus beschreibt und steuert, kommt diesem Artefakt im Kontext der Entwicklungsprozesse eine besondere Bedeutung zu.

Die Zahl der in Bugzilla registrierten und über Bugzilla bearbeiteten Softwarefehler ist immens: "As of February 2002, there have been over

129 http://www.mozilla.org/cvs.html
130 http://www.mozilla.org/community.html

123,000 bugs reported in the Bugzilla defecttracking tool, with over 80,000 bug reports for the Browser product alone." [ReMa02]

#### 4.2.6.1 Format eines Bug Reports

Bugzilla dient zur strukturierten Verwaltung der Bug Reports über verschiedene Metadaten, welche die jeweilige Anforderung beschreiben.

Identifizierbar sind 'Bug Reports' anhand eines eindeutigen Schlüssels: "All change requests and their associated implementations have a unique number which identifies them." [ReMa02]. In [ReMa02] werden die wichtigsten Metadaten eines 'Bug Reports' in Form von Attributen wie folgt charakterisiert:

- **Owner**: Inititator des Eintrages
- **Summary**: Zusammenfassende Beschreibung des Softwarefehlers
- **Attachments**: Dateien, die in direktem Zusammenhang mit dem Softwarefehler stehen; z.B.: Test Cases, Screenshots oder etwaige Patches. In frühem Status dient dieses Feld der Reproduzierbarkeit und Beschreibung des Fehlers und dient in der späteren Folge als Container für die darauf basierenden Patches.
- **Comments**: Kommentare, die das Problem und mögliche Lösungswege beschreiben; stellt wichtige Informationsquelle für neu hinzugekommene Entwickler dar
- **Status**: Beschreibung des aktuellen Status mit folgenden Ausprägungen: UNCONFIRMED, NEW, ASSIGNED, RESOLVED, REOPENED, VERIFIED, CLOSED.
- **Severity**, **Priority**: Attribute, welche die Auswirkungen eines Softwarefehlers und dessen Priorität beschreiben

[MozillaOrg02][131] beschreibt weitere Attribute eines Bugzilla-Eintrags:

- **Component**: Problemdomäne des Eintrags, respektive Softwarekomponente, der er zugeordnet werden kann. Das Attribut "Component" steht in direktem Zusammenhang mit einer klar abgegrenzten Menge von Quellcode-Modulen: "Most code modules in Mozilla have one or more associated components in the Bugzilla bug tracking tool." [ReMa02]
- **Status Whiteboard**: Notizen bezüglich des Status-Attributs
- **Keywords**: Schlüsselwörter zur Suche
- **Helpwanted**: Signal zur Kommunikation eines Hilfegesuchs in Bezug auf die Bearbeitung des Bug Reports

131 http://www.mozilla.org/bugs

- **Target Milestone**: Zielsetzung für die Behebung des Fehlers: "If a bug is marked mozilla1.3, it means an assigned developer picked Mozilla 1.3 as his/her estimate of the earliest milestone at which that bug might be resolved. This field should only be set by the person responsible for the bug." [MozillaOrg02][132]
- **Dependency**: Abhängigkeit des Fehlers von anderen Fehlern: "If a bug can't be fixed until another bug is fixed, that's a dependency." [MozillaOrg02][132]. Bugzilla ist in der Lage, Graphen mit allen abhängigen Bug Reports zu generieren.

Die adäquate Verwendung dieser Attribute ermöglicht die effiziente Recherche nach bestimmten Einträgen und die spätere Bearbeitung durch die Entwicklergemeinde. Ein Bug Report dient nicht ausschließlich der Anforderungsdefinition sondern beschreibt und koordiniert über seine Metadaten und vor allem über sein Status-Attribut den darauf basierenden Patchentwicklungsprozess. Daher ist die Pflege der Metadaten der Bug Reports von zentraler Bedeutung in allen Phasen der Patchentwicklung. Dabei ist es nicht allen Entwicklern möglich, jedes beliebige Attribut eines Bug Report selbständig zu modifizieren: "Those to whom additional permissions have been given have the ability to change all the fields of a bug (by default, you can only change a few)." [MozillaOrg02][132]

Zur besseren Verfolgung der Evolution eines Bug Reports werden alle involvierten Akteure über Änderungen automatisch in Kenntnis gesetzt: "Whenever someone makes a change to the bug or adds a comment, the owner, reporter, the CC list and those who voted for the bug are sent an email (unless they have switched it off) showing the changes to the bug report." [MozillaOrg02][132]

#### 4.2.6.2 Enhancement Requests (RFE)

Enhancement Requests repräsentieren Anfragen für eine funktionale Systemerweiterung und werden in Bugzillla genau wie ein Bug Report generiert und verwaltet. Im folgenden wird auf derartige Anforderungsartefakte durch den Term 'RFE' (Request For Enhancement) referenziert.

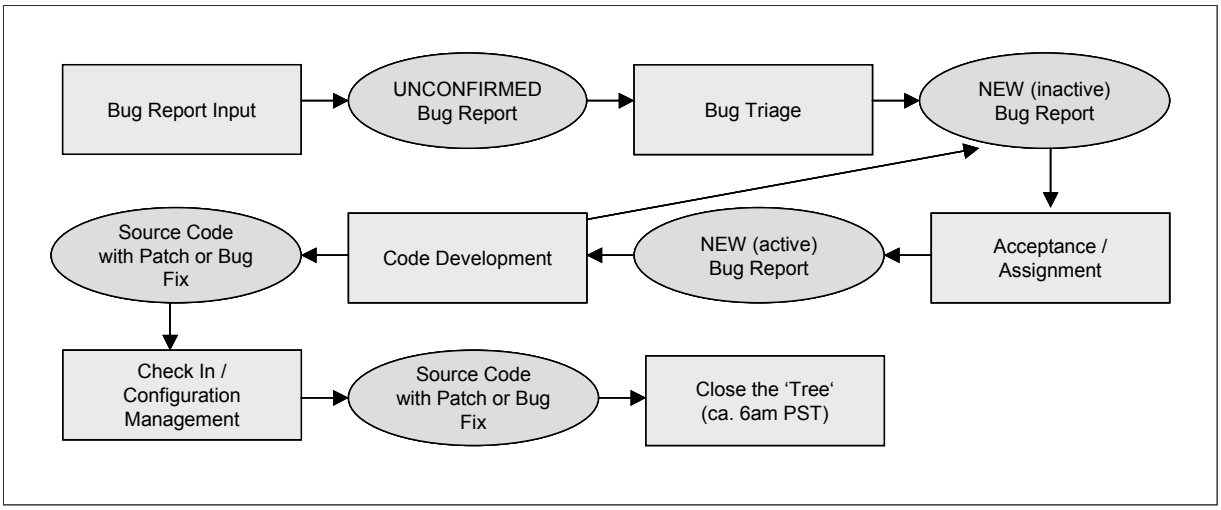
Ein RFE zeichnet sich dadurch aus, dass das Attribut Severity des entsprechenden Bug Report den Wert Enhancement aufweist: "An enhancement request is a bug whose severity is 'enhancement'." [MozillaOrg02][133]

132 http://www.mozilla.org/bugs/
133 http://www.mozilla.org/get-involved.html

### 4.2.6.3 Lebenszyklus eines Bug Reports

Die Ausprägungen des Attributs 'Status' beschreiben den Lebenszyklus eines Bug Report, der in der folgenden Grafik nach [CaLeCh02] dargestellt wird:

Bild 11 Mozilla: Lebenszyklus eines Bug Report (nach [CaLeCh02])




Generell sollten Statusänderungen möglichst durch den Initiator des Bug Report bzw. den Entwickler des Patches ausgeführt und gewünschte Transitionen zuerst in Form von Kommentaren kommuniziert werden: "Be careful when changing the status of someone else's bugs. Instead of making the change yourself, it's usually best to make a note of your proposed change as a comment and to let the bug's owner review this and make the change themselves." [MozillaOrg02][134]

Aufgrund der zentralen Bedeutung dieses Lebenszyklus' und starken Determinierung der Patchentwicklung werden die einzelnen Stati und die in deren Rahmen durchgeführten Prozesse im folgenden kurz dargestellt. Die vorgestellten Prozesse werden aber im Rahmen der Prozessdarstellung noch ausführlich thematisiert.

#### Initialisierung eines "Bug Report"

Jeder Bug Report wird mit seiner Initialisierung als 'NEW' bzw. 'UNCONFIRMED' eingestuft wobei die erstgenannte Ausprägung nur von erfahrenen Entwicklern

134 http://www.mozilla.org/bugs/

vergeben werden sollte. Im Falle der Klassifizierung als UNCONFIRMED wird die Reproduzierbarkeit des Softwarefehlers erst durch die Entwicklergemeinde verifiziert wird, bevor er den Status 'NEW (inactive)' erhält und damit als relevanter, neuer und reproduzierbarer Fehler bestätigt wurde: "New Bugzilla accounts by default create bugs which are UNCONFIRMED - this means that a QA (Quality Assurance) person needs to look at it and confirm it exists before it gets turned into a NEW bug." [MozillaOrg02][135]

### Verifikation des "Bug Report"

Dieser Prozess der Verifikation des Softwarefehlers findet anhand eines 'Bug Triage' genannten Prozesses statt, der zur Folge hat, dass entweder dem Status des Bug Reports der Wert 'NEW' zugewiesen wird, oder aber der Bug Report verworfen wird. Dieser Prozess wird in Kapitel 4.2.8 noch explizit dargestellt.

### Zuweisung der Verantwortlichkeit

Sofern der Entwickler des assoziierten Moduls bzw. der Component Owner den Softwarefehler nicht selbst bearbeitet, wird durch einen dieser beiden Akteure versucht, die Aufgabe der Bearbeitung einem Entwickler zuzuordnen: "A Bugzilla component owner is the default recipient of bugs filed against that component. When someone files a bug, he or she may specify to whom the bug will be assigned. But if no specific assignment is made by the bug reporter, the bug will go to the Bugzilla component owner. [MozillaOrg02][136]. Sobald eine definitive Zuordnung erreicht werden konnte, wird dem Status des Bug Reports der Wert 'New (active)' zugewiesen.

### Bearbeitung und Schließung eines Bug Reports

Nach der erfolgreichen Bearbeitung eines Bug Reports und der damit einhergehenden Generierung eines entsprechenden Patches, wird dem Bugzilla-Eintrag der Status 'Resolved' zugewiesen: "When a bug is fixed it's marked RESOLVED..." [MozillaOrg02][135]. Dies setzt somit die lokale Entwicklung und die Kommunikation des Patches an die Community voraus.

Außerdem wird i.d.R. einer der folgenden Werte als finaler Status vermerkt:

- FIXED: Getestes Patch wurde in den Quellcode integriert
- INVALID: Beschriebenes Problem ist kein relevanter, verifizierbarer Softwarefehler

135 http://www.mozilla.org/bugs/
136 http://www.mozilla.org/about/roles.html

- WONTFIX: Beschriebenes Problem kann nicht durch ein Patch gelöst werden oder repräsentiert einen RFE und keinen Softwarefehler
- DUPLICATE: Beschriebenes Problem wird bereits durch anderen Bug Report bearbeitet
- WORKSFORME: Beschriebenes Problem nicht reproduzierbar

#### Verifikation der Schließung

Die Akteure des Qualitätssicherungsteams, dessen Workflows in Kapitel 4.2.8 "Bug Triage" noch weiter konkretisiert werden, haben das Privileg, Bug Reports, welche bereits durch den Status RESOLVED markiert sind, als VERIFIED zu markieren. Dadurch wird die Bearbeitung des Fehlers vollständig eingestellt und der so markierte Fehlerbericht verbleibt in diesem Status bis das betroffene Modul in einem neuen Release veröffentlicht wird. Anschließend wird dem Status der Wert CLOSED zugewiesen.

#### Erneutes Öffnen

Es ist durchaus auch möglich, dass Softwarefehler erneut geöffnet werden, sofern sich herausstellen sollte, dass ihre bisherige Bearbeitung nur in ungenügendem oder unbefriedigendem Ausmaß erfolgte: "Bugs may come back to life by becoming REOPENED." [MozillaOrg02][137]

### 4.2.7 Erstellung eines Bug Reports

Jeder Anwender hat die Möglichkeit, Anforderungen in Form eines Bug Reports zu kommunizieren. Hierzu ist es zu Beginn notwendig einen Account im Bugzilla-System anzulegen: "A one-time prerequisite process is to open a Bugzilla account." [CaLeCh02]. Weiterhin sollte vor der Erstellung eines entsprechenden Fehlerberichts eine möglichst umfassende Recherche zur Verifikation der Novität des Fehlers erfolgen: "Before submitting a bug report, query Bugzilla to make sure your bug hasn't already been reported." [MozillaOrg02][138]. Hierzu stehen neben dem Bugzilla-System alle sonstigen Informationsquellen, wie die Mozilla-Website, Newsgroups und Mailing Listen und die entsprechenden Archive zur Verfügung.

Häufig geht der eigentlichen Erstellung des Bug Reports auch die Kommunikation der identifizierten Anforderung über die verschiedenen Kommunikationskanäle voraus. Damit wird das Ziel verfolgt, bereits im Vorfeld

137 http://www.mozilla.org/bugs/
138 http://www.mozilla.org/get-involved.html

zu sicherzustellen, dass die Anforderung eine aktuelle und relevante Anforderung darstellt und nicht bereits Entwickler an der Erstellung eines Patches arbeiten, wodurch die Generierung eines diesbezüglichen Bug Reports obsolet und irreführend wäre.

Anschließend werden ein entsprechendes Artefakt im Bugzilla-System erstellt und alle nötigen Metadaten durch den Akteur definiert. Konkrete Vorgaben für die Entwicklung von Bug Reports werden in verschiedenen Dokumentations- und Managementartefakten und primär in den 'Bug-Writing-Guidelines' definiert. Im Kontext von Berichten über Softwarefehler kommt der Reproduzierbarkeit auf der Basis der Informationen des Fehlerberichtes besondere Bedeutung zu.

### 4.2.8 Bug Triage

Der Prozess des 'Bug Triage' ist von immenser Bedeutung für die effiziente Bearbeitung von Bug Reports: "This process is important, highly visible and touted as critical to Mozilla's success as an Open Source Software Development project." [CaLeCh02]

[ReMa02] beschreibt die Aufgaben des Bug Triage wie folgt: "Bug triage involves working on the thousands of bugs filed in Bugzilla, reproducing and assigning problems, invalidating bugs when the described 'bug' is actually intended behavior, or when there is a lack of reproducible steps, and working to better share the burden of fixes among the developers available." Somit ist es ein definiertes Ziel des Bug Triage unbestätigte, respektive noch inaktive Fehlerberichte zu evaluieren und gegebenenfalls zu verifizieren, also die Transition zu einem aktiven Bug Report auszuführen, oder dem Bug Report den Status RESOLVED mit der Kennung 'Invalid' zuzuweisen, sofern der Fehler nicht den Kriterien eines relevanten und reproduzierbaren Softwarefehlers genügt: "Obviously, every bug that is reported does not make it to the NEW (inactive) state because the process of bug triage catches any non-critical bugs that are duplicates or inaccurate." [CaLeCh02]. Außerdem werden relevante und bestätigte Fehlerberichte den entsprechenden Entwicklern zugeordnet und gegebenenfalls das Attribut Component, also die zugehörige Problemdomäne modifiziert und angepasst.

Dieser Prozess stellt sicher, dass interessierte Entwickler stets auf eine konsistente, redundanzfreie Datenbank zugreifen können, die ausschließlich relevante Anforderungen beinhaltet. Dadurch wird gewährleistet, dass die Entwicklungsressourcen zur Ausführung der primären Entwicklungstätigkeiten zur Verfügung stehen: "[…] the process of bug triage saves code developers hours of precious time by offloading the work of actually figuring out if a bug is worthy of attention, narrowing the scope of the reported bugs' domain, and eliminating bugs that have already been fixed." [CaLeCh02]

Prinzipiell kann der Prozess des Bug Triage durch alle Anwender der Software ausgeführt werden, wobei aber primäre Akteure die sogenannten QA-Owner und freiwilligen Softwaretester sind: "QA Owners and Volunteer Testers perform bug triage." [CaLeCh02]. Die spezielle Rolle des QA-Owners wird im Kapitel 4.2.10 noch explizit erläutert werden.

### 4.2.9 Entwicklung von Patches

Jeder individuelle Patchentwicklungsprozess eines Entwicklers basiert auf einem vorher definierten Bug Report, welcher die Anforderungen für das zu entwickelnde Patch definiert. Wie bereits dargestellt wurde, repräsentiert ein Bug Report jede erdenkliche Art von Änderungsvorschlag bezüglich der Software. Da der Entwicklungsprozess im Mozilla-Kontext vollständig auf der Bearbeitung der in Bugzilla verwalteten Änderungsanfragen basiert, kann der Entwicklung eines Patches also entweder ein registrierter Softwarefehler oder eine Anfrage zu einer funktionalen Systemerweiterung zugrunde liegen.

Verschiedene Guidelines definieren diverse Vorgaben für die Entwicklung von Patches so z.B [MozillaOrg02][139]. Außerdem stellt die bereits zitierte 'Development Roadmap' (siehe [Eich03b]) eine wichtige Informationsquelle für Entwickler dar, da hier verschiedene architektonische und grundlegende Informationen über die Zielsetzung und verschiedene, strategisch wichtige Anforderungen an die Softwaresuite definiert wurden.

Basierend auf den Stati des Bug Reports und den Anforderungen, welche sich durch den Review-Prozess im Rahmen der Entwicklung stellen, lassen sich die folgenden Phasen eines Patchentwicklungszyklus unterscheiden:

- Lokale Entwicklung und Kommunikation des Patches
- Review
- Integration des Patches

Diese Teilprozesse werden in den folgenden Abschnitten näher dargestellt.

#### 4.2.9.1 Anforderungen an Patches

Primäre Anforderung für zu entwickelnden Quellcode stellt die Plattformkompatibilität dar, um somit eine möglichst große Anwendergemeinde erreichen zu können: "One of our primary design considerations is that our code work on a wide variety of platforms." [MozillaOrg02][140]. Diese Zielsetzung ist typisch für Open Source - Projekte,

139 http://www.mozilla.org/hacking

deren fundamentale Basis eine möglichst große Anwender- und Entwicklergemeinde darstellt. Weiterhin existiert mit [Williams98] der sogenannte "The C++-Portability Guide", der Leitlinien vorgibt, welche die Generierung von möglichst portablem und plattformunabhängigen C++-Quellcode unterstützen sollen. Eine weitere wichtige Anforderung ist die möglichst flexible Konfigurierbarkeit der Software: "To facilitate the distribution of Mozilla, please take customizability into account so optional features can be added, removed, modified easily." [MozillaOrg02][140]

Zudem spielt die Adaptierbarkeit von Softwaremodulen an die spezifischen Anforderungen verschiedener regionaler Anwenderkreise eine große Rolle. Dies wird im Mozilla-Kontext unter den Begriffen Localization und Customization zusammengefasst. [Cheng01] definiert diese Begriffe dabei wie folgt und spezifiziert außerdem verschiedene konkrete Maßnahmen, um diese Ziele zu unterstützen:

- **Localization**
  Übersetzung und Anpassung der Benutzerschnittstelle an lokale und regionale Kultur und Gewohnheiten
- **Customization**
  Sehr ähnlich zur Localization mit Fokus auf dem User Interface

#### 4.2.9.2 Lokale Entwicklung und Kommunikation des Patches

Die übliche Vorgehensweise zur Generierung eines Patches ist, für den zu modifizierenden Quellcode zwei kongruente Quellcodeverzeichnissbäume einzurichten, wobei die Modifikationen nur in einem der beiden Verzeichnisbäume vorgenommen werden. Dieser Aktivität geht dabei stets der Bezug des aktuellsten Entwicklungsquellcodes aus dem CVS-Archiv voraus, um Konflikte bei der Integration des Patches zu vermeiden. Im Anschluss an die lokale Entwicklung und den lokalen Test des modifizierten Quellcodes kann mit dem Werkzeug `diff` eine Datei erstellt werden, welche lediglich die vorgenommenen Quellcodeänderungen enthält und somit die Menge des zu transportierenden Quellcodes minimiert. Diese Datei enthält den Quellcode als reinen ASCII-Text. Die Extraktion des Patches mit dem diff-Werkzeug stellt eine fundamentale Bedingung für die spätere Integration des Patches in das zentrale CVS-Archiv dar. Zudem gewährleistet dies die Kommunikation des Patches in einem standardisierten Format, wodurch es für alle Entwickler les- und bearbeitbar ist.

Im Anschluss an die Extraktion des Patches kann es durch den Entwickler an die Community kommuniziert werden, indem dazu die etablierten

140 http://www.mozilla.org/hacking

Kommunikationskanäle (Mailing Listen, Newsgroups) instrumentalisiert werden oder das Patch direkt dem zugrundeliegenden Bug Report über dessen Attachement-Attribut zugeordnet wird.

#### 4.2.9.2.1 Format eines Patches

Patches werden in der Regel in Form von Attachments den entsprechenden Bug Reports zugeordnet, zu deren Problem sie eine Lösung zur Verfügung stellen. Basierend auf den Charakteristika des Diff-Werkzeuges, welches als Patch reine ASCII-Text-Dateien erstellt, werden Patches in diesem Format erstellt, wodurch sich zudem große Vorteile für die Lesbarkeit des Quellcodes ergeben.

Weiterhin definiert [MozillaOrg02][141], dass neu erzeugte Quellcodedateien möglichst einen sogenannten 'Boilerplate License Header' enthalten sollen, in welchem die konkrete Software-Lizenz definiert wird, welche für den Quellcode Gültigkeit besitzt: "When creating a new source file, add boilerplate license headers as appropriate, and fill in the blank fields."

Außerdem beinhaltet der Quellcode stets ein Verzeichnis der Namen aller in die Entwicklung des entsprechenden Quellcodeartefakts involvierten Entwickler, die durch ein Patch um den Namen des jeweiligen Entwicklers ergänzt wird. Dies ermöglicht die spätere Rekonstruktion der für verschiedene Modifikationen verantwortlichen Entwickler.

#### 4.2.9.2.2 Aktivitäten der Component Owners

Dieser Abschnitt thematisiert die Aktivitäten von Component Owners im Rahmen der Entwicklungstätigkeiten, die auf einem Bericht über einen Softwarefehler basieren.

"Each Bugzilla component has an owner, which is the default assignee for new issues reported, and a Quality Assurance contact, which oversees the progress of these issues and can intervene as needed to help the developer [...]" [ReMa02]

Primäre Ansprechpartner für die Bearbeitung von Bug Reports sind die Component Owner der mit dem Bug Report assoziierten Components, die auch automatisch von Bugzilla über Modifikation existierender oder das Einstellen neuer Fehlerberichte unterrichtet werden: "Component owners are

141 http://www.mozilla.org/hacking

notified of bugs filed in their responsibility, and schedule them according to their priority and severity." [ReMa02]. Es ist nicht unüblich, dass Component Owner und Modul Owner identisch sind: "In some cases the Bugzilla component owner and the related module owner may be the same person. But in many cases they will be different." [MozillaOrg02][142]. Die Aufgaben der Component Owner zur Sicherstellung der rechtzeitigen Bearbeitung durch die probaten Entwickler setzen sich aus dem regelmäßigen Review der Bug Reports, deren Zuordnung zu den richtigen Components, der Sicherstellung der Existenz von geeigneten Test Cases für die entsprechende Komponente und aus allgemeinem Management der mit der eigenen Komponente assoziierten Fehlerberichte zusammen: "Component owners are expected to review bug reports regularly, reassign bugs to correct owners, ensure test cases exist, track the progress toward resolving important fixes, and otherwise manage the bugs in the component." [MozillaOrg02][142]

Wichtigstes Identifikationskriterium für die dezentrale Bearbeitung von Bug Reports stellt dabei die eindeutige Bug-Nummer zu Referenz auf einen expliziten Fehlerbericht dar: "Bug numbers are used, in this manner, as a code between developers, and they are exchanged freely and continuously through email, chat and Web sites." [ReMa02]. Weiterhin dienen die Einträge im Comment-Attribut eines Bugzilla-Eintrages der Kommunikation von Entwicklungsrelevanten und koordinativen Informationen: "Bugzilla comments are usually the means of communication used to this end. This mechanism occasionally causes dispute, as virtually anyone has freedom to contest a decision made." [ReMa02]

Neben der direkten Verantwortlichkeit der Component Owners steht es jedem Entwickler frei, Fehlerberichte selbständig zu bearbeiten, wobei er dies aber üblicherweise dem entsprechenden Component Owner mitteilt: "If an issue interests another developer, he is allowed to assign the task to himself and propose fixes. He will usually coordinate with the former owner to avoid duplicated or mismatched work." Dadurch wird eine Delegation von Tasks von den Component Owners zu anderen Entwicklern ermöglicht. Zudem kann jederzeit über das helpwanted-Attribut um Unterstützung aus der Entwicklergemeinde gebeten werden: "At other times, if the engineer is overburdened or unable to make the change, a 'help wanted' marker is placed upon the bug report to make it clear that volunteers are needed." [ReMa02]

Die Entscheidungsgewalt liegt auf dieser Stufe des Entwicklungsprozesses aber stets bei dem Component Owner: "The component owner is responsible for the module, and is considered an authority with regard to selecting and implementing changes. If he considers a contributed change relevant and well-

142 http://www.mozilla.org/about/roles.html

implemented, he will allow it; otherwise, he can provide advice on how it should be improved, or ultimately ignore or veto the change." [ReMa02]

#### 4.2.9.2.3 Bearbeitung von Enhancement Requests

In Abgrenzung zur Bearbeitung von klassischen Bug Reports, also Berichten über Softwarefehler, lässt sich die Bearbeitung von Enhancement Requests explizit betrachten. Die Enhancement Requests bzw. RFEs (Requests For Enhancements) stellen eine spezielle Ausprägung von Bug Reports dar, deren Ziel nicht die Behebung von Softwarefehlern, sondern das Hinzufügen von funktionalen Systemerweiterungen oder neuen Systemeigenschaften darstellt. Ein RFE wird wie ein gewöhnlicher Fehlerbericht in Bugzilla verwaltet und in Form eines Patches bearbeitet. Analog zu gemeinen Fehlerberichten stellen auch hierfür die Component Owners die primären Ansprechpartner dar. Da aber ein RFE i.d.R. nicht direkt einer konkreten Component zugeordnet werden kann, da die entsprechenden Initiatoren der Anfrage nicht über die detaillierte Kenntnis der verschiedenen Components verfügen, ist für diesen Prozess vor allem die Eigeninitiative der Entwicklergemeinde von Bedeutung. Diese Aufforderung wird auch auf der Mozilla-Website kommuniziert: "Browse through Bugzilla's enhancement requests and see if there's anything you'd like to implement." [MozillaOrg02][143]. Sobald ein Entwickler entscheidet, die Bearbeitung eines RFE zu übernehmen, sollte er den entsprechenden Bugzilla-Eintrag seiner Person zuordnen, indem er dies über einen Eintrag in dem Comment-Attribut des Bug Reports zum Ausdruck bringt und sein Vorhaben über eine adäquate Newsgroup kommuniziert: "If you decide to work on one, assign the bug to yourself and add a comment saying so. Post to a newsgroup explaining what you plan to do." [MozillaOrg02][143]. Sobald ein geeignetes Patch existiert, wird dieses wie auch ein gewöhnliches, auf der Bearbeitung eines Softwarefehlers basierenden Patch bearbeitet und an die Gemeinde kommuniziert.

#### 4.2.9.3 Review

Die Review-Aktivitäten werden durch alle Adressaten des Patches und durch die sogenannten Modul Owners des betroffenen Quellcode-Artefakts durchgeführt: "We're big believers in code review, so before code is checked in to the CVS Repository it must be reviewed by the appropriate module owner or one of his peers." [MozillaOrg02][144]. Hierbei muss unterschieden werden zwischen Modulen, die den Prozess des sogenannten 'Super Review' erfordern

143 http://www.mozilla.org/get-involved.html
144 http://www.mozilla.org/hacking/

und Quellcode, der anderen, individuell durch die Modul Owner und alle involvierten Entwickler ausgeführten Reviewprozessen unterliegt: "The Mozilla Code Reviewers document distinguishes areas of code that require this so-called 'super-review' from those hosted on cvs.mozilla.org that have other policies set by module owners." [Eich03b][145]. Prinzipiell wird aber jede Modifikation eines Quellcodemoduls dem Review durch den Modul Owner unterzogen: "All code hosted by mozilla.org requires active ownership, and therefore module owner review of changes before a contributor can type 'cvs commit'." [Eich03a]

Die entsprechenden Component- bzw. Modul Owner repräsentieren die primären Verantwortlichen für den Review der bereitgestellten Patches. I.d.R. besteht eine starke Korrelation zwischen Softwaremodulen und den definierten Components im Bugzilla-System, wobei dies aber keine allumfassende Gültigkeit hat. Außerdem stellen die Initiatoren des Bug Reports eine wichtige Kontaktperson für den Review-Prozess dar: "Ask the bug owner (assignee) explicitly for a review by adding comment (unless that assignee is you)." [MozillaOrg02][144]. Falls der Bug Owner nicht in einer adäquaten Zeit reagiert, besteht für den Entwickler weiterhin die Möglichkeit, andere Personen zu konsultieren, die bereits Reviews bezüglich der betroffenen Komponente ausgeführt haben oder über den IRC-Channel irc://irc.mozilla.org/#mozilla nach relevanten Ansprechpartnern zu recherchieren respektive eine entsprechende Anfrage zu kommunizieren. Außerdem besteht nach [MozillaOrg02][144] die Möglichkeit, über die Mailing Liste contributor-help@mozilla.org.uk eine Beschreibung des Problems zu kommunizieren.

Das Ergebniss eines Review-Prozesses wird im Attribut 'Comments' vermerkt, wobei dies neben dem Ergebnis des Reviews auch Verbesserungsvorschläge, Fragen oder Änderungswünsche enthalten kann: "The actual review consists of bug comments describing changes that should be made to improve the code quality, questions about an unclear section, or recommendations on various other aspects of the patch (performance impact, dependencies, related problems)." [ReMa02]. Im Falle der Akzeptanz des Patches durch den Reviewer, bringt dieser durch einen standardisierten Eintrag in diesem Attribut sein Einverständnis zum Ausdruck: "If no issues remain, the reviewer will mark his seal of approval by adding 'r=(reviewer name)' to a bug comment." [ReMa02]

In manchen Fällen ist ein sogenannter 'Super Review' erforderlich, dessen Teilprozesse in Kapitel 4.2.9.4 noch explizit erläutert werden: "In most cases an additional level of review known as 'super-review' is required." [MozillaOrg02][146]. Nachdem alle Reviewer das Patch bestätigt haben, wird dies

145 http://www.mozilla.org/roadmap/roadmap-05-Jun-2002.html
146 http://www.mozilla.org/hacking/

eindeutig über die entsprechenden Metadaten des Bug Reports kommuniziert: "Again, the super-reviewer may request changes to your patch, and when they are happy, they will say so in the bug." [MozillaOrg02][146]

Sofern ein Release einer neuen Version der Software bevorsteht, fokussiert der Reviewprozess auf die geplanten Änderungen des neuen Release und versucht diese zu kategorisieren. Patches werden entsprechend dem mit ihnen verbundenen Risiko für die Stabilität und Performance der Software priorisiert:

"Every time a Mozilla.org milestone releases, the review process focuses on the importance of implementing a particular set of fixes that have been targeted for that milestone. These are prioritized based on risk to stability and performance presented by that particular patch." [CaLeCh02]

Modul Owner die selbständig das mit Ihnen assoziierte Modul modifiziert haben, somit also sowohl den Entwickler als auch den Reviewer in Personalunion repräsentieren, lassen das Patch durch weitere Personen evaluieren: "Module owners should have a peer review their patches, and not do self-review and make a wish." [EiBa03]. Weiterhin definiert [EiBa03] verschiedene Vorgaben und Empfehlungen an die Entwickler von Patches, um den Prozess des Review zu beschleunigen und seine Ausführung zu ermöglichen.

#### 4.2.9.4 Super Review

Neben den bereits dargestellten, relativ informellen Review-Prozessen, wird in einigen Fällen der sogenannte 'Super Review' praktiziert: "Checking into most (but not all) of the Mozilla tree also requires an additional level of pre-check-in code review." [EiBa03]. Der Prozess des Super Review ist für einige Quellcodemodule erforderlich und erfolgt i.d.R. nach dem Code Review durch den Modul Owner.

Die mit dem Super Review verbundene Zielsetzung ist die gesamtheitliche Qualitätssicherung des Softwareentwurfs und des Quellcodes und eine hohe Kompatibilität zu existierenden Standards und definierten, projektweiten Richtlinien: "The goals of super-review are to increase code quality, promote best practices, and reduce regressions." [EiBa03]. Diese Ziele sollen durch die modulübergreifende Evaluierung des Quellcodes und des Softwaredesigns, der Abhängigkeiten und der Softwarequalität im Allgemeinen erreicht werden: "This level of review will look at the quality of the code itself, its potential effects on other areas of the tree, its use of interfaces, and otherwise its adherance to Mozilla coding guidelines." [EiBa03]

Der Prozess des Super Review darf nur durch eine speziell privilegierte Gruppe von Entwicklern ausgeführt werden, welche die Modifikationen am Quellcode

aus einer gesamtheitlichen Perspektive bezüglich einer klar definierten Zielsetzung untersuchen: "The super-reviewers are a designated group of strong hackers who review code for its effects on the overall state of the tree, use of interfaces, overall quality, API and XPCOM use, and adherence to Mozilla coding guidelines." [MozillaOrg02][147]. Dieser spezielle Personenkreis ist über eine dedizierte Mailing-Liste bzw. Newsgroup erreichbar: "There is a reviewers@mozilla.org mailing list, which has a newsgroup mirror, with the usual, canonically named, mail gateway (mailto:mozilla-reviewers@mozilla.org)." [EiBa03]. Diese Kommunikationskanäle dienen i.d.R. der Diskussion von Patches und der Transparenz des Reviewprozesses für interessierte Entwickler und Reviewer. Positives Feedback auf ein Patch, also die Zustimmung durch den Super Reviewer wird analog zum gewöhnlichen Reviewprozess durch das Hinzufügen eines Eintrages in die Liste der Comments des entsprechenden Bug Reports zum Ausdruck gebracht: "A senior developer ... must mark approval for integration to be allowed (the mark used is sr=(name))." [ReMa02]

Die Super Reviewer müssen zur Ausführung ihrer Aufgaben nicht gezwungenermaßen über spezielle Kenntnisse der jeweiligen Problem-Domäne des zu evaluierenden Softwaremoduls verfügen, da sie lediglich die gesamtheitlichen Aspekte betrachten. Welche Module den Super Review-Prozess absolvieren müssen, wird von der Mozilla Organisation definiert: "The current rule is: code that is 'in-process' or built with the Mozilla browser/mail/news/editor application suite requires super-review." [EiBa03]. Weiterhin empfiehlt [EiBa03] den Super Review Prozess für eine Vielzahl weiterer Module und definiert verschiedene Ausnahmefälle, in denen spezielle Zuständigkeiten für den Super Review existieren.

#### 4.2.9.5 Integration von Patches

Den erfolgreich durchgeführten Patch-Review-Prozessen schließen sich die Aktivitäten zur Integration des Patches in den zentralen Entwicklungsquellcode an: "When you have review and super-review, and your patch is as good as it can be, it can be checked into the tree." [MozillaOrg02][148]. Diese Integration kann nur durch entsprechend privilegierte Akteure erfolgen. Falls der Entwickler nicht über die entsprechenden Privilegien verfügt, das Patch direkt in den CVS-Verzeichnisbaum zu integrieren, muss er eine entsprechend befugte Person identifizieren. Falls der Reviewer dazu nicht in der Lage sein sollte, gilt es als typische Vorgehensweise, wieder über die einschlägigen Kommunikationskanäle um Unterstützung zu bitten. Möglicherweise ist der

147 http://www.mozilla.org/about/roles.html
148 http://www.mozilla.org/hacking/

Verzeichnisbaum aus Gründen eines geplanten Releases auch für das Hinzufügen von Patches gesperrt, worauf der Abschnitt bezüglich des Release-Prozesses noch explizit eingeht. In diesem Falle ist in der Regel noch ein separater Approval-Prozess notwendig: "Occasionally, the tree is locked down a little tighter approaching a release, and a third level, called 'approval', is required before you can check in." [MozillaOrg02][148]. Dieser Zustand des CVS-Tree wird in dem sogenannten 'Freeze'-Prozess zusammengefasst und über das Tinderbox-Werkzeug kommuniziert. Für den Approval-Prozess ist es in diesem Kontext notwendig, die sogenannten 'Drivers' zu kontaktieren, die entscheiden, ob das Patch noch in das neue Release oder erst in ein späteres integriert werden kann: "To get approval, mail drivers@mozilla.org. If approval is refused, you may have to wait a week to check in your patch, until the release branch is cut." [MozillaOrg02][148]

Der Prozess der Integration von Patches ist somit als zweistufiger Integrationsprozess rezipierbar. Auf der ersten Stufe erfolgt eine Integration in den CVS-Verzeichnisbaum (Commit) durch eine entsprechend autorisierte Person, die häufig durch den Modul Owner repräsentiert wird. In der zweiten Phase werden so aufgenommene Modifikationen an der Software in eine neu zu veröffentlichende Version der Software integriert, wobei dies nur durch die Mozilla Organisation bzw. deren Mitglieder erfolgen kann.

#### 4.2.9.5.1 Integration in das CVS-Repository (Commit)

Entsprechend legitimierte Entwickler können Patches selbständig in den Quellcode integrieren: "If you're a frequent contributor or if you're contributing your own module you may be given write access to our CVS repository so you can commit the changes yourself. [MozillaOrg02][149]

Bei einer selbständigen Integration durch einen Entwickler reglementieren die in [MozillaOrg02][150] definierten Vorgaben die Aktivitäten im Rahmen dieses Teilprozesses. Diese schreiben z.B. vor, dass vor der Integration Tests auf verschiedenen Plattformen ausgeführt werden müssen und der Entwickler nach der Integration für die Implementierung von Änderungen zur Verfügung stehen sollte. Des weiteren muss er eigenständig recherchieren, ob das CVS-Archiv kurz vor einer temporären Schließung steht: "As someone checking in code to the tree, you must use bonsai to find out news about the tree, such as whether it's open and whether there are any planned closures." [MozillaOrg02][150]

149 http://www.mozilla.org/source.html
150 http://www.mozilla.org/hacking

Außerdem ist für das Hinzufügen von Verzeichnissen zu CVS die Legitimation durch den Owner des übergeordneten Verzeichnisses einzuholen bzw. direkt die Mozilla Organisation zu konsultieren: "When creating a new directory, get the permission from the owner of the containing directory. To add a new directory to the 'mozilla' top-level directory, contact the mozilla.org staff." [MozillaOrg02][150]

#### 4.2.9.5.1.1 Schreibzugriff auf CVS

Regelmäßig beitragende Entwickler können die Zuweisung von Commit-Rechten bezüglich des gesamten Quellcode-Archivs oder einzelner Quellcode-Segmente anfordern: "When you've been contributing patches frequently and have an established track record for good quality code, you can start the process of getting CVS Commit access." [MozillaOrg02][150]

[MozillaOrg02][151] beschreibt detailliert den Bugzilla-basierten Teilprozess, durch den ein Entwickler die notwendigen Privilegien zur selbständigen Integration von Patches erhalten kann:

1. Erstellung eines Bug Reports mit den folgenden Metadaten:
   Product: mozilla.org
   Component: CVS Account Request
2. Hinzufügen eines Kommentars durch etwaige Fürsprecher, die bereits über das Commit-Privileg verfügen
3. Begründung, falls keine Einwilligung durch einen Super-Reviewer notwendig ist
4. Bitte an die Super Reviewer, ihr Einverständnis zu erklären, falls dies erforderlich sein sollte.
5. Versendung eines entsprechenden Fax-Formulars und Aktualisierung des Bug Reports. Durch das Unterschreiben des Formulars erklärt sich der Entwickler mit den gesetzlichen und prozeduralen Bestimmungen der Mozilla Organisation einverstanden: "Signing the form indicates you understand and agree to our legal requirements."
6. Bestätigung des Formularempfangs durch ein Mitglied der Mozilla Organisation
7. Aktualisierung des Bug Reports durch den Entwickler, sobald alle nötigen Informationen existieren
8. Automatische Weiterleitung des Bug Reports durch Bugzilla an das zuständige Mitglied der Mozilla Organisation

151 http://www.mozilla.org/hacking/getting-cvs-write-access.html

9. Generierung des Accounts und Aktualisierung des Bug Reports durch das zuständige Mitglied der Mozilla Organisation

#### 4.2.9.5.1.2 Anforderungen an Committer

Anforderungen an entsprechend privilegierte Entwickler werden in [MozillaOrg02][151] wie folgt dargestellt und anhand von Peer- und Code Review durch die Mozilla Organisation verifiziert: "You'll need to demonstrate competence with the code you're working with, other Mozilla code you might affect with your work, mozilla.org check-in, build and related processes (like watching the builds when you're on the hook) and basic good coding practices."

[MozillaOrg02][151] definiert weiterhin die folgenden unabdinglichen Eigenschaften des von einem Committer entwickelten Quellcodes:

- Kompatibilität der Mozilla Architektur
- Probate Verwendung der verschiedenen Code-Bibliotheken
- Modularität und Erweiterbarkeit
- Minimale Abhängigkeiten
- Kompatibilität zu Standards
- Konformität mit den Mozilla-Code-Konventionen
- Konsistente Formatierungen

#### 4.2.9.5.1.3 Integration durch Modul Owner

Falls der Entwickler nicht über die gestellten Anforderungen zur selbständigen Quellcodeintegration verfügen sollte, ist er auf die Integration durch den Modul Owner bzw. andere, autorisierte Akteure angewiesen: "To contribute code, submit your changes to the owner(s) of the module(s) you plan to change. If they approve the code they'll commit your changes." [MozillaOrg02][152]

Ein Modul Owner hat dabei die fachliche Verfügungsgewalt über den ihm zugeordneten Quellcode: "A module owner is someone to whom mozilla.org staff delegates leadership of the development of a module of code. This includes a range of responsibilities relevant to the daily management of the module, including approving patches as ready to be checked into the module." [MozillaOrg02][153]. Sofern ein Modul Owner eine Modifikation positiv bewertet, integriert er sie in das entsprechende CVS-Verzeichnis.

152 http://www.mozilla.org/source.html
153 http://www.mozilla.org/about/roles.html

#### 4.2.9.5.2 Integration in die Quellcode-Distribution

Nach der Integration in das CVS-Archiv kann die Integration der Änderungen in eine Distribution des Quellcodes durch die Mozilla Organisation erfolgen: "Changes to a module are made by that module's owner, and then sent to mozilla.org for inclusion in our source distributions." [MozillaOrg02][154]. Dies wird in enger Zusammenarbeit zwischen den Modul Owners und der Mozilla Organisation durchgeführt und durch die Publikation im Rahmen des Release Prozess abgeschlossen, welcher an probater Stelle detailliert dargestellt wird.

### 4.2.10 Quality Assurance - Testprozess als Entwicklungsprozess

Der Aspekt der Qualitätssicherung findet im Kontext des Mozilla-Projektes explizite Beachtung und wird daher in diesem Abschnitt separat dargestellt: "Though many OSS projects do in fact include in their process Quality Assurance (QA) procedures, the Mozilla Project was also a pioneer in the use of the term itself to describe an explicit process." [ReMa02]. Die bisher genannten Review- und Testprozesse dienen zwar auch der Qualitätssicherung, werden aber nicht durch ein dediziertes QA-Team ausgeführt bzw. gesteuert.

Im folgenden werden die Entwicklungsprozesse aus der Perspektive der ganzheitlichen Qualitätssicherung dargestellt und die einzelnen Teilprozesse im Kontext der Prozesse der Quality Assurance betrachtet. Die Qualitätssicherung im Mozilla-Kontext ist ein Hybrid der Formalismen, die in der klassischen bzw. proprietären Systementwicklung zur Etablierung von qualitätssichernden Aspekten angewendet werden und der demokratischen und informellen Prozesse, die den inhärenten Spezifika der OSS-Entwicklung Rechnung tragen: "The QA process in the Mozilla project is a mix between the classical software engineering view and the process that happens to adapt to the open source software environment." [CaLeCh02]

Die Hauptaufgabe des QA-Teams besteht aus der Durchführung von iterativ ausgeführten und formalisierten Softwaretests mit dem Ziel, Bug Reports in Bugzilla zu generieren und damit einer weiteren Bearbeitung zuzuführen: "The main activities QA is responsible for are testing and bug triage. There are many testing activities in Mozilla: ad-hoc volunteer testing, smoketests and even contributed functional tests, which are run by Netscape client QA." [ReMa02]. Damit stellt die explizite Formulierung dieses qualitätssichernden Elements im Rahmen der Mozilla-Entwicklung nur die Betonung eines ohnehin existenten Aspekts des Entwicklungsprozesses dar. Im Rahmen dieses Kapitels sollen daher auch nur die Prozesse thematisiert werden, die abseits der üblichen Test-

154 http://www.mozilla.org/mission.html

Szenarien durchgeführt und durch die Mozilla Organisation explizit automatisiert und formalisiert wurden.

#### 4.2.10.1 Dedizierte QA-Kommunikationskanäle

Der Hauptanteil der Kommunikation wird über dedizierte Newsgroups durchgeführt: "In addition to the development newsgroups, several unmoderated QA newsgroups exist to foster communication between Mozilla QA members." [MozillaOrg02][155]

[MozillaOrg02][156] bezeichnet die verschiedenen dedizierten Newsgroups, die explizit für die Akteure der Quality Assurance zur Verfügung gestellt wurden.

- netscape.public.mozilla.qa.general
- netscape.public.mozilla.qa.i18n

Weiterhin existieren nach [MozillaOrg02][156] die folgenden IRC-Kanäle bei irc.mozilla.org, für die Kommunikation im Rahmen der Qualitätssicherung:

- #qa
- #mozillazine
- #smoketest

#### 4.2.10.2 Aktive Mitwirkung am Prozess der QA

Die Akteure im Rahmen der Qualitätssicherung setzen sich aus verschiedenen dedizierten Rollen und allen freiwillig Akteuren zusammen, die Review- oder Testaufgaben ausführen: "By inviting volunteers to perform QA tasks, the Mozilla Project has effectively harnessed the attention of a very large and dedicated population of testers, who report bugs and follow up on reproducibility and resolution issues." [ReMa02]

Generell führen fast alle Anwender und Mitglieder der Entwicklergemeinde implizit oder explizit Tätigkeiten aus, die zur Qualitätssicherung im Mozilla-Projekt beitragen: "The fact the user base is a combined beta test community and QA assistance group is a fundamental feature in the Mozilla quality process." [ReMa02]. Nach [MozillaOrg02][156] existieren verschiedene Optionen für jeden Akteur der Community, um explizit am Prozess der Qualitätssicherung

155 http://www.mozilla.org/quality/
156 http://www.mozilla.org/quality/help/

teilzuhaben. Diese Handlungsalternativen lassen sich in die folgenden Kategorien einteilen:

1. Bug Triage
2. Aktivitäten im Prozess der Nightly Builds
3. Verbesserung der Performance
4. Generierung von Test Cases

#### 4.2.10.3 Bug Triage und QA

Der Prozess des Bug Triage wurde bereits an anderer Stelle thematisiert und dargestellt (Kapitel 4.2.8). Im Rahmen der Quality Assurance stellt diese Aktivität einen elementaren Bestandteil zur effizienten Bearbeitung von Bug Reports dar. Zudem existieren verschiedene Bug Reports, die explizit für die Akteure der QA von Relevanz sind und durch das Keyword-Attribut qawanted markiert sind. Diese Fehlerberichte sollten durch Mitglieder der Qualitätssicherungsteams bearbeitet werden. Diese Anforderungsartefakte müssen im Prozess des Bug Triage selektiert und den QA-Akteuren zugeordnet werden.

#### 4.2.10.4 Aktivitäten im Prozess der Nightly Builds

Im Rahmen der Nightly Builds, welche im Kapitel 4.2.10.6 noch genauer erläutert werden, können sich Anwender durch die Durchführung von Tests in den Prozess der QA einbringen. Diese Tests können unterschieden werden in

- Smoke Tests
- SSL-Tests
- Allgemeine Tests

Die verschiedenen Kategorien von Softwaretests werden im folgenden noch weiter konkretisiert.

#### 4.2.10.5 Verbesserung der Performance

Die sehr anspruchsvolle Aufgabe des Ausführens von Performance-bezogenen Tests und der Implementation entsprechender Änderungen wird nur durch sehr versierte Entwickler wahrgenommen und wird aufgrund seiner hohen Anforderungen und der damit einhergehenden peripheren Bedeutung für den gesamtheitlichen Entwicklungsprozess im Rahmen dieser Arbeit nicht weiter konkretisiert.

#### 4.2.10.6 Testprozess der ‘Nightly Builds’

In diesem Abschnitt werden der Prozess der 'Nightly Builds' und die mit ihm assozierten Teilprozesse dargestellt. Der in diesem Kapitel dargestellte Prozess repräsentiert eine spezielle Ausprägung eines Releaseprozesses und umfasst auch die Testprozesse, welche auf diesem speziellen Softwareartefakt, dem 'Nightly Build' basieren. Die eigentliche Durchführung des Release Prozesses der Nightly Builds stellt eine Management Aufgabe dar, welche im entsprechenden Abschnitt (siehe Kapitel 4.3.2) explizit erläutert wird.

##### 4.2.10.6.1 Nightly Builds

Die formalisierten Testprozesse im Kontext spezieller Software-Artefakte stellen einen sehr spezifischen und für die Mozilla-Entwicklung wichtigen Aspekt dar: "Nightly builds are a fundamental aspect of the Mozilla project." [CaLeCh02]. Nightly Builds werden durchgeführt, indem täglich kompilierte Binärversionen des Quellcodes erstellt werden und diese über die Website veröffentlicht werden. Anwender können diese Artefakte dann Tests unterziehen und entsprechende Fehler über Bugzilla kommunizieren: "As their name indicates, they occur nightly and are then made available for users who can report bugs back to the Mozilla organization using Bugzilla bug tracking tool or who can be volunteers joining Mozilla community." [CaLeCh02]. Die Artefakte der Nightly Builds unterliegen nicht den Ansprüchen von Stabilität und hoher Produktreife und werden zur Realisierung dieser Ansprüche instrumentalisiert: "The nightly build product is not really mature when it is released, but is released for the purpose of eventually reaching maturity." [CaLeCh02]

Diese Vorgehensweise, respektive die Entwicklung und Veröffentlichung von kompilierten Test-Releases basierend auf dem aktuellsten Entwicklungsquellcode, hat den Vorteil, dass integrierte, aber nicht publizierte Änderungen am Quellcode einer breiten Masse zugänglich gemacht werden können und somit eine größere Anzahl von Testern in den Prozess integriert werden können: "The advantage of this is that more people from around the world are able to get involved in the testing of the product so that more bugs can be caught, reviewed, and fixed." [CaLeCh02]

##### 4.2.10.6.2 Test der Builds

Die Akteure des Test-Prozesses können die Test-Artefakte nach der Durchführung des Erstellungs- und Releaseprozesses durch das Project Management von der Website des Projekts herunterladen.

In einem weiteren Schritt geben die Tester ihr Testvorhaben über eine IRC-Nachricht bekannt. Der Testprozess umfasst verschiedene Kategorien von

Softwaretests, die nachfolgend näher erläutert werden: "There are three main types of tests that can be run in the testing process; smoketests, general test cases and SSL tests. Inputs to these processes are predefined test cases that are produced by Mozilla QA teams and volunteer testers." [CaLeCh02]. Weiterhin kann eine Liste der sogenannten 'Smoke Test Blockers' eingesehen werden, welche bereits bekannte Test-Probleme definiert und in Bugzilla verwaltet wird.

Falls die Tester Fehler entdecken, recherchieren sie in Bugzilla, der Smoketest-Blocker-Liste oder anderen gebräuchlichen Informationsquellen, ob dieser Fehler bereits kommuniziert wurde und fügen dem entsprechenden Artefakt gegebenenfalls weitere Informationen hinzu. Andernfalls wird der Softwarefehler anhand einer IRC-Nachricht kommuniziert: "If research provides no further insight into whether the bug is known or not, the QA participant uses IRC Messages to exchange bug information with anyone that is involved with that day's build and testing." [CaLeCh02]

#### 4.2.10.6.3 Smoke Tests

Der Prozess der Smoke Tests ist ein elementarer Bestandteil des Prozesses der Erstellung der Nightly Builds: "Smoke testing in Mozilla is an absolutely integral QA activity that is performed every day." [CaLeCh02]. Die Mozilla Smoke Tests sind ein weiteres Distinktionsmerkmal, durch das sich das Mozilla-Projekt sowohl von proprietären als auch von vielen OSS-Projekten unterscheidet. Sie repräsentieren eine minimale Menge von formal durch die Mozilla Organisation spezifizierten Test Cases, durch die ein Build in Mozilla seine Korrektheit auf den wichtigsten Plattformen beweisen muss: "The Mozilla QA Smoke Tests are a minimal set of testcases which are required to pass/work in order to bless a Mozilla build as stable and permit developers to continue work. The code must pass on Win32, Mac and Linux." [MozillaOrg02][157]. Smoke Test Cases stellen keine vollständigen, funktionalen Tests dar, aber gehen im Mozilla-Kontext weit über die gemeinhin verwendete Bedeutung des Terms hinaus: "Asa Dotzler, a mozilla.org staff member in charge of QA, explains that Mozilla smoketest plans are more thorough than what is generally considered a 'smoketest', but not as complete as full functional tests." [ReMa02]. Die Smoke Tests werden zwar nur für die gebräuchlichsten Plattformen (Windows, Linux, Mac OS) ausgeführt, wobei aber natürlich vor einem offiziellen Software Release einer neuen Mozilla Version auch Tests auf anderen Plattformen durchgeführt werden.

Die Smoke Tests werden durch interessierte Anwender ausgeführt, wobei für die Koordination und Kommunikation ein dedizierter IRC-Kanal zur Verfügung

157 http://www.mozilla.org/quality/smoketests/faq.html

steht: "Smoke tests are performed by volunteers who join the #smoketest channel in IRC and volunteer to perform the predefined smoke tests in one of the many Quality Assurance product areas." [CaLeCh02]. Die Smoke Tests werden nach [CaLeCh02] in neun verschiedene, funktional getrennte Bereiche eingeteilt: "Installer, Profile Manager, Browser, Editor, Composer, Mail, Internationalization Browser, Internationalization Composer, Internationalization Mail and Embedding." Eine Liste der verschiedenen, formal definierten Smoke-Tests mit den konkreten Tätigkeitsbeschreibungen zu deren Ausführung ist auf [MozillaOrg02][158] zu finden. Die dort beschriebenen Smoke Tests testen verschiedene Standardfunktionalitäten in möglichst allen Softwaremodulen der Browser-Suite und stellen die Rahmenbedingung dar, um ein Build als stabil einzustufen.

Wie bereits in der Prozessbeschreibung der Nightly Builds erwähnt, werden die gefundenen Fehler entweder direkt behoben, oder, falls sie als unkritisch für das Gesamtsystem eingestuft werden, als Bug Report in Bugzilla vermerkt. Im Rahmen der Erstellung eines Smoke-Test-basierten Bug Reports werden nach [CaLeCh02] verschiedene Metadaten des Bug Reports mit spezifischen Werten besetzt:

- Severity: blocker
- Keyword: smoketest

Falls ein kritischer Fehler entdeckt wird, wird zuerst versucht, den Autor des Quellcodes zu kontaktieren, damit dieser den Fehler beheben oder die Änderungen rückgängig machen kann: "If the tests fail, the people responsible for the offending changes are contacted and required to either fix or roll back their changes." [ReMa02]. Die Kommunikation erfolgt wie im restlichen Projekt auch über die bereits erwähnten Kommunikationswerkzeuge: "All tasks are greatly coordinated through web tools such as IRC, Newsgroup, and Bugzilla." [CaLeCh02]

Zielsetzung der Smoke Tests ist es, weitere Entwicklungen am Quellcode auf einem möglichst stabilen, korrekten und langfristig einsetzbaren Quellcode-Fundament basieren zu lassen: "This allows development to continue on a code base that is guaranteed to not include major regressions." [ReMa02]

#### 4.2.10.6.4 Allgemeine Testfälle

Allgemeine Testfälle werden nicht täglich ausgeführt und sind somit nicht integraler Bestandteil jeder Iteration der Nightly Builds: "However, QA volunteers and QA engineers are encouraged to run these frequently and to

158 http://www.mozilla.org/quality/smoketests

track the last time each test was successfully executed." [CaLeCh02]. Ebenfalls nach [CaLeCh02] sind diese Test Cases in vier funktional voneinander abgrenzbare Kategorien unterteilbar:

- Browser
- Layout bzw. Gecko
- Mail / News
- Internationalization

Die verfügbaren Test Cases sind auf [MozillaOrg02][159] für jeden Anwender verfügbar. Es existiert kein formal spezifizierter Prozess zur Erstellung Allgemeiner Testfälle, da diese nicht durch die Mozilla Organisation, sondern durch die QA-Owner eigenständig für ihre Problem Domäne erstellt werden: "Since each QA owner is charged with creating test cases for their product area, there is not a standard process for creating or running general test cases. Instead, each QA owner maintains his/her own test cases on the Mozilla.org web portal." [CaLeCh02]. Dadurch bestehen große Differenzen in Quanti- und Qualität der allgemeinen Test Cases.

Die Kategorie der Cases die zum Testen des Layouts verwendet werden stellt eine Besonderheit dar und wird deswegen im folgenden Abschnitt explizit erläutert.

#### 4.2.10.6.5 Layout Test Cases

'Layout Test Cases' definieren bestimmte Testaktivitäten und deren Ablauf zur Identifikation von layoutbezogenen Fehlern. Diese können z.B. auf der Verarbeitung von Kombinationen bestimmter HTML-Tags mit definierten Parametern basieren, die von Mozilla möglichst fehlerfrei interpretiert und dargestellt werden sollten. Diese speziellen Test Cases werden ebenfalls nicht täglich, aber dennoch möglichst häufig ausgeführt:

**Generierung von Layout Test Cases**

Die Layout Test Cases sind ein Spezialfall der allgemeinen Test Cases, da sie sukzessive durch die Anwender und Entwickler erweitert werden, indem verschiedene layoutbezogene Bug Reports zu allgemeinen Test Cases zusammengefasst und verallgemeinert werden. "This type of test case is generally written by volunteers or QA engineers and is called the 'BugAThon' project." [CaLeCh02]. Die layoutbezogenen Test Cases können, im Gegensatz zu anderen, bereits aufgeführten Kategorien von Test Cases, durch jeden

159 http://www.mozilla.org/quality/help/index-to-tests.html

Akteur definiert werden. Der Terminus BugAThon beschreibt im Mozilla Kontext den Aufruf an die Anwender, Test Cases bezüglich der Layout Engine Gecko zu definieren: "… without help, the number of bugs submitted in the core areas of Gecko (Mozilla's Web display engine), such as the DOM, CSS, Layout etc. would overwhelm the developers." [MozillaOrg02][160]

Ein Layout Test Case repräsentiert die kleinstmögliche Kombination von HTML-Tags und assoziierten Parametern, die es ermöglicht, einen definierten Layout Bug zu reproduzieren: "A simplified test case means the simplest possible web page that still reproduces the bug." [MozillaOrg02][160]. Layout Testfälle werden i.d.R. erstellt, indem ein Anwender Bugzilla nach layout-relevanten Fehlerberichten durchsucht und deckungsgleiche Instanzen zu einem allgemeinen Test Case zusammenfasst: "Decomposing layout bugs into test cases is very helpful to engineers - it saves them loads of time." [MozillaOrg02][161]. Dadurch wird der Aufwand zur Fehlerbehebung für die Entwickler stark minimiert.

Die durch den Anwender ausgeführte Teilprozesse beschreibt [MozillaOrg02][160] wie folgt:

1. Öffnen eines Bugzilla Accounts um nach relevanten Bug Reports recherchieren zu können
2. Download der neuesten Builds (Milestone oder Nightly Build)
3. Recherche nach Layout-Bug Reports
4. Auswahl eines spezifischen Bugs und Markierung des/ der Bug Report(s) und Dokumentation des Vorhabens in den Comments des Bug Reports inkl. Angabe der E-Mail Adresse
5. Download der definierten Webseiten, die den Fehler reproduzieren inkl. aller verwendeten Scripts, CSS-Dateien und Grafiken
6. Löschen aller nicht relevanten Tags, Scripts und Elemente, bis nur noch die den Fehler verursachenden Informationen enthalten sind "The task of simplifying a test case involves taking the downloaded test case and stripping as much text out of the page as one can without removing the bug." [CaLeCh02]
7. Anfügen des Test Cases an den entsprechenden Bug Report (als ASCII-Text bzw. als ZIP-Datei, falls der Test Case mehrere Dateien beinhaltet)
8. Hinzufügen des Schlüsselwortes "testcase" zu den Keyword-Attributen des Bug Reports

160 http://www.mozilla.org/newlayout/bugathon.html
161 http://www.mozilla.org/quality/help/

Das so generierte Artefakt "Layout Test Case" dient als Input-Artefakt sowohl für den Teilprozess des Ausführens der Allgemeinen Testfälle als auch für die Behebung von Softwarefehlern, also der Generierung von Patches, sofern sich dieses Patch auf einen layoutrelevanten Bug Report bezieht: "... the test case artifacts that are attached to the bug report serve as input to both the release cycle's run general test case sub-process ... and the code development (bug fixing) activities... ." [CaLeCh02].

#### 4.2.10.6.6 SSL-Testfälle

SSL-Test Cases dienen der Sicherstellung der Funktionalitäten bezüglich der Datenübertragung über SSL (Secure Socket Layer). Da sie wie Smoke Tests auch täglich im Prozess der Nightly Builds ausgeführt werden, werden sie auch häufig als Security Smoke Tests bezeichnet. SSL Testfälle werden ebenfalls durch die Mozilla Organisation definiert und auf [MozillaOrg02][162] zusammengefasst und beschrieben.

Nach [CaLeCh02] müssen zur Durchführung der SSL-Testcases die folgenden Teilprozesse ausgeführt werden

1. Installation des Builds
2. Test der SSL-Verbindung durch Aufruf von sicheren Websites über HTTPS
3. Änderung der Sicherheits-Optionen von Mozilla
   a. Anwendung des Personal Security Manager User Interface
   b. Import eines persönlichen Zertifikats

Die Aktivitäten zur Ausführung dieser Teilprozesse werden auf [MozillaOrg02][162] detailliert spezifiziert.

### 4.2.11 Dokumentation beitragen

"Open documentation is as important to the success of Mozilla as open source." [MozillaOrg02][163]

Das Erstellen von Dokumentation ist ein wichtiger Beitrag zum Mozilla-Projekt und ist vor allem in Anbetracht der schnellen Quellcode-Evolution von besonderer Bedeutung. Eine Möglichkeit, Dokumentation zur Verfügung zu stellen, ist die direkte Generierung von Kommentaren im Quellcode: "If you

162 http://www.mozilla.org/quality/security/smoketest.html
163 http://www.mozilla.org/docs/contribute.html

write your own code, document it. Much of the existing code isn't very well documented." [MozillaOrg02][164] .

Möglichkeiten, sich direkt in den Prozess der Entwicklung von Dokumentationsartefakten einzubringen, können in Anlehnung an [MozillaOrg02][163] wie folgt beschrieben werden:

- Anwenderdokumentation schreiben und anschließend über die Dokumentations-Newsgroup posten um Feedback und Unterstützung zu erhalten
- Dokumentation von Patches
- Studium der Newsgroups zur Identifikation von impliziter Dokumentation, die aufgrund ihrer Relevanz in eine explizite Form transformiert werden sollte
- Hinzufügen häufig identifizierter Fragen und Antworten zu einer probaten Liste mit "Frequently Asked Questions"
- Recherche in Bugzilla nach dokumentationsspezifischen Bug Reports, welche Modifikation oder die Generierung von Dokumenten thematisieren

Als mögliche Kommunikationskanäle für Dokumentationsvorhaben stehen explizite Newsgroups und Mailing Lists zur Verfügung: "If you'd like to contribute, let us know by posting to the news://news.mozilla.org/ netscape.public.mozilla.documentation newsgroup and mailing webmaster@mozilla.org." [MozillaOrg02][164]

#### 4.2.11.1 Dokumentationsartefakte

"Mozilla documentation therefore should be freely distributable, modifiable, and open for use by anyone, including commercial entities." [MozillaOrg02][163] Damit gelten für freie Dokumentation ähnliche Kriterien wie für freie Software im Sinne der GNU GPL, wie auch Richard Stallman in [Stallman02] definiert und worauf sich auch [MozillaOrg02][165] beruft: "The criterion for a free manual is pretty much the same as for free software: it is a matter of giving all users certain freedoms. Redistribution (including commercial redistribution) must be permitted, so that the manual can accompany every copy of the program, on-line or on paper. Permission for modification is crucial too." [Stallman02]. Dokumentationsartefakte können in Gestalt von impliziten Kommentaren im Quellcode, expliziter Dokumentation in Form von Handbüchern, Manuals oder Tutorials, Dokumentation von APIs basierend auf Javadoc oder ähnlichem

164 http://www.mozilla.org/get-involved.html
165 http://www.mozilla.org/docs/contribute.html

vorliegen. Die Bearbeitung dieser Artefakte basiert auf einem Bugzilla-basierten Entwicklungsprozess.

Weiterhin existiert ein Styleguide [MozillaOrg02][166], welcher verschiedene Vorgaben für das Format von Dokumentationsartefakten und damit die Rahmenbedingungen für deren Entwicklung definiert. (siehe Kapitel 4.6.1.1.3)

#### 4.2.11.2 Documentation Review

Der Aspekt der Qualitätssicherung wird auch im Rahmen der Generierung von Dokumentationsartefakten in Form von informellen Reviewprozessen analog zur Entwicklung von Quellcodeartefakten berücksichtigt. Die positiven Erfahrungen der Bug Report-getriebenen Entwicklung von Patches hat dazu geführt, dass auch für die Entwicklung von Dokumentationsartefakten eine lebenszyklusorientierte Bearbeitung basierend auf Bug Reports vorgesehen wurde: "To facilitate the review process we have created a Documentation product in Bugzilla." [MozillaOrg02][165]. Diese spezielle Ausprägung eines Eintrages in das Bugzilla-System ermöglicht eine Bearbeitung und sukzessive Weiterentwicklung der Dokumentationsartefakte basierend auf ihrem jeweiligem Status, wie dies bereits für die Entwicklung von Patches anhand von Bug Reports in Bugzilla praktiziert wird.

In Bugzilla wird dabei zwischen den folgenden Zielgruppen von dokumentationsbezogenen Bug Reports unterschieden:

- Mozilla Developers
- Web Developers
- Users

Diese Kategorien definieren die Domäne der entsprechenden Dokumentation. So sind Artefakte der Kategorie 'Mozilla Developers' "probably only useful for people who compile mozilla" ([MozillaOrg02][165]) und umfassen sowohl explizite Handbücher für verschiedene Quellcodemodule als auch Dokumentation von APIs basierend auf Javadoc. Die Kategorie 'Users' betrifft lediglich Dokumentationsartefakte, die für das Hilfemenü von Mozilla vorgesehen ist während die Kategorie 'Web Developers' Artefakte umfasst, die sich an "content creators, web site administrators, system administrators and people using Mozilla as an application development platform" ([MozillaOrg02][167]) wendet.

166 http://www.mozilla.org/README-style.html
167 http://www.mozilla.org/docs/contribute.html

Basierend auf diesen Zielgruppen der Bug Reports wird ein Review-Prozess analog zur Entwicklung von Patches durchgeführt.

#### 4.2.11.3 Verwaltung und Bearbeitung von Dokumentationsartefakten

Durch die Instrumentalisierung von CVS zur Verwaltung der Dokumentationsartefakte, wurde ein Bearbeitungsworkflow analog zur Bearbeitung von Patches initiiert: "We use CVS to manage the documents on our web site and use the same mechanism to grant access to the document tree as to the source tree." [MozillaOrg02][167]

Existierende Dokumente stehen über CVS zur Verfügung, können über den anonymen CVS-Zugriff bezogen und anschließend lokal durch die Anwender editiert werden. Parallel zur Entwicklung eines Patches basierend auf Quellcode-Modifkationen mit dem Werkzeug Diff, kann mit diesem Werkzeug auch die Menge der Änderungen ausgehend von dem ursprünglichen Dokument entwickelt und in Form eines "Dokumentations-Patches" der Gemeinde zur Verfügung gestellt werden.

Primärer Adressat für ein solches Dokumentations-Patch ist der ursprüngliche Autor des bearbeiteten Dokumentationsartefakts. Falls der Dokument-Initiator nicht identifizierbar ist, werden die letzten Bearbeiter des Dokuments oder als letzte Alternative die Mozilla-Organisation kontaktiert: "If the document doesn't list its owner, click on the Last Modified link at the bottom of the page to see who has recently updated the page and ask them. If all else fails, mail webmaster@mozilla.org." [MozillaOrg02][167]

Außerdem ist es auch möglich, Änderungen direkt in das CVS-Archiv zu integrieren, wobei dieses Privileg durch einen Profilierungsprozess analog zu dem der Quellcodeentwicklung erreicht werden kann: "As with the source code, if you establish a track record of good work then you may be granted access to the repository, especially if you contribute documents and become their official owner and maintainer." [MozillaOrg02][167]

## 4.3 Managementprozesse

In diesem Abschnitt werden ausschließlich Prozesse dargestellt, die dem Metamodell entsprechend als Managementprozess betrachtet werden können. Diese Aktivitäten werden i.d.R. durch Mitglieder der Mozilla Organisation ausgeführt.

### 4.3.1 Initiale Prototypentwicklung- und Release

Das Mozilla-Projekt basiert auf dem Quellcode des kommerziell verwerteten Netscape-Browsers: "The original Netscape Navigator 5 codebase was donated by Netscape to mozilla.org, so there was a significant amount of pre-existing code at the time the Open Source project was officially started." [ReMa02]. Diese Praxis ist ein nicht ungewöhnliches Vorgehen im OSS-Kontext, und wurde z.B. auch im Rahmen des OSS-Projekts Zope[168] angewandt.

#### 4.3.1.1 Initiale Prototypentwicklung

Der initiale Prototyp des Projekts wurde, wie bereits erwähnt in einem proprietären Kontext durch ein kommerziell orientiertes Unternehmen entwickelt. Diese Methode nimmt somit die initiale Anforderungsdefinition und die Spezifikation der Software Architektur voraus: "This is not uncommon among OSS projects in general, but it presents significant changes to the 'natural' design and requirements processes in a conventional, from-scratch project." [ReMa02]

#### 4.3.1.2 Initialer Prototyprelease

Im Jahre 1998 wurde durch die Firma Netscape Communications der Quellcode des Projekts offengelegt und einem Open Source Prozess zugeführt. Netscape repräsentiert durch diesen Schritt das erste kommerzielle Unternehmen, welches sein primäres Softwareprodukt einem Open Source Prozess zuführte: "Netscape was the first proprietary corporation to release ist technology as Open Source;...." [Rosenberg00]

#### 4.3.1.3 Re-Engineering des Prototyps

Als ein Hauptproblem der Offenlegung des Quellcodes des Mozilla-Projekts kann der relativ späte Zeitpunkt im Entwicklungszyklus des Netscape-Browsers angesehen werden. Daher war der Quellcode bereits sehr komplex, genügte zudem nicht den expliziten Anforderungen der OSS-Entwicklung, wie z.B. Lesbarkeit und Modularität, und stellte somit eine Einstiegsbarriere für interessierte Entwickler dar: "Netscape's level of complexity, introduced by opening a preexisting project rather late in its development cycle, is probably the main reason for difficulties in attracting new developers. It is clear that the complexity of Netscape's code represents a formidable barrier of entry that is

168 http://www.zope.org

not easily overcome by even highly motivated and qualified developers." [Sandred01]

Trotz dieser umfangreichen Ausgangsbasis wurde daher ein beträchtlicher Anteil des Quellcodes im späteren Verlauf des Projektes neu entwickelt. Der veröffentlichte Quellcode stellt somit die maßgebliche Spezifikation der Anforderungen und auch des prototypischen Softwaredesigns dar: "... many of the requirements had been determined by the original code and documentation developed by Netscape." [ReMa02]. Als weitere Anforderung an die künftige Weiterentwicklung der Software wurde die HTML-Kompatibilität definiert: "The other main requirement established by the steering group was compliance to HTML standards, which is a very noteworthy intention, since the W3C had yet to see an organization devote itself to strictly implementing its standards." [ReMa02]

Das ursprüngliche Design des Mozilla-Browsers wurde direkt durch die Architektur des Nescape Browsers definiert: "Since it was essentially the same codebase, the original design matched the Netscape Navigator 5 design." [ReMa02]. Die Mozilla Organisation kam aber zu dem Schluss, dass es notwendig sein würde, den gegebenen Quellcode fast vollständig zu überarbeiten, um den Anforderungen, wie z.B. der Kompatibilität zu existierenden Standards, gerecht werden zu können: "After less than a year of development, however, the technical lead of mozilla.org came to the somewhat controversial conclusion that the original codebase would prove impossible to evolve to suit the requirements of a standards-compliant Web browser." [ReMa02]

### 4.3.2 Prozess der "Nightly Builds"

In diesem Abschnitt werden der Prozess der "Nightly Builds" und die mit ihm assozierten Teilprozesse im Hinblick auf ihre Eigenschaft als Releaseprozess dargestellt. Der zentral ausgeführte Prozess der Nightly Builds kann auch als eine spezielle Ausprägung des Prozesses zum Release der Software betrachtet werden.

Die folgende Grafik visualisiert den grundlegenden QA- bzw. Release-Prozess nach [CaLeCh02] im Kontext der "Nightly Builds", der nur vereinfacht dargestellt wird. Da er sehr stark regelementiert und formalisiert wurde, stellt er ein Charakteristika dar, welches für Entwicklungsprozesse im OSS-Kontext untypisch ist und somit für die Identifikation von generalisierbaren Strukturen nicht von elementarer Bedeutung ist.

### Schließen des Quellcode-Verzeichnisses

Den ersten Schritt im Rahmen des Testprozesses stellt nach [CaLeCh02] das Schließen des Quellcodeverzeichnisses durch den Maintainer bzw. Build Engineer dar, der dies über das Tinderbox-Werkzeug kommuniziert und über die Webschnittstelle des Bonsai-Tools den Status des Verzeichnisbaumes entsprechend modifiziert: "Using Bonsai to change the tree status to "closed" has the affect of closing the Mozilla CVS repository to code check-ins, which are not allowed until the tree reopens again." [CaLeCh02]. Diese Aktivität wird täglich um 18:00 Uhr (PST) durchgeführt: "Every day, at around 2pm GMT (6am PST), the tree is closed and a build is made for testing." [MozillaOrg02][169]

Die Werzeuge Tinderbox und Bonsai werden im Kapitel 4.7 noch explizit vorgestellt.

### Kompilierung

Nach der Schließung des Quellcodes wird das CVS-Repository für das Hinzufügen weiterer Patches gesperrt und die aktuellen Änderungen werden zu einem weiteren Build kompiliert: "Once the tree has been closed, the build engineer uses Tinderbox to build/compile the source code that has been checked in since the last tree closure." [CaLeCh02]. Falls bei der Kompilation Konflikte auftreten, kann mit Unterstützung des Tinderbox-Werkzeugs das fehlerhafte Quellcode-Artefakt identifiziert werden, um in Zusammenarbeit mit dem dafür zuständigen Entwickler den Fehler zu beheben oder das entsprechende Patch aus dem Quellcode zu entfernen: "Completing this task involves using Tinderbox to invoke the build/compilation of the source code on a set of automated build machines (a cluster), which pulls the source code from the Mozilla CVS repository. These automated build machines are tasked with compiling, testing and reporting results back to the Tinderbox server." [CaLeCh02]. Tinderbox visualisiert dabei den Kompilationserfolg über die systemeigene Webschnittstelle und signalisiert einen Mißerfolg entsprechend: "This invokes the process of fixing compilation errors, which produces a fix/patched source code. The build engineer commits the fix source into the tree and then restarts the build process. Due to the Mozilla community belief that all platforms are equal, the tree is not open for check-ins until all platforms have been successfully built." [CaLeCh02].

Dieser Prozess wird wiederholt, bis für die gebräuchlichsten Plattformen erfolgreich 'Builds' erstellt werden konnten: "With the artifact source code, he/

169 http://www.mozilla.org/quality/smoketests/faq.html

she generates a build for at least each of the major platforms (Window, Linux, and Mac OS)." [CaLeCh02]

### Publikation der Builds

Anschließend wird der Quellcode und die Binärdaten auf dem FTP-Server des Projektes veröffentlicht und somit im Rahmen des allgemeinen Test-Prozesses publiziert: "As soon as the tree has built completely green, we wait for the builds to appear on the FTP site (ftp://ftp.mozilla.org/pub/mozilla/), in a numbered directory […] This normally happens around 4pm GMT (8am PST). There's no way to predict when exactly." [MozillaOrg02][170]. Dies wird durch den sogenannten Smoketest-Koordinator in Form einer IRC-Nachricht initiiert: "However, until the seemingly trivial process of the smoketest coordinator giving the OK to download the build, volunteer testers and Mozilla QA engineers wait for the signal, which comes in the form of an IRC message on the #smoketest channel." [CaLeCh02]

### Test der Builds

Die Akteure des Test-Prozesses können den Quellcode nun konkreten Tests unterziehen, wobei diese Aktivitäten bereits im entsprechenden Abschnitt im Rahmen der Entwicklungsprozesse erläutert wurden.

### Kategorisierung und Behebung der Softwarefehler

Anschließend werden die kommunizierten Softwarefehler durch den Build Engineer als kritisch bzw. nicht kritisch kategorisiert: "From the informational messages on IRC, determinations of the seriousness of bug findings are made. Bugs are classified as either critical or non-critical to the current build." [CaLeCh02]. Kritische Fehler, welche das gesamtheitliche System negativ beeinflussen, werden möglichst sofort durch die entsprechenden Entwickler behoben, wobei diese i.d.R. mittels des Bonsai-Werkzeuges identifiziert werden: "For critical bugs/problems, the build engineer has the task of finding the source check-in and engineer that may have caused the bug (using Bonsai)." [CaLeCh02]. Nach der Integration des Bug-Fixes wird das System erneut kompiliert und dieser Vorgang so oft iterativ durchlaufen, bis das System erfolgreich getestet wurde.

Nicht kritische Fehler werden zur Ermöglichung einer späteren Behebung in Form eines 'Bug Reports' in Bugzilla vermerkt.

170 http://www.mozilla.org/quality/smoketests/faq.html

Nach erfolgreicher Absolvierung aller Tests wird der Verzeichnisbaum wieder geöffnet: "Once the build passes the required tests, and there are no longer any bugs deemed critical to the build, the sheriff and build engineer change the status of the tree to 'open'." [CaLeCh02]

### 4.3.3 Software Release

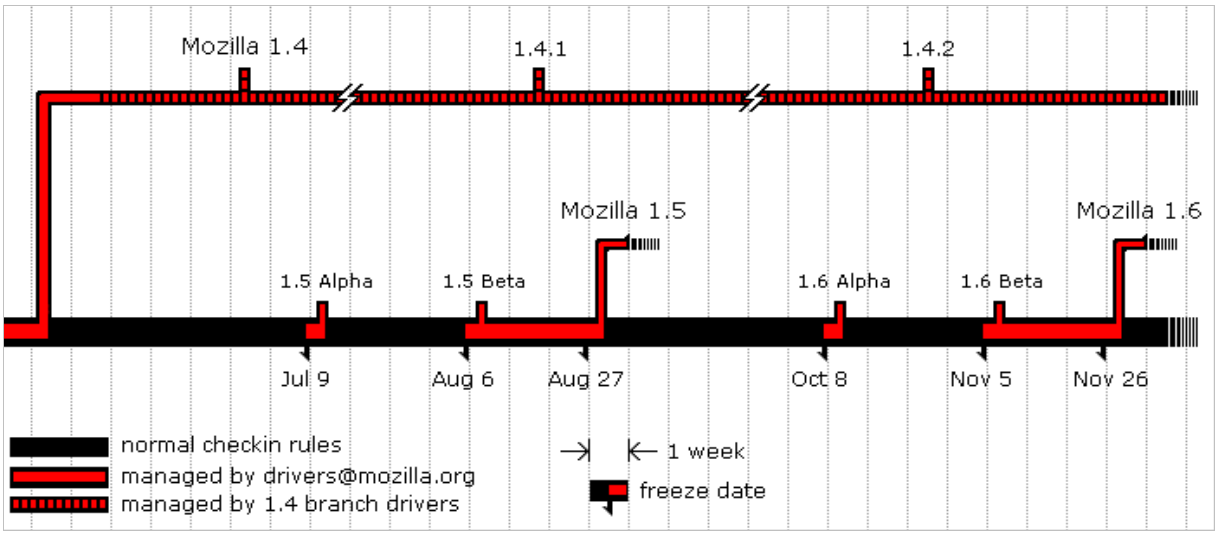
Der Prozess der Veröffentlichung neuer Software-Versionen wird durch die Mozilla Organisation ausgeführt: "We will collect changes, help authors synchronize their work, and periodically make new source releases which incorporate the best work of the net as a whole." [MozillaOrg02][171]

Der Release neuer Versionen wird in verschiedene, sogenannte Milestone Builds eingeteilt, wobei vor der Veröffentlichung einer offiziellen und stabilen neuen Version entsprechende Alpha- und Beta-Versionen veröffentlicht werden. Die Alpha- und Beta Milestone Builds werden als instabil betrachtet und dienen vor allem dazu, eine große Anwendergemeinde zu Testzwecken zu erreichen: "As we do not intend to promote alpha and beta releases for uses other than testing, we will ship those releases according to the predetermined schedule, and let the chips fall where they may." [Eich03a]. Im Gegensatz zu Quellcode-Modulen oder kompilierten Binärdaten, die im Rahmen der Nightly Builds oder über Entwicklerressourcen wie das CVS-Archiv oder den projektinteren FTP-Server zur Verfügung gestellt werden, werden die Alpha- und Beta-Releases bereits sowohl in kompilierter Form als auch als Quellcode auf der öffentlichen Projekt-Website veröffentlicht. Dadurch steht die Software allen interessierten Personen zur Verfügung und erreicht somit eine größtmögliche Zielgruppe für den Prozess der Software-Tests.

Die nachfolgende Grafik aus [Eich03b] stellt die praktizierten und geplanten Veröffentlichungsprozesse mit dem Stand vom 27.10.2002 dar:

171 http://www.mozilla.org/mission.html

Bild 12 Mozilla: Release Schedule des Mozilla-Projekts (aus [Eich03b])




Die Grafik visualisiert außerdem den Umstand, dass für jeden Release eines Milestone-Builds ein eigenständiger Zweig im Quellcode-Archiv ('Branch') generiert wird, der im Rahmen des Veröffentlichungsprozesses durch die sogenannten Drivers verwaltet wird.

Für die Veröffentlichung von umfassenden Milestone Builds wird ein Zeitraum von ca. drei Monaten eingräumt. Dabei wird immer mit einer Alpha-Phase über die Dauer von fünf Wochen begonnen, an welche sich die Beta-Phase für weitere fünf Wochen anschließt: "[...] each major milestone takes one quarter of a year or thirteen weeks to be delivered, and starts with a five-week 'alpha' milestone, during which destabilizing development should be done. This is followed by a five-week 'beta' milestone period during which less risky bug fixes, in particular followup and cleanup of the alpha phase work that tend to restabilize the codebase, should be targeted." [Eich03a]. Die Alpha-Phase dient dabei dem hinzufügen von notwendigen Änderungen und Modifikationen, die den Code noch einmal destabilisieren können, indem zum Beispiel neue Funktionalitäten hinzugefügt werden. Die Beta-Phase dient der lediglich der Restabilisierung des Quellcodes, indem in dieser Zeitspanne nur noch fehlerkorrigierende Patches integriert werden.

Die folgende Tabelle aus [Eich03b] gibt einen Überblick über die geplanten bzw. praktizierten Milestone Builds und deren Veröffentlichungstermine:

Tabelle 5 Mozilla: Geplante bzw. erfolgte Milestone Builds (Quelle: [Eich03b])

| Milestone | Start | Freeze | Branch | Ideal Release | Actual Release |
|---|---|---|---|---|---|
| 1.1 alpha | 09-Apr-2002 | 05-Jun-2002 | n/a | 07-Jun-2002 | 11-Jun-2002 |
| 1.1 beta | 07-Jun-2002 | 10-Jul-2002 | n/a | 12-Jul-2002 | 22-Jul-2002 |
| 1.1 | 12-Jul-2002 | 31-Jul-2002 | 02-Aug-2002 | 09-Aug-2002 | 26-Aug-2002 |
| 1.2 alpha | 02-Aug-2002 | 04-Sep-2002 | n/a | 06-Sep-2002 | 11-Sep-2002 |
| 1.2 beta | 06-Sep-2002 | 09-Oct-2002 | n/a | 11-Oct-2002 | 16-Oct-2002 |
| 1.2 | 11-Oct-2002 | 30-Oct-2002 | 01-Nov-2002 | 08-Nov-2002 | 26-Nov-2002 |
| 1.3 alpha | 01-Nov-2002 | 04-Dec-2002 | n/a | 06-Dec-2002 | 13-Dec-2002 |
| 1.3 beta | 06-Dec-2002 | 22-Jan-2003 | n/a | 24-Jan-2003 | 10-Feb-2003 |
| 1.3 | 24-Jan-2003 | 12-Feb-2003 | 14-Feb-2003 | 21-Feb-2003 | 13-Mar-2003 |
| 1.4 alpha | 14-Feb-2003 | 26-Mar-2003 | n/a | 28-Mar-2003 | 01-Apr-2003 |
| 1.4 beta | 26-Mar-2003 | 23-Apr-2003 | n/a | 25-Apr-2003 | 07-May-2003 |
| 1.4 | 25-Apr-2003 | 14-May-2003 | 16-May-2003 | 21-May-2003 | ? |
| 1.5 alpha | 16-May-2003 | 09-Jul-2003 | n/a | 11-Jul-2003 | ? |
| 1.5 beta | 11-Jul-2003 | 06-Aug-2003 | n/a | 08-Aug-2003 | ? |
| 1.5 | 08-Aug-2003 | 27-Aug-2003 | 29-Aug-2003 | 03-Sep-2003 | ? |

Eine wichtige Rolle in dieser Phase des Projekts wird durch die sogenannten Drivers ausgeführt, welche dafür zuständig sind, Quellcodemodifikationen von großer Bedeutung für einen nahenden Software-Release zu entwickeln oder deren Entwicklung zu forcieren: "In the configuration management phase, drivers are charged with the duty of committing code/ bug fixes that are deemed as important to the overall quality of the Mozilla product. The involvement of the drivers in this capacity usually only happens when the tree nears an important release and project management type decisions about what should be included in a particular release need to be made." [CaLeCh02]

### Freeze-Process

Ein wichtiger Bestandteil der Release-Prozesse stellt die Schließung des Quellcode-Archivs ("Freeze") dar. Dadurch wird die Integration von neuem Quellcode verhindert, um den existierenden Quellcode letzten Tests zu unterziehen und bestehende Fehler zu beheben. "During the freeze, only bugs deemed 'must-fix-for-this-milestone', typically last minute regressions, are fixed on the trunk, as everyone tries to qualify the daily builds as branch-worthy." [Eich03a]. Für den Freeze-Prozess im Rahmen der Alpha- und Beta-Releases werden gemeinhin nur zwei bis drei Tage veranschlagt, während die Schließung des Archivs vor einem offizielen neuen Release ca. eine Woche dauert: "We hope that alpha and beta milestones need no more than a few days' tree closure to prepare for release." [Eich03a]

Der Freeze-Prozess wird i.d.R. an einem Mittwoch um 0:00 Uhr Pacific Time gestartet, wobei automatisch durch die Mitglieder der Mozilla Organisation ein

entsprechender Branch generiert wird, der einen expliziten Qualitätssicherungs- und Test-Prozess durchlaufen muss: "Traditionally, the milestone freeze time is 11:59 P.M. Pacific Time (see the tinderbox page for notices) on a Tuesday. [...] Release staff@mozilla.org create the milestone branch, which receives even more intensive QA over the weekend, leading to a release at the earliest by the following Wednesday" [Eich03a]

Nach der Generierung eines neuen Quellcode-Zweiges im CVS-Archiv im Rahmen des Freeze-Prozesses, werden Änderungen des Quellcodes ausschließlich durch die Drivers integriert, die somit in dieser Phase für die Fragen des Tree-Managements zuständig sind: "The drivers are particularly active after the trunk is frozen for a milestone release, and in managing the milestone branch until a milestone is released." [MozillaOrg02][172]. Während dieser Phase unterstützen die Drivers explizit die Entwickler, welche noch Bug Fixes bzw. Patches notwendigerweise vor dem nächsten Release-Termin integrieren müssen. Außerdem überwachen die Drivers jegliche Quellcodeänderung und unterziehen diese einem speziellen Review-Prozess, der sich nicht speziell auf implementations- oder entwurfsspezifische Fragen, sondern auf die Wichtigkeit des Patches und dessen mögliche Auswirkungen auf die Stabilität und Performance des Systems bezieht: "[...] the drivers review focuses on the importance of that particular fix to the milestone release, and the risk to stability and performance presented by that particular patch." [MozillaOrg02][172]. Die technischen Aspekte der Patches sind in diesem Stadium optimalerweise bereits durch die üblichen Review- bzw. Super-Review Prozesse evaluiert worden, die bereits vor der Integration in den Entwicklungsquellcode stattgefunden haben.

## 4.4 Infrastrukturelle Prozesse

Dieser Abschnitt thematisiert die Prozesse, die zur Entwicklung und Verwaltung der projektweiten Infrastruktur notwendig sind und entsprechend dem Metamodell als infrastrukturelle Prozesse betrachtet werden können. Der Fokus liegt hierbei auf der Bereitstellung der verwendeten Kollaborations- und Kommunikationswerkzeugen, die im Rahmen der Entwicklungsprozesse instrumentalisiert werden, da die individuell eingesetzten Entwicklungswerkzeuge auf Seiten der Entwickler nicht zentral verwaltet und organisiert werden.

Da die Etablierung und Verwaltung aller zentral bereitgestellten Werkzeuge nicht zu den primären Entwicklungsprozessen zählt, sondern im Rahmen dieser Arbeit als projektbegleitende und periphere Prozesse betrachtet werden,

172 http://www.mozilla.org/about/roles.html

werden die infrastrukturellen Prozesse nur auf einem sehr hohen Abstraktionsniveau dargestellt. Aufgrund des zentralen und z.T. untransparenten Charakters dieser Prozesse, sind sie zudem relativ wenig dokumentiert und formalisiert.

Die Existenz und der Einsatz dieser Werkzeuge ist vor allem von besonderer Bedeutung für den Fokus dieser Arbeit, wenn sich daraus Implikationen oder Determinanten für den Entwicklungsprozess ableiten lassen. Daher werden entsprechend relevante Werkzeuge in einem expliziten Kapitel (Kapitel 4.7) näher dargestellt.

### 4.4.1 Infrastrukturelle Anforderungen

Wie [ReMa02] feststellt, hat die Evolution der Entwicklungsprozesse von den Netscape-internen, traditionellen Entwicklungsprozessen zu einer dezentralen, und heterogenen Softwareentwicklung nach den Prinzipien von Open Source zu einer Reihe von Anforderungen geführt, die an die Realisierung und Unterstützung dieser Praktiken gestellt werden: "This evolution has been guided by a need to suit the open community which has been attracted to the browser. Coordinating source code changes from a large number of concurrent, distributed developers, has generated a number of demands in both process and support tools." [ReMa02] definiert diese Anforderungen wie folgt:

- Versionskontrolle bzw. Konfigurationsmanagement
- Formalisierte Prozesse für die Integration von Quellcodeänderungen
- Starke Modularisierung des Quellcodes
- Einsatz gebräuchlicher Entwicklungswerkzeuge
- Etablierung von Kommunikationskanälen

Diese Anforderungen repräsentieren z.T. die klassischen Anforderungen von OSS-Projekten. Im Gegensatz dazu stehen aber die expliziten Anforderungen an eine starke Formalisierung der Prozesse, wie sie im Mozilla-Projekt zur Realisierung der Qualitätssicherung etabliert wurden. Zur Unterstützung dieser Anforderungen werden verschiedene Werkzeuge im Mozilla-Kontext eingesetzt: "To aid in the development processes, the Mozilla organization has developed and adopted many tools. These include Bugzilla for bug tracking, Bonsai for performing queries to the CVS code repository, Tinderbox (an automated build and regression testing system), CVS (Control Versioning System) for code repository, Internet Relay Chat (IRC) for message exchange, and LXR for browsing hypertext-based source code." [CaLeCh02]. Diese verschiedenen Werkzeuge, deren Verwendung bereits z.T. in den beschriebenen Entwicklungsprozessen dargestellt wurde, stellen eine wichtige Basis des Projekts dar und wurden z.T. durch projekteigene Ressourcen selbständig entwickelt.

Die Kommunikations- und Kollaborationswerkzeuge zollen den beschriebenen Anforderungen Tribut, die sich aus der Koordination der Aktivitäten einer heterogenen und dezentral organisierten Entwicklergemeinde ergeben: "Much of the development infrastructure used in the Mozilla Project was developed to deal with its massive organizational requirements. Software engineering tools to support distributed development were a requirement and developed since day one of the project's existence." [ReMa02]

Nachdem im folgenden die verschiedenen infrastrukturellen Prozesse dargestellt werden, wird die eigentliche Infrastruktur, also die Werkzeuge des Projekts explizit in einem eigenen Kapitel (Kapitel 4.7) erläutert, sofern dies nicht bereits im Kontext der Entwicklungsprozesse geschehen ist.

### 4.4.2 Bereitstellung von Kommunikationsinfrastrukturen

Die Mozilla Organisation, die primär mit den Aufgaben der Maintenance des Projekts betraut ist und somit auch den Hauptakteur im Kontext der infrastrukturellen Prozesse darstellt, beschreibt ihre diesbezüglichen Aufgaben wie folgt: "We will operate discussion forums (mailing lists, newsgroups, or whatever seems most appropriate.)" [MozillaOrg02][173]

Eine wichtige Anforderung ergibt sich aus der Abstinenz von direkter persönlicher Kommunikation: "One of the premises the Mozilla Project was based upon was that face-to-face communication should not be required for development, which is strictly the rule for most, if not all, OSS projects [...]" [ReMa02]. Somit müssen jegliche Entwicklungsprozesse auf der Instrumentalisierung einer zuverlässigen Kommunikationsinfrastruktur beruhen, die für jedes kommunikative Anwendungsszenario die entsprechenden Werkzeuge zur Verfügung stellt.

Echtzeitkommunikation stellt hierbei ein wichtiges Hilfsmittel für Problemlösungsprozesse dar, insbesondere sobald zentrale Personen des Entwicklerkerns in den Prozess involviert sind: "[...] real time communication provides an important mechanism to educate developers and clarify the code. Because many of the lead developers are often online and available, questions and informal design reviews can be quickly performed (at the cost of some dispersion and interruption)." [ReMa02]

Die Mozilla Organisation versucht anhand der existierenden Kommunikation der Entwickler zu identifizieren, ob ein Bedarf für weitere, themenspezifische Kommunikationskanäle existiert und stellt diese gegebenenfalls zur Verfügung:

173 http://www.mozilla.org/mission.html

"For now, we plan to use the traditional Usenet approach for deciding when a new forum should be created: that is, new forums are created when interest has been demonstrated by existing traffic. So, if there is not a newsgroup or mailing list specific to the topic that you are interested in, pick the one that is the closest fit, and discuss it there. If it becomes clear that there is enough traffic on the new topic to warrant its own forum, then we will create one." [MozillaOrg02][174]

### 4.4.3 Bereitstellung von Kollaborationsinfrastrukturen

Die Bereitstellung von zentralen Werkzeugen zur Koordination der Entwicklungsprozesse ist eine zentrale Aufgabe der Mozilla Organisation: "We will coordinate bug lists, keep track of and publicize works in progress, and generally attempt to provide 'roadmaps' to the code, and to projects based on the code." [MozillaOrg02][175]

Elementar für das Mozilla-Projekt sind hierfür die Werkzeuge für das Konfigurationsmanagement und in außerordentlichem Maße das System zur Unterstützung des Bug Tracking (Bugzilla). Gerade letzteres ist ein integraler Bestandteil des Mozilla-Projektes, da fast alle Entwicklungsprozesse auf den Informationen dieser Datenbank beruhen und über dieses Werkzeug koordiniert werden. Dadurch determiniert Bugzilla den Entwicklungsprozess in vergleichsweise mächtigem Umfang. Bugzilla wurde durch die Community des Mozilla-Projekts entwickelt und wird als eigenständiges Open Source-Projekt verwaltet.

Neben dem Bug Tracking System Bugzilla, stellt CVS eine wichtige weitere Ressource für das Konfigurationsmanagement und die Kollaboration der Entwickler dar. Außerdem wurden durch die Mozilla Organisation die Werkzeuge Tinderbox, Bonsai und LXR in den Prozess der kollaborativen Systementwicklung integriert und z.T. auch eigenständig entwickelt.

Da die infrastrukturellen Prozesse im Rahmen der Softwareentwicklung nur sekundären und unterstützenden Charakter besitzen, werden diese Prozesse in diesem Rahmen eingehend analysiert.

174 http://www.mozilla.org/community.html
175 http://www.mozilla.org/mission.html

## 4.5 Rollen

In diesem Abschnitt werden die Rollen dargestellt, die im Prozess der Mozilla-Entwicklung identifiziert werden konnten und die für eine Betrachtung des Prozessmodells von Bedeutung sind. Hierbei wird eine Aggregation von verschiedenen Akteuren zu spezifischen Rollen durchgeführt, die sich zwar z.T. an den Rollen orientieren, welche durch die Mozilla Organisation formal spezifiziert wurden, sich aber nicht zwingend darauf berufen wird. Vielmehr ist es das Ziel, die real existierenden Rollen darzustellen, welche sich formal oder informell im Mozilla-Kontext etabliert haben. Dabei orientieren sich die hier dargestellten Rollen z.T. an der Projektstudie in [CaLeCh02].

### 4.5.1 User

Die minimale Rolle im Mozilla-Kontext wird durch den Anwender der Mozilla Software repräsentiert, der als 'User' bezeichnet wird. Neben dem primären Prozess der Verwendung der Software, zeichnet sich diese Rolle durch den der Verwendung vorausgehenden Prozess des Bezugs der Software in binärer Form oder als Quellcode aus. Im letzten Fall ist der Prozess des Kompilierens ein weiteres wichtiges Szenario dieser Rolle.

Sobald der Akteur aktiv am Projekt mitwirkt , indem er z.B. eine Frage kommuniziert, einen Software-Fehler identifiziert und kommuniziert oder Patches bereitstellt, geht er direkt andere Rollen über, z.B. Contributor oder Developer.

### 4.5.2 Contributor

Ein Contributor repräsentiert die minimale Ausprägungen der aktiv mitwirkenden Rolle im Mozilla Kontext, deren Akteure z.B. eine Frage über eine Mailing Liste stellen, einen Änderungsvorschlag kommunizieren oder ähnliches. Die Akteure dieser Rolle tragen noch keinen Quellcode aktiv bei und sind daher von den Entwicklern abzugrenzen.

In diesem Kontext können verschiedene Spezialisierungen dieser Rolle unterschieden werden, die jeweils für spezifische Prozesse verantwortlich zeichnen:

- Tester
- Bug Reporter

#### 4.5.2.1 Tester

Neben den allgemeinen Testprozessen, die durch die User ausgeführt werden, existieren wie bereits dargestellt auch explizite Test Cases im Mozilla Kontext. Diese Test Cases können durch einen Anwender ausgeführt werden, der somit in die Rolle eines Testers übergeht: "Individuals in this role conduct the predefined smoketests and test cases against specific product areas (of their choice)." [CaLeCh02]. Dieser Rolle werden somit Aufgaben wie die Ausführung und Erstellung von Test Cases zugeordnet. Die Anforderung an die Qualifikation der Akteure dieser Rolle gehen nicht über die Anforderungen an einen User hinaus: "This role does not require a high level of technical expertise." [CaLeCh02]. Sobald ein Tester einen Fehler identifiziert und diesen in Form eines Bug Reports kommuniziert, geht er in die Rolle eines Bug Reporters über.

#### 4.5.2.2 Bug Reporter

Ein Anwender oder ein Tester, der einen eruierten Bug in Form eines Bug Reports kommuniziert, nimmt automatisch die Rolle eines Bug Reporters ein. "These agents report bugs into the Bugzilla tracking system and correspond with QA owners and contacts regarding test cases." [CaLeCh02]. In diese Klasse der Akteure werden entsprechend der Definition eines Bug Reports auch die Akteure zusammengefasst die einen RFE, also einen Request for Enhancement in Form eines Bug Reports beigetragen haben.

### 4.5.3 Developer

Die Rolle des Entwicklers wird von jeder Person bekleidet, die aktiv Quellcode zur Verfügung stellt oder modifiziert und die Änderungen kommuniziert. Ein Entwickler verfügt noch nicht über spezielle Privilegien und ist zur Integration seiner Quellcodeänderungen auf entsprechend privilegierte Rollen angewiesen. Durch aktive Mitwirkung am Projekt über einen längeren Zeitraum, in dem der Entwickler seine Kompetenz beweist, kann er aber in eine Rolle mit den enstprechenden Rechten und Verantwortlichkeiten übergehen: "We operate as a meritocracy, so the more good code you contribute, the more you will be allowed to contribute: that is, the better a developer you prove yourself to be through your actions, the more responsibility you will be given." [MozillaOrg02][176]

176 http://www.mozilla.org/mission.html

#### 4.5.3.1 Module Owners

Eine speziell privilegierte Ausprägung eines Entwicklers wird durch den sogenannten Modul Owner repräsentiert. Die Aufgaben und Privilegien dieser Rolle werden durch die Mozilla Organisation an die jeweiligen Akteure delegiert: "The basic model we intend to operate under is that each major module will have an owner designated by mozilla.org." [MozillaOrg02][176]. Die Mozilla-Website www.mozilla.org/owners.html bietet einen Überblick der verschiedenen aktuellen Module und ihrer Modul Owner. Der Begriff des Moduls im Mozilla-Kontext wird im Abschnitt 'Artefakte' noch genauer erläutert.

Eine wichtige Anforderung an einen Modul Owner ist die eingehende Kenntnis des Quellcodes des entsprechenden Softwarebausteins, die er vor allem durch den häufigen Beitrag von Patches unter Beweis gestellt hat: "That person will be someone who knows the code well; is actually involved in developing the code, works well in the open source environment, and who feels that they are up to the task of being the arbiter of what should go in to that module, and what shouldn't. This person will also be the primary coder, but this is not a requirement." [MozillaOrg02][176].

Ein Modul Owner ist der primäre Kompetenzträger und Ansprechpartner für die Bearbeitung von Fehlerberichten oder Patches, die mit seinem Modul assoziiert sind: "An Owner is responsible for fielding bug reports, enhancement requests, patch submissions, and so on. ... " [MozillaOrg02][176]

Modul Owner werden durch die Mozilla Organisation definiert, wobei keine formalen Kriterien für deren Ernennung existieren: "There are no hard and fast rules here, but one thing that is definitely true is that module owners will come to be judged, both by us and by the public, based on what they've accomplished." [MozillaOrg02][176]

#### 4.5.3.2 Bugzilla Component Owner

Der Akteur in der Rolle eines Bugzilla Component Owners ist oft identisch mit dem des korrelierenden Modul Owners. Dies resultiert aus der Tatsache, dass eine Component in Bugzilla eine Problemdomäne definiert, die häufig durch genau ein oder mehrere Software-Module abgedeckt wird.

Bug Reports in Bugzilla können individuellen Entwicklern zur Bearbeitung zugeordnet werden. Sofern kein expliziter Entwickler im Bug Report definiert wurde, wird der Bug Report an den Component Owner weitergeleitet, der die Bearbeitung übernimmt oder koordiniert: "A Bugzilla component owner is the default recipient of bugs filed against that component. When someone files a bug, he or she may specify to whom the bug will be assigned. But if no specific

assignment is made by the bug reporter, the bug will go to the Bugzilla component owner." [MozillaOrg02][177]

Die Aufgaben der Component Owners setzen sich zusammen aus dem regelmäßigen Review der Bug Reports, der Zuordnung von Bug Reports zu den entsprechenden Component Owners, der Sicherstellung der Existenz von Test Cases für ihre Komponente und der allgemeinen Verwaltung und Koordination der Bearbeitung der Bug Reports: "A Bugzilla component owner is not necessarily the person that fixes the bugs, instead this agent role ensures that bugs get fixed by the appropriate parties, which means monitoring progress and soliciting status reports is a daily task." [CaLeCh02]

#### 4.5.3.3 Reviewer

Der Reviewer von Quellcodeänderungen an einem Modul ist häufig deckungsgleich mit dem Modul Owner: "Since code is written on a per module basis, the module owner is usually the reviewer for any code developed in his/her area." [CaLeCh02]. Außerdem können verschiedene andere Entwickler als Reviewer agieren, wobei aber die oberste Entscheidungsgewalt bei dem Modul Owner liegt: "Reviewers can include module peers, but the official sign-off on the code comes from the module owner." [CaLeCh02]

#### 4.5.3.4 Super Reviewer

Die Super Reviewer repräsentieren eine spezielle Ausprägung eines Reviewers und sind Teil des formal durch die Mozilla Organisation spezifizierten Super Review Prozesses. Die Projekt Website stellt unter http://www.mozilla.org/hacking/reviewers.html eine Liste aller Super Reviewer zur Verfügung, die von der Mozilla Organisation ernannt wurden. Neben der primären Aufgabe des Review von Quellcodeänderungen kommuniziert der Super Reviewer Änderungsvorschläge und stellt die Einhaltung von projektweiten Programmierstandards sicher: "This person signs off on code, makes recommendations and ensures that the code being development meets community standards." [CaLeCh02]. Super Reviewer können ausschließlich durch die Mozilla Organisation ernannt werden und repräsentieren somit Akteure, die wie auch die Modul- und Component Owners, der Mozilla Organisation sehr nahe stehen.

177 http://www.mozilla.org/about/roles.html

### 4.5.4 Drivers

Die Rolle der Drivers führt verschiedene Projekt Management Aufgaben aus und agiert primär in den Phasen des Software Release Prozesses, respektive in der Phase des Code Freeze. Hierbei koordinieren die Drivers die Integration und Entwicklung von letzten Patches in den zu veröffentlichenden Quellcode im Rahmen dieser Phase: "The drivers are particularly active after the trunk is frozen for a milestone release, and in managing the milestone branch until a milestone is released." [MozillaOrg02][178]. Während des Code Freeze ist es ein zwingendes Kriterium, dass jeglicher neuer Quellcode durch die Drivers evaluiert wird, bevor er in den aktuellen Quellcode integriert werden kann: "During this time the Drivers watch the checkins very closely, and generally require that every patch be reviewed by the drivers before it is checked in." [MozillaOrg02][178]. Dabei fokussiert dieser Review nicht auf die technischen Aspekte sondern hinterfragt aus einer gesamtheitlichen Perspektive auf das Patch dessen Bedeutung für das Gesamtsystem und die Risiken, welche die Integration für die Stabilität und Performance des Systems darstellt: "... the drivers review focuses on the importance of that particular fix to the milestone release, and the risk to stability and performance presented by that particular patch." [MozillaOrg02][178]. Daher wird der Driver auch nicht als spezielle Ausprägung der Developer betrachtet, zumal ein Driver auch nicht notwendigerweise über Quellcodekenntnisse verfügen, sondern eher ein Verständnis der Gesamtarchitektur besitzen muss.

Die Driver werden durch die Mozilla Organisation ernannt und müssen intensiv mit der gesamten Architektur des Entwicklungsquellcodes vertraut sein, um mögliche Interdependenzen und Interaktionen zwischen verschiedenen Modulen und Patches absehen können. In der Development Roadmap des Projekts [MozillaOrg02][179] ist eine Liste der aktuellen Driver verfügbar.

### 4.5.5 QA Owner / QA Contact

Der Prozess der Qualitätssicherung wird in verschiedene Kategorien bzw. Produkt Domänen eingeteilt, wobei jede Kategorie im Rahmen dieses Prozesses einen dedizierten Owner und verschiedene Kontaktpersonen besitzt. "Each category or product area has a QA owner, which in turn has potentially several QA contact people." [CaLeCh02]. Diese Rollen erstellen und verwalten hauptsächlich Test Cases für ihre Kategorien: "QA owners and contacts write test and maintain test cases for their product/category area." [CaLeCh02]. Bedingt durch die historische Entwicklung des Projekts sind diese Akteure

178 http://www.mozilla.org/about/roles.html
179 http://www.mozilla.org/roadmap.html

typischerweise Angestellte der Firma Netscape Communications. Die URL http://www.mozilla.org/quality/qacontacts.html gibt einen Überblick über die aktuellen QA Kontaktpersonen.

### 4.5.6 Nightly Build Coordinator (Sheriff)

Diese Rolle ist zuständig für die Koordination des Prozesses der Nightly Builds und wird typischerweise durch ein langfristig aktives Mitglied der Mozilla Community ausgeführt, das i.d.R. auch andere Rollen (z.B. Modul Owner, Component Owner) innehat. Seine Aufgaben sind deckungsgleich zu denen der in [CaLeCh02] definierten Rolle des Sheriffs und beinhalten primär die Schließung und Öffnung des Quellcodes: "The sheriff is responsible for overseeing the daily build of the tree; which includes closing the tree prior to build, adding comments about the build, and finally reopening the tree when a build succeeds." [CaLeCh02]. Außerdem kontaktiert der Nightly Build Coordinator die Mitglieder der Entwicklergemeinde, sobald Probleme bei der Erstellung des Builds auftauchen.

Aufgrund des zentralen Charakters dieser Rolle und der Notwendigkeit zum direkten und erweiterten Zugriff auf die zentralen Kollaborationswerkzeuge steht diese Rolle der Mozilla Organisation sehr nahe. Über die Transition eines Akteurs in die Rolle des Nightly Build Coordinators konnten zwar keine Informationen identifiziert werden, aber es wird davon ausgegangen, dass diese Rolle ebenfalls durch die Mozilla Organisation vergeben wird.

### 4.5.7 Build Engineer

Der Build Engineer ist primär für die Kompilierung des Quellcodes in Tinderbox zuständig [CaLeCh02] und arbeitet im Rahmen der Nightly Builds eng mit dem Nightly Build Coordinator zusammen. Sobald Probleme bei der Kompilierung auftreten koordiniert er die Behebung des Fehlers mit dem Nightly Build Coordinator, der eine entsprechende Anfrage an den verantwortlichen Entwickler kommuniziert. Der Build Engineer benötigt wie der Nightly Build Coordinator erweiterte Zugriffsmöglichkeiten auf die zentralen Kollaborationswerkzeuge. Es wird auch in diesem Fall angenommen, dass der Build Engineer durch die Mozilla Organisation ernannt wird.

### 4.5.8 Maintenance Instanz - Mozilla Organisation

Die Maintenance des Projektes wird durch die gesamte Mozilla Organisation augeführt. Dabei unterstützt sie das im Grunde autarke und selbstregulierende System durch koordinierende und steuernde Aktivitäten und ist zuständig für strategische Entscheidungen, die den Projektverlauf betreffen: "Members in

these roles are responsible for the overall direction of the community. They maintain the community portal (http://www.mozilla.org), develop tools, assist developers, and create policies and procedures." [CaLeCh02]. Die Mozilla Organisation repräsentiert somit die primäre Institution, welche in erster Linie für die infrastrukturellen und die Managementprozesse verantwortlich zeichnet, sofern sie die Aufgaben nicht an ihr nahestehende Rollen, wie Modul Owner, Driver etc. delegiert hat.

Außerdem definiert die Mozilla Organisation die Milestone Releases des Projekts und den Umfang und Zeitpunkt der Veröffentlichung der eigenen Quellcode Distributionen, wobei natürlich jeder interessierte Entwickler über die nötigen Privilegien verfügt, selbständig Quellcode-Distributionen zu vertreiben: "But, we here at mozilla.org decide what we put in the source distributions that we make. So that gives us final say on what goes into our distributions." [MozillaOrg02][180]. Die Mozilla Organisation setzt sich aus den Mitgliedern (Mozilla Staff) und den mit ihr assoziierten Personen (Associates) zusammen, wie bereits im entsprechenden Abschnitt dargestellt wurde. Die URL http://www.mozilla.org/about/stafflist.html gibt einen Überblick über die aktuellen Mitglieder der Mozilla Organisation.

## 4.6 Artefakte

Dieser Abschnitt stellt die identifizierten Artefakte im Mozilla-Kontext dar.

### 4.6.1 Managementartefakte

Die Kategorie der Management Artefakte wird entsprechend dem Metamodell in strategische und operative Management Artefakte unterteilt,die sich im Mozilla-Kontext aus Programmier-Richtlinien, Quellcode-Roadmaps und ähnlichen Dokumenten zusammensetzen: "The website also provides coding guidelines, source code roadmaps and lists of works in progress [...]" [NoSt98]

#### 4.6.1.1 Strategische Managementartefakte

Die strategischen Managementartefakte stellen Dokumente von strategischer Bedeutung für die Entwicklungsprozess dar, welche das praktizierte Entwicklungsmodell erheblich und langfristig determinieren und reglementieren.

180 http://www.mozilla.org/mission.html

#### 4.6.1.1.1 Mozilla Public License

Ein elementares strategisches Artefakt für den weiteren Projektverlauf wird durch die verwendete Lizenz, respektive der Mozilla Public License repräsentiert.

Die MPL basiert auf der GNU General Public License [GnuGpl91] , ist aber nicht direkt mit ihr kompatibel. Die aktuelle Version der MPL kann auf der Mozilla Webseite [MozillaOrg02][181] eingesehen werden.

[MozillaOrg02][182] weist darauf hin, dass momentan Teile des Quellcodes noch unter der Netscape Public License veröffentlicht werden: "At the moment, parts of the source are available under either the Netscape Public License (NPL) or the Mozilla Public License (MPL), often in combination with either the GNU General Public License (GPL) or the GNU Lesser General Public License (LGPL), or both.". Es ist geplant, jeglichen Quellcode unter einer MPL/GPL/LGPL bzw. einer NPL/GPL/LGPL-Triple License zu veröffentlichen, wobei die sich daraus ergebenden Implikationen für die Entwicklung und die Distribution des Quellcodes in diesem Kontext aus Gründen mangelnder Relevanz nicht weiter erläutert werden sollen: "mozilla.org is working towards having all the code in the tree licensed under a MPL/LGPL/GPL tri-license..." [MozillaOrg02][182]

#### 4.6.1.1.2 Mozilla Licensing Policy

Die Mozilla Licensing Policy definiert ein Informationsobjekt, welches die Vorgaben für die Lizensierung von jeglichem, durch die Akteure der Community entwickelten Quellcode definiert. Diese verbindlichen Vorgaben bezüglich der Lizensierung eigener Patches durch die Entwickler, gelten für jeglichen Quellcode, der in das zentrale CVS-Repository integriert werden soll: "Any code checked into our CVS tree needs to comply with the licensing policy." [MozillaOrg02][182]

#### 4.6.1.1.3 Projektweite Richtlinien

Weiterhin existieren im Rahmen des Projektes verschiedene Richtlinien, welche projektweite Vorgaben bezüglich der Durchführung verschiedener Aktivitäten definieren. Diese Dokumente werden i.d.R. durch die Mozilla Organisation verfasst bzw. einem Review durch ihre Mitglieder unterzogen, bevor sie als möglichst verbindliche Verhaltens-Leitlinien auf der Website veröffentlicht

181 http://www.mozilla.org/MPL/MPL-1.1.html
182 http://www.mozilla.org/MPL/

werden. Damit versucht die Mozilla-Organisation, den heterogenen und verteilten Prozess der Software-Entwicklung soweit wie möglich zu regulieren und zu formalisieren, um die kollaborative Software-Entwicklung zu unterstützen.

### Kommunikationsleitfaden

Unter der URL http://www.mozilla.org/community.html ist ein Dokument verfügbar, welches die verschiedenen Kommunikationskanäle wie Mailing Listen, Newsgroups, Foren u. ä. vorstellt und die adäquaten Kommunikationskanäle für verschiedene Anwendungsszenarien definiert. Außerdem werden verschiedene Vorgaben bezüglich der Art und Weise der Kommunikation und der zu kommunizieren Inhalten definiert.

### Bug Writing Guidelines

Ein umfassender Leitfaden bezüglich der Generierung von Bug Reports und den damit verbundenen Inhalten steht auf der Projekt Webseite [MozillaOrg02][183] zur Verfügung. Dieses Dokument definiert die Inhalte und die Workflows die zur Erstellung eines Bug Reports notwendig sind: "These bug writing guidelines, and the Bugzilla bug tracking system found at bugzilla.mozilla.org, are for reporting bugs in mozilla.org open-source projects [...]"

### C++ Portability Guide

In [Williams98] wurde ein sogenannter "C++ Portability Guide" definiert, der für das Projekt als Programmier-Richtlinie verwendet wird, um den Ansprüchen an eine größtmögliche Portabilität des Quellcodes gerecht zu werden: "What follows is a set of rules, guidelines, and tips that we have found to be useful in making C++ code portable across many machines and compilers." .

Dieses Artefakt wird als strategisches Management-Artefakt kategorisiert, obwohl es primär für den Prozess der Implementation relevant ist, da es ein Dokument von strategischer Bedeutung darstellt und der strategisch wichtigen Anforderung der Portabilität Rechnung trägt.

In [MozillaOrg02][184] wird auf die Notwendigkeit dieses Artefakts hingewiesen: "There are people porting Mozilla to embedded platforms using compilers that

183 http://www.mozilla.org/quality/bug-writing-guidelines.html
184 http://www.mozilla.org/hacking/

may not support all the latest standards. If you are writing compiled code (C, C++), help these platforms by following our C/C++ Portability Guidelines ...."

### Programmierkonventionen

In [MozillaOrg02][185] werden verschiedene Programmier-Richtlinien, wie z.B. Namenskonventionen oder Vorgaben für die Quellcodeformatierung definiert.

### Mozilla Tree Rules

In den Mozilla Tree Rules [MozillaOrg02][186] wird der Workflow der Integration von Quellcode thematisiert, um diesen Prozess so weit wie möglich zu formalisieren.

### XUL- und CSS Guide

Der XUL- und CSS-Guide [MozillaOrg02][187] definiert Vorgaben zu Erstellung von standardisierten Benutzerschnittstellen über die XUL-Technologie bzw. die Verwendung von Cascading Style Sheets (CSS).

### Reviewers Guide

Der Prozess des Reviews wird durch die Richtlinien des "Reviewers Guide" ([MozillaOrg02][188]) reguliert und standardisiert.

### Dokumentations-Style Guide

Bezüglich der Generierung von Dokumentation, also Inhalten, welche über die Projekt-Website publiziert werden sollen, wurde eine Style Guide definiert, welcher die Anforderung der größtmöglichen Browserkompatibilität der Webinhalte Rechnung trägt: "Our goal is to create content that works with a wide variety of browsers, even old, lame ones." [MozillaOrg02][189]

Dieser Style Guide ist unter http://www.mozilla.org/README-style.html verfügbar und definiert verschiedene Standardisierungen für die Erstellung von

185 http://www.mozilla.org/newlayout/doc/codingconventions.html], http://www.mozilla.org/hacking/best-coding-practices.html
186 http://www.mozilla.org/hacking/rules.html
187 http://www.mozilla.org/xpfe/skins.html
188 http://www.mozilla.org/projects/seamonkey/rules/code_review.html
189 http://www.mozilla.org/docs/contribute.html

HTML-basierten Dokumentationsartefakten. Eine wichtige Anforderung ist z.B. die Verwendung von standardisierten und weithin lesbaren Datenformaten und HTML-Elementen: "... make sure your document's main content is readable by everyone. In particular, use GIF images instead of PNG. Use HTML, not XML. Don't require the use of JavaScript except for demos since many hackers leave JavaScript turned off." [MozillaOrg02][189]

**HowTo-Guides**

Außerdem definieren verschiedene Howto-Guides explizit verschiedene Teilprozesse, wie z.B. das Verfassen von Bug Reports o.ä. Tätigkeiten. [MozillaOrg02][190] referenziert auf einige dieser Artefakte.

#### 4.6.1.2 Operationale Managementartefakte

Die operationalen Managementartefakte dienen ebenfalls der Regulierung und Steuerung des OSS-Projekts und sind aufgrund einer kürzeren und operativen Wirksamkeit bzw. häufigeren Aktualisierungen der Inhalte von den strategischen Artefakten abgrenzbar.

##### 4.6.1.2.1 Development Roadmap

Die Development Roadmap [MozillaOrg02][191] definiert verschiedene strategische und operative Anforderungen des Projekts, die operativen Meilensteine und den Zeitplan für die damit verbundenen Software Releases. Außerdem wird ein Überblick über die mit der Verwaltung des Quellcode-Archivs (Tree Management) verbundenen Termine und Prozesse gegeben: "It proposes key 'road rules' and a release schedule for ongoing Mozilla-the-browser source milestone releases, from which anyone can build commercial and other products. It also hints at how Mozilla-the-platform should evolve, again from an operational or 'release process' point of view." Da diese Roadmap somit die operative Ausführung der Entwicklungs- und Managementprozesse steuert und unterstützt und regelmäßig durch die Management Instanzen des Projekts aktualisiert wird, kann es entsprechend dem Metamodell als operationales Managementartefakt betrachtet werden.

190 http://www.mozilla.org/quality/help/
191 http://www.mozilla.org/roadmap.html

#### 4.6.1.2.2 Status Updates

Die Status Updates sind ein operatives Artefakt zur Kontrolle des Status der Entwicklung des Browsers, die von einem entsprechend legitimierten Mitglied der Mozilla Organisation erhoben werden. [MozillaOrg02][192] definiert sie als wöchentlich erstellte Berichte, die durch eine Management Instanz ausgewertet werden: "Weekly reports on the status of the browser collected by Tim Rowley."

Die Status Updates umfassen eine Beschreibung der aktuell integrierten Patches oder geplanter Entwicklungen und sind ebenfalls als operationales Managementartefakt zu kategorisieren.

### 4.6.2 Technologische Artefakte

Die technologischen Artefakte sind von primärer Relevanz für die Entwicklung des Quellcodes und peripherer Artefakte. Sie sind entsprechend dem Metamodell direkt mit einer oder mehreren Phasen der Systementwicklung assoziierbar und haben auch nur in diesen Phasen Gültigkeit.

#### 4.6.2.1 Anforderungsartefakte

Dieser Abschnitt beschreibt die identifizierten Artefakte, welche primär in der Phase der Anforderungsanalyse von Bedeutung sind, wobei der Zeitpunkt ihrer Generierung aber durchaus einer anderen Phase zugeordnet werden kann.

#### 4.6.2.1.1 Bug Report

Prinzipiell basiert jede Modifikation an bestehendem Quellcode in Mozilla auf einem sogenannten Bug Report, in welchem die zu implementierenden Anforderungen definiert werden. Im Rahmen dieser Anforderungsartefakte werden sowohl klassische Softwarefehler dokumentiert als auch funktionale Systemerweiterungen der Software in Form von sogenannten Enhancement Requests definiert. Da jeglicher Patchentwicklungsprozess auf einem individuellen Bug Report basiert, sind diese Anforderungsartefakte von zentraler Bedeutung für den gesamten Entwicklungsprozess.

192 http://www.mozilla.org/mozorg.html

#### Anforderung der Softwarefehlerbehebung

Der Bug Report definiert einen identifizierten Softwarefehler. Dadurch stellt er das grundlegende Anforderungsartefakt bezüglich der Implementation eines Bug Fix in Form eines Patch dar, welches das identifizierte Fehlverhalten beseitigt. Aus Verständnisgründen wurden die speziellen Metadaten, Eigenschaften und Merkmale eines Bug Reports und dessen Format bereits im Rahmen der Prozessbeschreibung näher dargestellt.

#### Enhancement Request

Neben den konventionellen Fehlerberichten existieren sogenannte Enhancement Requests, die auch RFE (Request For Enhancement) genannt werden. Diese werden genau wie ein klassischer Bug Report in Bugzilla verwaltet und organisiert: "Not all 'bugs' are bugs. Some items in the database are known as Enhancement Requests or Requests For Enhancement (RFE for short)." [MozillaOrg02][193]. Ein Enhancement Request verfügt analog zu klassischen Bug Reports über verschiedene Metadaten und unterscheidet sich lediglich durch eine spezifische Ausprägung des Severity-Attributs: "An RFE is a bug whose severity field is set to 'enhancement'. People often say 'bug' when they mean 'item in Bugzilla', so RFEs often wind up being called bugs." [MozillaOrg02][193].

Durch das Artefakt RFE wird die Koordination und die Entwicklung der funktionalen Systemerweiterung ermöglicht und unterstützt: "Enter the tasks you're planning to work on as enhancement requests and Bugzilla will help you track them and allow others to see what you plan to work on." [MozillaOrg02][193]

### 4.6.2.2 Entwurfsartefakte

Entwurfsartefakte existieren i.d.R. nicht explizit, da im Rahmen des OSS-Prozesses Software-Architektur kaum formal spezifiziert wird. Dafür enthalten diverse technologische Artefakte anderer Projektphasen implizite Informationen über das Design und die Architektur des Systems. Die sogenannte Concrete Architecture, also die bereits implementierte Architektur, ist implizit primär den Implementationsartefakten, respektive dem existierenden Quellcode zu entnehmen. In manchen Fällen existieren auch explizite Dokumentationen des Software-Entwurfs, wobei diese aber nach [ReMa02] i.d.R. schnell obsolet werden, da sie nicht mit der gleichen Geschwindigkeit

193 http://www.mozilla.org/bugs/

und Frequenz aktualisiert werden, mit welcher der Quellcode modifiziert wird. Dadurch überholt die Evolution des Quellcodes die Evolution der Dokumentation in vielen Fällen.

Die sogenannte Conceptual Architecture, also das geplante Design der Software, ist ebenfalls in diversen Dokumentationsartefakten implizit und z.T. auch explizit spezifiziert. Eine wichtige Grundlage stellen zum Beispiel die Bug Reports dar, welche z.T. den Entwurf eines zu implementierenden Patches vorwegnehmen.

#### 4.6.2.3 Implementationsartefakte

Dieser Abschnitt spezifiziert die technologischen Artefakte, welche primär in der Phase der Implementation von Bedeutung sind, oder in dieser Phase generiert werden.

##### 4.6.2.3.1 Patch

Ein Patch repräsentiert den Quellcode, welcher basierend auf einem Bug Report oder einem Enhancement Request entwickelt wurde. Die Prozesse der Entwicklung und Bearbeitung des Patches und dessen Charakteristika und Format wurden aus Verständnisgründen bereits in der entsprechenden Prozessdarstellung näher erläutert und werden an dieser Stelle nicht weiter thematisiert.

##### 4.6.2.3.2 Modul

Ein Modul repräsentiert im Mozilla-Kontext eine bestimmte Menge von Quellcode-Dateien, die eine spezifische Funktionalität erfüllen: "A Module is a collection of source files that form a coherent bundle, probably a library (whether DLL/DSO or static)." [MozillaOrg02][194]. Einige Module sind ausschließlich in Form von Quellcode verfügbar, während andere auch in binärer Form veröffentlicht werden: "Some modules, notably the Front Ends, link as executables." [MozillaOrg02][194]

Wie bereits dargestellt wurden, konnten verschiedene Akteure identifiziert werden, welche für spezifische Quellcodesegmente, respektive Module, als

194 http://www.mozilla.org/owners.html

primäre Kompetenzträger betrachtet werden können. [MozillaOrg02][195] liefert einen Überblick über verschiedene Module, ihre Owner und verschiedene modulspezifische Newsgroups.

#### 4.6.2.3.3 License Boilerplate

Die sogenannte License Boilerplate ist essentieller Bestandteil jeder Quellcode-Datei im Mozilla-Kontext und definiert die Lizenz, unter welcher der Quellcode veröffentlicht wird. Je nach Art der Datei existieren verschiedene vorformatierte Versionen, welche nur noch durch die spezifischen Informationen bezüglich des Autors und der Datei ergänzt werden müssen: "This table contains preformatted versions of the boilerplate text that goes at the top of files which are licensed under the MPL/LGPL/GPL (the preferred mozilla.org license for new Mozilla code) or the MPL." [MozillaOrg02][196]. Ein Beispiel ist das im folgenden dargestellte MPL-License Boilerplate für C-Dateien:

```
/* ***** BEGIN LICENSE BLOCK *****
 * Version: MPL 1.1
 *
 * The contents of this file are subject to the Mozilla
   Public License Version
 * 1.1 (the "License"); you may not use this file
   except in compliance with
 * the License. You may obtain a copy of the License at
 * http://www.mozilla.org/MPL/
 *
 * Software distributed under the License is
   distributed on an "AS IS" basis,
 * WITHOUT WARRANTY OF ANY KIND, either express or
   implied. See the License
 * for the specific language governing rights and
   limitations under the
 * License.
 *
 * The Original Code is
   ____________________________________________.
 *
 * The Initial Developer of the Original Code is
 * ______________________________________________.
```

195 http://www.mozilla.org/owners.html
196 http://www.mozilla.org/MPL/boilerplate-1.1/

```
* Portions created by the Initial Developer are
  Copyright (C) 2___
* the Initial Developer. All Rights Reserved.
*
* Contributor(s):
*
* ***** END LICENSE BLOCK ***** */
```

#### 4.6.2.4 Einsatzartefakte

In diesem Abschnitt werden die technologischen Artefakte dargestellt, welche der Phase des Einsatzes der Software zugeordnet werden können.

##### 4.6.2.4.1 Software Releases

Im Rahmen der Software Releases kann zwischen verschiedenen Kategorien unterschieden werden, die durch die Mozilla Organisation bzw. dedizierte Akteure zur Verfügung gestellt werden. Wie bereits im Rahmen der Beschreibung des Release-Prozesses bzw. des Prozesses der Nightly Builds dargestellt wurde, konnten die folgenden Artefakte identifiziert werden:

- Nightly Build
- Alpha Release
- Beta Relase
- Official Release

Alpha- und Beta Releases werden genauso wie Nightly Builds zu Testzwecken im Rahmen bzw. im Vorfeld der Generierung eines offiziellen Software Release veröffentlicht. Diese Test Releases unterscheiden sich lediglich im Grad ihrer Stabilität und Fehlerdichte, da sie sukzessive zur Fehlerbehebung publiziert werden. Außerdem werden Nightly Builds, also kompilierte Snapshots verschiedener Quellcodesegmente, nur in kompilierter Form zur Verfügung gestellt und adressieren in erster Linie die Entwickler des Projekts. Alpha- und Beta-Releases werden meistens sowohl als Quellcode, als auch in binärer Form zur Verfügung gestellt und richten sich an Entwickler und auch an interessierte Anwender, welche diese Releases lokal einsetzen und somit zur Identifikation von weiteren Anforderungen beitragen.

#### 4.6.2.4.2 Test Case

Ein Test Case spezifiziert formal die Aktivitäten zur Ausführung eines speziellen Tests, wie z.B. einem Layout Test oder einem Smoke Test. Test Cases werden entweder zentral durch die Mozilla Organisation definiert, wie dies im Falle der Smoke Tests geschieht, oder aber durch die Entwicklergemeinde generiert, wie dies i.d.R. bei Layout Test Cases durchgeführt wird.

#### 4.6.2.4.3 Dokumentation

Dokumentationsartefakte existieren z.T. implizit in Form von Kommentaren des Quellcodes oder in Form eines Bug Reports, der ein Patch beschreibt. Weiterhin enthalten viele Diskussionen über Newsgroup- oder Mailing List Nachrichten implizite Dokumentation von Patches oder Software-Modulen.

Explizite Dokumentationsartefakte stellen eine weitere Kategorie dieser Artefakte dar, wobei vor allem die Ausprägungen als Webbasierte Dokumentation in Form von Manuals, Frequently Asked Questions oder ähnlichem von Bedeutung sind.

Dokumentation von zentralen und häufig gestellten Fragen in Form von FAQ-Listen stellt ein zentrales Dokumentationsartefakt im OSS-Kontext dar. Elementare Inhalte, die häufig in Mailing Listen oder Newsgroup-Diskussionen identifiziert werden können, werden in Form einer verallgemeinerten Frage und der entsprechenden Anwort formuliert und in Form von themenspezifischen Listen zusammengefasst. Die URL http://www.mozilla.org/faq.html beschreibt die verschiedenen FAQ-Listen des Mozilla-Projektes, die neben den allgemeinen Fragen z.B. auch explizit Fragen bezüglich des Entwicklungsprozesses, bezügliche einzelner Softwaremodule oder anwendungsspezifische Fragen in jeweils einer eigenständigen Liste verhandeln.

## 4.7 Werkzeuge

Dieses Kapitel definiert die Werkzeuge des Mozilla Projekts, die entsprechend dem Metamodell in die zentralen Kommunikations- und Kollaborationswerkzeuge und die dezentralen Entwicklungswerkzeuge eingeteilt werden.

### 4.7.1 Kommunikationswerkzeuge

Wie auch [ReMa02] feststellt, stellt die Abstinenz von persönlicher Kommunikation hohe Anforderungen an die Kommunikationsinfrastruktur:

"The fact that developers rarely meet personally has some consequences which are often overlooked. First, participants are forced to communicate clearly and through written English, using email, online chat, newsgroups and bug comments.". Diese Infrastruktur wird durch die Mozilla Organisation im Rahmen der infrastrukturellen Prozesse instrumentalisiert, koordiniert und verwaltet: "The Mozilla organization provides communication channels for developers such as IRC, newsgroups and mailinglists." [NoSt98]

Neben diesen Werkzeugen, die primär und ausschließlich der Aufgabe der

Kommunikation dienen, existieren verschiedene weitere Werkzeuge, deren primärer Verwendungszweck die Ermöglichung von kollaborativer Arbeit ist, die aber damit auch der Kommunikation unter den Projektmitgliedern dienen. Bugzilla ist hierfür ein sehr geeignetes Beispiel: "The Mozilla community uses Bugzilla to a great extent as a communication tool …." [ReMa02]

#### 4.7.1.1 Projekt Website

Das primär eingesetzte Werkzeug zur Kommunikation von Informationen der Mozilla Organisation an die Entwicklergemeinde und zur Bereitstellung aller projektrelevanten Ressourcen, Artefakte und Informationen ist die Website des Projekts. "All the development activity that goes into coordinating builds and maintaining the Mozilla browser has been done over the Internet using the Mozilla.org web portal as the central communications device." [CaLeCh02]

Damit stellt diese Plattform ein zentrales Portal dar, über das alle relevanten Prozesse gebündelt und in transparenter Form einer kollaborativen Bearbeitung zugeführt werden. Zudem repräsentiert die Website des Projekts die Schnittstelle zu allen Kollaborationswerkzeugen, wie CVS oder Bugzilla, die im Rahmen der Entwicklungsprozesse zentral verwendet werden.

#### 4.7.1.2 Mailing Lists / Newsgroups

Generell werden die meisten entwicklungsrelevanten Diskussionen über die Mailing Listen und Newsgroups des Projekts ausgeführt. "Mailing lists and newsgroups are generally used for discussion that is less focused than bug comments: new features, design review, measurements and statistics, and less technical inquiries." [ReMa02]. Dabei existieren verschiedene allgemeine Kommunikationskanäle, während für spezifische Aufgaben eigenständige Newsgroups und Mailing Listen durch die Mozilla Organisation etabliert werden, sobald ein entsprechendes Interesse auf Seiten der Entwickler besteht. Weiterhin ist ein wichtiges Merkmal des Mozilla-Projekts, dass für jeden Verwendungszweck sowohl eine Mailing Liste als auch eine Newsgroup mit den äquivalenten und somit redundanten Inhalten etabliert wird, um den

individuellen Präferenzen der Entwickler entgegen zu kommen. Die Inhalte werden regelmäßig gespiegelt: "Since some people prefer newsgroups, and some people prefer mailing lists, we have created our discussion forums in pairs: both a newsgroup and a mailing list. Each one mirrors its mate: messages sent to one of the newsgroups will also show up in its corresponding mailing list, and vice versa." [MozillaOrg02][197]

Neben den Kommunikationsressourcen für die Entwickler des Projektes stehen auch verschiedene Mailing Listen und Newsgroups für den Support der Anwender zur Verfügung: "The newsgroups for discussing and getting help with Netscape products are described over at http://help.netscape.com/nuggies/." [MozillaOrg02][197]

Durch einen zentralen und projekteigenen Newsgroup-Server werden die Newsgroups bereitgestellt und zusätzlich über das traditionelle Usenet kommuniziert: "All of the mozilla.org newsgroups are served by the machine news://news.mozilla.org/. However, the groups are also distributed over Usenet […]" [MozillaOrg02][197]

**NewsBot**

"NewsBot is a tool to create a user generated list of pointers to important discussions and announcements in the newsgroups." [MozillaOrg02][198]. NewsBot wird verwendet, um auf die wichtigsten Artikel und Threads der Newsgroups zu verlinken und dadurch den Prozess der Recherche nach relevanten Informationen zu beschleunigen: "Since not everyone has a chance to keep up with all the mozilla news groups, newsbot is here to collect pointers to some of the more important announcements, discussions, and goings-on." [MozillaOrg02][199]

#### 4.7.1.3 Internet Relay Chat - IRC

Echtzeitkommunikation stellt ebenfalls eine wichtige Anforderung an die projektweite Kommunikation dar. Dies wird durch die Etablierung geeigneter IRC-Kommunikationskanäle realisiert, welche basierend auf verschiedenen fachspezifischen Kategorien aggregiert und umgesetzt werden: "IRC is a real time communication system which organizes participants in channels." [ReMa02]. Für die verteilte Projektarbeit stellt IRC eine sehr sinnvolle Ressource dar, da dieses Werkzeug verschiedene Merkmale aufweist, die es als ein

197 http://www.mozilla.org/community.html
198 http://www.mozilla.org/news.html
199 http://www.mozilla.org/newsbot/

adäquates Substitut für direkte persönliche oder telefonische Korrespondenz charakterisieren. Zum einen ermöglicht es die Kommunikation in Echtzeit und zum anderen die Adressierung eines einzelnen Anwenders oder einer einzelnen Anwendergruppe: "It presents an interesting scenario for a distributed development environment: although it is real time, it allows for selective communication and thus can be used as a reasonable substitute for a telephone when technical matters are to be discussed." [ReMa02]

Die zentralen IRC-Channel des Projekts wurden entsprechend ihrer Inhalte und der damit assoziierten Zielgruppe eingeteilt: "For example, there is a channel for technical discussions relating to the browser (#mozilla) but also a channel for volunteers performing smoke tests (#smoketest)." [ReMa02]

**IRC-Bots**

Zur Ergänzung wurden Applikationen entwickelt, die als virtuelle IRC-Anwender agieren, welche die Kommunikation nach Schlüsselwörtern durchsuchen und gegebenenfalls weiterführende Informationen zu den entsprechenden Themen zur Verfügung stellen: "A number of 'bots', which are programs that present themselves as IRC users, but which respond to specific commands, are hosted and provide convenient reference material when needed." Zum Beispiel existiert nach [ReMa02] ein Bot, der nach dem Schlüsselwort 'Bug' recherchiert und bei erfolgreicher Suche entsprechende Informationen und einen Link auf Bugzilla zur Kommunikation beiträgt.

### 4.7.2 Kollaborationswerkzeuge

Zur Unterstützung der kollaborativen, dezentralen und evolutionären Systementwicklung instrumentalisiert das Mozilla-Projekt verschiedene Werkzeuge, die z.T. auch projektintern entwickelt wurden und werden.

Diese Werkzeuge stellen die zentral bereitgestellten Hilfsmittel zur Koordination und Organisation der verteilten und freien Softwareentwickler dar und determinieren den Entwicklungsprozess durch ihre spezifischen Merkmale.

Ein wichtiges Charakteristika dieser Werkzeuge ist ihre Erreichbarkeit und Verwendbarkeit über HTTP und Standard Webbrowser: "Most of the Mozilla software tools are accessed through a Web browser. The easy access provided by the Web is suited to the needs of the Mozilla community, and underscores the importance of supporting a distributed software development process." [ReMa02]

Alle Werkzeuge, die zu diesem Zweck eigenständig durch die Mozilla-Organisation bzw. die Community des Projekts entwickelt oder adaptiert

wurden, sind unter einer OSS-Softwarelizenz erfügbar: "All tools are available under an Open Source license, so they can be freely used in any (private or OSS) project." [ReMa02]

#### 4.7.2.1 Konfigurationsmanagement - CVS

"The Mozilla CVS repository is the version control system that tracks code changes and allows individuals to check-in or checkout code." [CaLeCh02].

Der gesamte Entwicklungsquellcode des Mozilla-Projekts wird in einem CVS-Archiv verwaltet und steht auf diesem Wege auch den interessierten Entwicklern zur Verfügung: "In addition to the other methods, the Mozilla source code is available via CVS." [MozillaOrg02][200]

CVS (Concurrent Versions System) ist ein weit verbreitetes Werkzeug für die Verwaltung von verschiedenen Quellcode-Varianten und das damit verbundene Konfigurationsmanagement. Damit repräsentiert CVS die fundamentale Basis zur verteilten Software-Entwicklung basierend auf einer einheitlichen Quellcodebasis. CVS identifiziert im Rahmen der Integration von Quellcodemodifikationen die Abhängigkeiten zwischen den verschiedenen Modifikationen und dem existierenden Quellcode und stellt sicher, dass keine miteinander konkurriende Änderungen basierend auf demselben Ausgangsquellcode integriert werden können. Außerdem erkennt und signalisiert CVS einen Konflikt, sobald Quellcode integriert werden soll, dessen Ausgangsbasis oder dessen abhängige Softwaremodule sich in der Zwischenzeit geändert haben.

[ReMa02] stellt die folgenden Funktionalitäten von CVS in den Mittelpunkt:

- Zentrales Repository: Client/Server-Ansatz stellt über zentralen Server Quellcode zur Verfügung
- Versionskontrolle
- Branches: Verwaltung multipler Quellcodeverzeichnissbäume, z.B. für einen stabilen und einen Entwicklungspfad
- Netzwerkfähigkeit
- Kollaborative Entwicklung: CVS ermöglicht die simultane Bearbeitung eines speziellen Quellcode-Artefakts durch mehrere Benutzer

Die kollaborative Entwicklung des Quellcodes wird von CVS primär dadurch ermöglicht, dass ausgecheckte Quellcode-Dateien nicht für den weiteren Zugriff gesperrt werden, sondern weiterhin von anderen Entwicklern bearbeitet werden können. Etwaige Konflikte bei der späteren Integration von durch

200 http://www.mozilla.org/cvs.html

verschiedene Entwickler vorgenommenen Änderungen, werden von CVS erkannt, signalisiert und die Integration verhindert: "CVS does not place locks upon code 'checked out'. Instead, each developer checks out a local copy of the code, changes it, and when ready, submits it back to the repository. Conflicts in changes are handled client-side, and not in the repository." [ReMa02]

CVS unterstützt dadurch die spezifischen Anforderungen der OSS-Entwicklung und ist ein im OSS-Kontext sehr weit verbreitetes Werkzeug: "It is by far the most commonly used version control system in OSS projects, and it models very closely the code integration practices in them." [ReMa02]

#### 4.7.2.2 Bug Tracking - Bugzilla

Ein zentrales Werkzeug zur Erfassung, Verwaltung und Bearbeitung von Softwarefehlern ist die Bugzilla-Datenbank, die nach [ReMa02] durch die Mozilla-Entwickler eigenständig entwickelt und weitgehend in Perl implementiert wurde. Bugzilla ermöglicht die zentrale Verwaltung und Erfassung von Fehlerberichten über ein Web-Interface: "Bugzilla is where developers can report bugs in the Mozilla source releases, and browse an online database of already-reported bugs." [MozillaOrg02][201]. Außerdem schafft Bugzilla durch verschiedene Mechanismen die Voraussetzung für eine dezentrale und verteilte Bearbeitung dieser Fehlerberichte und unterstützt auf sehr adäquate Weise die Entwicklung im Mozilla-Kontext: "Bugzilla [...] acts as a central hub for communication and code review among community members." [ReMa02]. Bugzilla ist Open Source Software und basiert auf der Mozilla Public License: "Its source code has been released under the Mozilla Public License." [MozillaOrg02][202]. Dadurch wurde die Voraussetzung dafür geschaffen, dass Mozilla inzwischen auch in verschiedenen anderen OSS-Projekten eingesetzt wird: "It is used in a large number of organizations and companies outside of the Mozilla Project, including Netscape, NASA, Conectiva, Redhat and GNOME." [ReMa02]

Außerdem definiert [ReMa02] die folgenden Eigenschaften bzw. Funktionalitäten des Systems:

- Dedizierte Rechtevergabe über Anwender Accounts
- Metainformationen der Bug Reports über spezielle Attribute definierbar
- Kommunikationsmedium durch die Generierbarkeit von Kommentaren
- Artefakt-Container
- Dedizierte Such-Mechanismen

201 http://www.mozilla.org/tools.html
202 http://www.mozilla.org/bugs/

- E-Mail Integration

Da die in Bugzilla verwalteten Artefakte, respektive die Bug Reports, über verschiedene Metadaten beschrieben werden, wird eine strukturierte Verwaltung der Bug Reports über ihren gesamten Lebenszyklus hinweg ermöglicht: "[...] bugs have properties that match very closely Mozilla process requirements, and allow for fine control over responsibility, scheduling, dependencies and status." [ReMa02]. Zudem kann das Bugzilla-System auch als Kommunikationswerkzeug betrachtet werden, da es zur Kommunikation verschiedener Artefakte wie z.B. Patches oder zur Diskussion der Patches und Bug Reports verwendet wird, indem die verschiedenen Metadaten als Container für Kommunikationsbeiträge, Kommentare oder Artefakte verwendet werden: "[...] because each bug keeps a sequential list of user comments, Bugzilla works very well as a focused discussion forum. Features like automatic hyperlinking and commenting are very convenient. [...] bugs can have 'attachments', which are files uploaded by the user and linked to a specific bug. Most of the attachments are patches for bugs, but they can also be test cases, screen captures and specifications." [ReMa02]

Die Definition der Metadaten zur strukturierten Beschreibung eines Bug Reports ermöglicht außerdem die umfassende, attributbasierte Recherche in den verwalteten Artefakten: "To support queries on this database, there is a search function that allows one to specify what attributes define the bug being searched for." [ReMa02]

Durch die Integrierbarkeit von E-Mail-Funktionalitäten können verschiedene Automatismen zur Information aller in die Bearbeitung eines spezifischen Artefakts involvierter Akteure konfiguriert werden: "Bugzilla changes are mailed to all parties that are registered with the bug, allowing people to be notified of requests and the bug's progress." [ReMa02]

In [ReMa02] werden die folgenden, primären Einsatzgebiete von Bugzilla beschrieben:

- Entwickler kommunizieren Bug Reports bzw. Patches und fordern Reviews an bzw. führen diese durch
- Akteure der Quality Assurance berichten Fehler, kontrollieren den Bearbeitungsfortschritt und führen den Prozess des 'Bug Triage' aus
- Mitglieder der Mozilla Organisation identifizieren Entwickler und steuern den Fortschritt bezüglich der Entwicklungsaktivitäten von zentraler Bedeutung

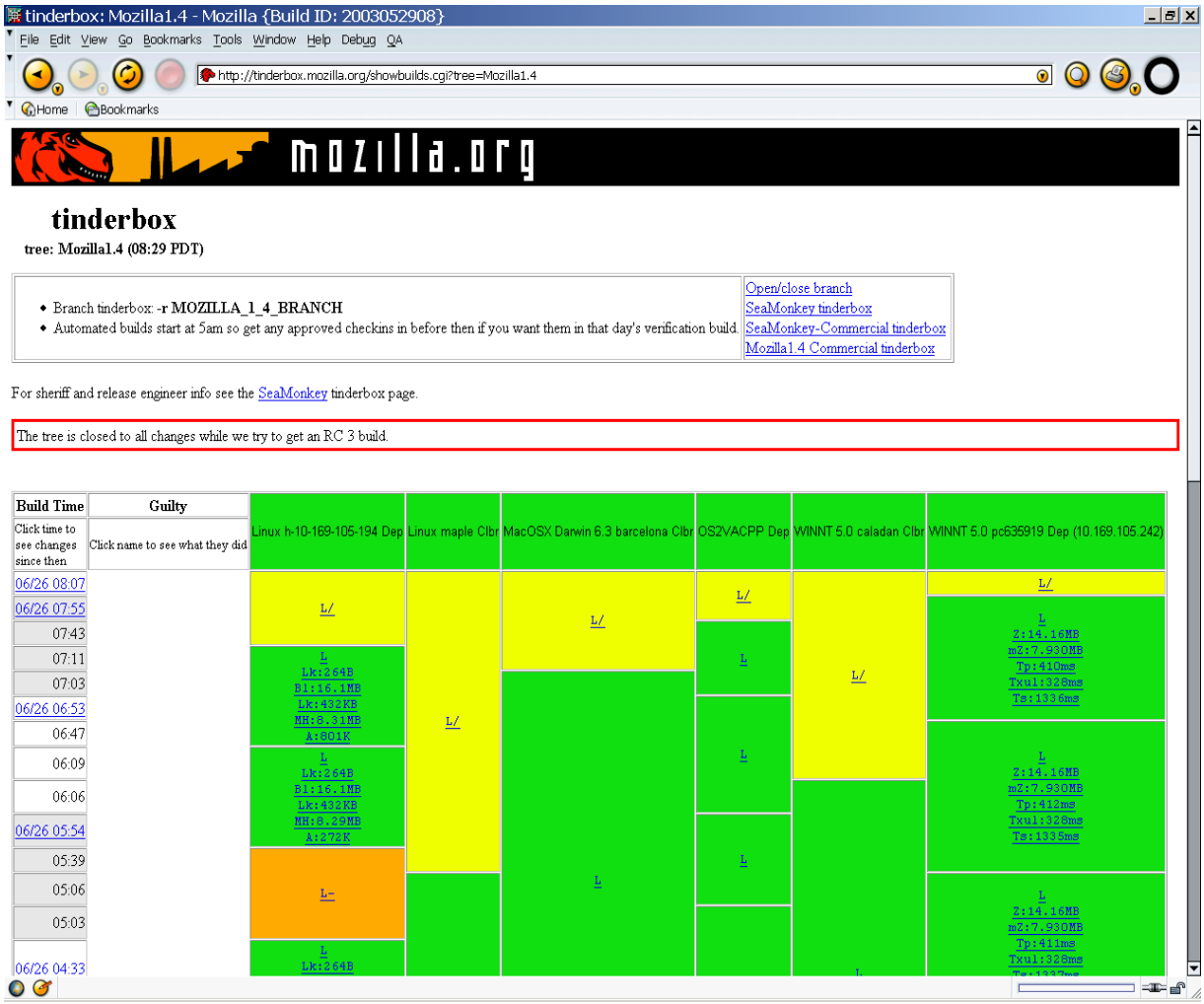
#### 4.7.2.3 Tinderbox

Tinderbox ist ein Werkzeug, welches es erlaubt, die plattformübergreifende Kompilation von Quellcode und das Testen der so erstellten Builds zu steuern und zu unterstützen: "Tinderbox is an automated system that tracks compilations and tests." [ReMa02]. Es wurde und wird wie auch Bugzilla durch die Mitglieder der Mozilla-Community entwickelt.

Es basiert auf einem Client/Server-Ansatz, bei dem verschiedene verteilte Rechner der Mozilla Organisation kontinuierlich den aktuellen Quellcode insbesondere der neuesten Änderungen kompilieren und die dabei erzeugten Erfolgs- oder Fehlermeldungen an den Server senden, welcher diese aufbereitet und über eine webbasierte Benutzerschnittstelle zur Verfügung stellt: "It is a client-server tool: client machines of various architectures and operating systems form a cluster. These machines' task is to compile, test and report results back to the Tinderbox server." [ReMa02]. Der primäre Nutzen für die Entwicklergemeinde entsteht dabei neben der Ausführung von plattformübergreifenden Kompilierungsprozessen durch eine zentrale, webbasierte Benutzerschnittstelle, welche diese Ergebnisse an die Community transportiert und somit den Prozess der Erstellung der Builds mit mehr Transparenz versorgt und den Anwendern einen Einblick in den Status der aktuellen Quellcodeänderungen ermöglicht: "The main user-visible feature [...] is a page displaying the compilation results associated with the individual machines in the cluster, and the code changes that were integrated to the repository at the same time." [ReMa02]. Dabei wird für jeden involvierten Client bzw. die jeweilige Plattform eine Spalte generiert und die durchgeführten Kompilationen und die dabei generierten Nachrichten ausgegeben. Die vertikale Achse repräsentiert die Zeit, in welcher der Kompilationsvorgang ausgeführt wurde, wobei der jeweils aktuellste Prozess in der obersten Zeile dargestellt wird. Wie auch in [MozillaOrg02][203] dargestellt wird, signalisiert die Farbe Grün eine erfolgreiche Erstellung, Rot signalisiert einen Fehler, Gelb repräsentiert einen noch aktiven Kompilationsprozess und Orange signalisiert einen erfolgreichen Kompilationsprozess, der aber Warnmeldungen zurückgeliefert hat.

203 http://www.mozilla.org/quality/smoketests/faq.html

Bild 13 Mozilla: Screenshot einer Tinderbox Ausgabe (aus [ReMa02])




Außerdem wird für jeden integrierten Quellcode der verantwortliche Entwickler in der Spalte "Guilty" angegeben: "This way, it is trivial to find out what code change was committed by which developer, and what bug it fixed." [ReMa02]. Dadurch ist es möglich, Fehler oder Inkompatibilitäten zu bestimmten Plattformen, welche bei der Kompilation entdeckt wurden, sofort durch die verantwortlichen Entwickler beheben zu lassen oder das jeweilige Patch vollständig aus dem Quellcode zu entfernen. Dadurch ist es nicht mehr notwendig, dass Entwickler oder Modul Owner vor der Integration des Quellcodes dessen plattformübergreifende Verwendbarkeit sicherstellen, da dies zentral durch Tinderbox geprüft wird: "It also allows code that is not cross-platform compatible to be identified and fixed, removing the need for each developer to guarantee it works on all architectures and operating systems. [ReMa02]

Weiterhin ist es möglich, die verschiedenen Plattformen, auf denen die Kompilationen ausgeführt werden, um weitere zu ergänzen, indem das existierende Tinderbox Client/Server System um einen weiteren Client mit der entsprechenden Plattform ergänzt wird. Die URL: http://www.mozilla.org/tinderbox.html verweist auf weiterführende Informationen über Tinderbox.

#### 4.7.2.4 LXR

LXR steht für Linux Cross Reference und ist ein Quellcode-Browser, der im Linux-Kontext entwickelt wurde und es ermöglicht, den aktuellen Quellcode webbasiert darzustellen. Dabei nutzt LXR die inhärenten Möglichkeiten von Hypertext konsequent zur Verlinkung von kohärenten Informationen: "You can browse the up-to-the-minute latest version of the source code online, through a massively-hyperlinked source code browser that lets you cross-reference function and variable names." [MozillaOrg02][204]

Weiterführende Informationen über LXR enthält die URL: http://lxr.mozilla.org/.

#### 4.7.2.5 Bonsai

Bonsai ist eine webbasierte Benutzerschnittstelle zur Durchführung von Abfragen auf das CVS-Archiv (vgl. [ReMa02]).

Es dient der Steigerung der Transparenz des aktuellen Quellcodes und der Kontrolle des CVS-Archivs und wurde ebenfalls intern durch die Mozilla-Community entwickelt: "It is our our home-grown web-based tool for watching the up-to-the-minute goings-on in our CVS repository (viewing checkins and log messages, reading diffs, etc.)" [MozillaOrg02][204]

Bonsai ermöglicht es den Entwicklern, nach den aktuellsten Änderungen am Quellcode zu recherchieren und fügt zu den Checkins automatisch Hyperlinks auf den Bug Report hinzu, der durch diesen Patch bearbeitet wurde. Außerdem stellt es nach [ReMa02] eine Schnittstelle zur Verfügung um Unterschiede zwischen verschiedenen Quellcodeversionen aufzuzeigen und ermöglicht die Identifikation von verantwortlichen Entwicklern für ein spezifisches Quellcodeartefakt: "[...] you can: get a list of checkins, see what checkins have been made by a given person, or on a given CVS branch, or in a particular time period. It also includes tools for looking at checkin logs (and comments); doing diffs between various versions of a file; and finding out which person is responsible for changing a particular line of code [...]" [MozillaOrg02][205].

204 http://www.mozilla.org/tools.html

Weiterhin ist Bonsai in der Lage, die aktuellen Quellcodeverzeichnissbäume grafisch zu visualisieren: "Bonsai ... provides a recently added graphing mechanism based on the cvsgraph tool, which depicts the different versions of files on the project repository's branches... " [ReMa02].

Dadurch ermöglicht Bonsai auch Statistiken und Analysen bezüglich der Quellcodeentwicklung und der Evolution des Quellcodes zu erstellen.

Weiterführende Informationen bezüglich Bonsai können der Website dieses Mozilla-Subprojekts entnommen werden: http://www.mozilla.org/bonsai.html

### 4.7.3 Entwicklungswerkzeuge

Die Werkzeuge, welche individuell durch die Entwickler instrumentalisiert werden, basieren auf keinen zentralen Vorgaben und determinieren den Entwicklungsprozess auch nicht in dem Maße, wie dies durch die zentral verwalteten und projektweit verwendeten Werkzeuge geschieht: "Since the work done in the Mozilla project is distributed and each person utilizes the resources that are available to them, there are no standards or recommended tools for development of code or in the case of QA, test cases." [CaLeCh02]. Daher können die identifizierten Entwicklungswerkzeuge nur auf einem sehr hohen Abstraktionsniveau dargestellt werden.

#### Text Editoren

Elementares Werkzeug zur Erstellung und Modifikation von Quellcode durch die Entwickler sind die jeweils verfügbaren Text-Editoren: "In QA, volunteers and QA personnel utilize text editors (e.g. emacs) of their choice/preferences to create different test cases." [CaLeCh02]. Da auf den verschiedenen Plattformen verschiedene Text Editoren verfügbar sind, kommen z.B. im Unix-Kontext emacs oder vi, im Windows-Kontext Notepad oder im Macintosh-Umfeld SimpleText zur Editierung von Quellcode oder Test Cases zum Einsatz.

#### Compiler

Zur lokalen Kompilation der erstellten Quellcode-Änderungen werden die jeweils verfügbaren Compiler verwendet, wobei die Mozilla Organisation hierfür die Verwendung des GNU-Compilers empfiehlt: "Tools in the code development processes include text editors of choice such as emacs or the GNU compiler collection." [CaLeCh02]

205 http://www.mozilla.org/bonsai.html

### Diff

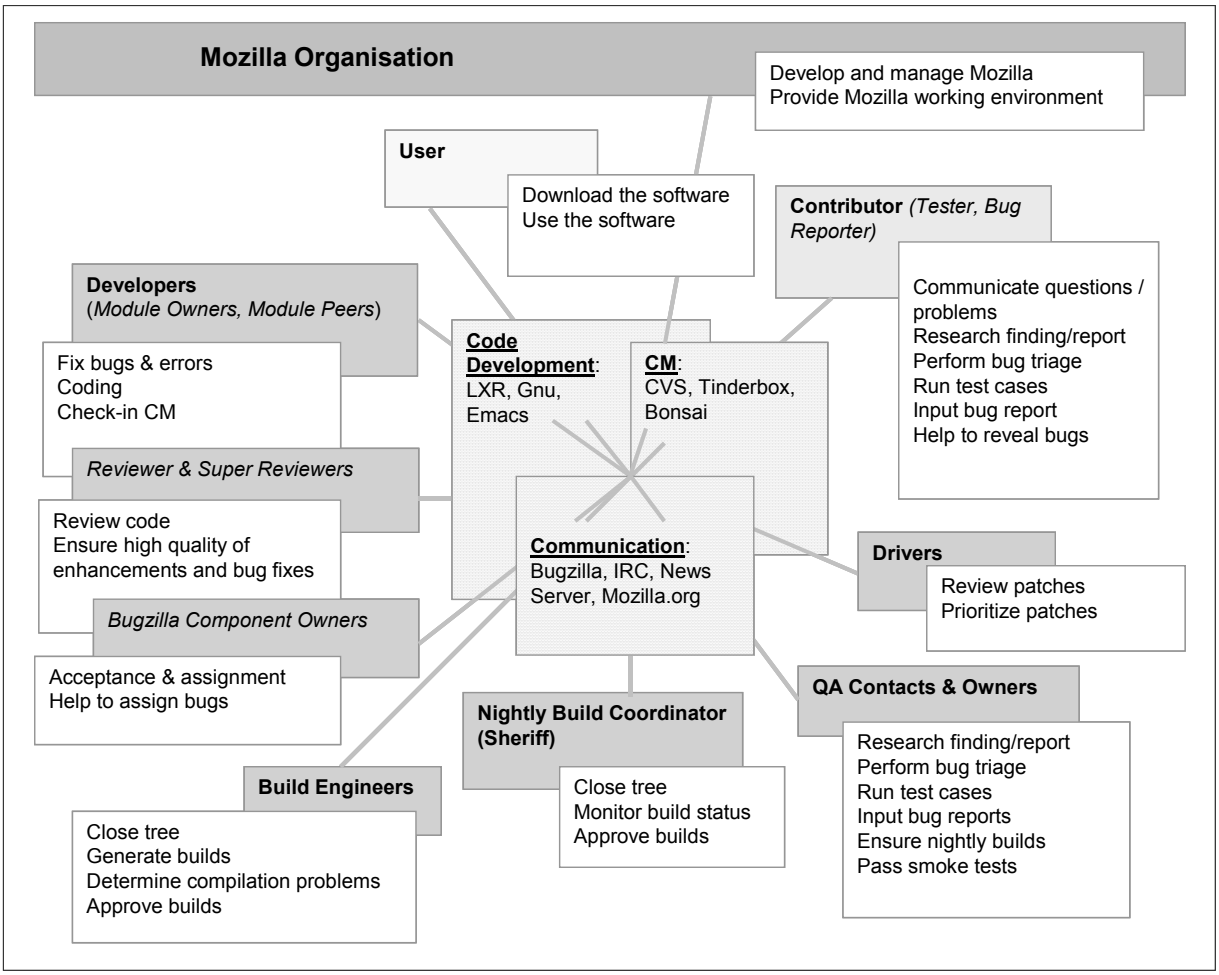
Zur Erzeugung von lokal erstellten Patches werden die Quellcode-Neuerungen mit dem Werkzeug Diff extrahiert und können somit in das zentrale Quellcode-Repository integriert oder in einem standardisierten Format kommuniziert werden.

## 4.8 Gesamtheitliche Prozessbetrachtung

Folgende grafik orientiert sich an [CaLeCh02] und ermöglicht eine gesamtheitliche Perspektive auf die Entwicklungsprozesse im Mozilla-Kontext und die involvierten Rollen und Werkzeuge. Die Abkürzung "CM" repräsentiert hierbei das Configuration Management.

Im Rahmen dieser Fallstudie wurde das ursprüngliche Modell aus [CaLeCh02] etwas modifiziert und verschiedene Änderungen vor allem bezüglich der Rollen und Teilprozesse vorgenommen, um das Modell dem Verständnis, das sich im Laufe der Fallstudie entwickelt hat, anzupassen.

Bild 14 Mozilla: Gesamtüberblick Mozilla-Entwicklungsprozess (nach [CaLeCh02])



# Literatur


[AcFe02] Silvia T. Acuna, Xavier Ferré: "Software Process Modelling"; URL: www.unse.edu.ar/congres/jornadas/silvac/documentos/congres1.pdfl; Abfrage: 28.08.2002

[ApacheOrg02] Apache Software Foundation: Webseite des Apache Projekts; URL: http://www.apache.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[ApWeekCom01] Apache Week Journal; URL: http://www.apacheweek.com; Abfrage: 2001 (verschiedene Dokumente)

[AtGaGe02] 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: 22.07.2002

[BitkprCom02] Produktwebsite Bitkeeper; URL: http://www.bitkeeper.com/; Abfrage 2002/2003 (verschiedene Dokumente)

[BoHoBr99] Ivan T. Bowman, Richard C. Holt, Neil V. Brewster: "Linux as a Case Study: Its Extracted Software Architecture"; Proceedings of the 21st Intl. Conference on Software Engineering (ICSE-21); Los Angeles, CA; May 1999

[CaLaMo03] Andrea Capiluppi, Patricia Lago, Maurizio Morisio: "Evidences in the Evolution of OS projects through Changelog Analyses"; Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03; Portland, USA; May 2003

[CaLeCh02] Bryce Carder, Baolinh Le, Zhaoqi Chen: "The Process of Quality Assurance in the Mozilla Project's Release Cycle"; URL: http://www.ics.uci.edu/~acarder/225/report.htm; Verfasst: 2002; Abfrage: 22.10.2002

[Cheng01] Tao Cheng: "Writing Global Customization Friendly Code"; URL: http://www.mozilla.org/projects/l10n/customizable-code.html; Verfasst: 2001; Abfrage: 25.09.2002

[Cubranic01] Davor Cubranic: “Open Source Software Development”; URL: http://sern.ucalgary.ca/~maurer/ICSE99WS/Submissions/Cubranic/Cubranic.html; Verfasst: 2001; Abfrage: 26.03.2002

[CvsApOrg02] Apache Software Foundation: CVS Repository; URL: http://cvs.apache.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[DeWeJoGr99] Bert J. Dempsey, Debra Weiss, Paul Jones, Jane Greenberg: “A Quantitative Profile of a Community of Open Source Linux Developers”; URL: http://ils.unc.edu/ils/research/reports/TR-1999-05.pdf; Abfrage: 10.07.2002

[Dietze03] Stefan Dietze: Arbeitsentwurf zur Dissertation “Identifikation und Optimierung eines generalisierbaren Prozessmodells für evolutionäre Softwareentwicklung unter dem Paradigma von Open Source“; Universität Potsdam und Fraunhofer Institut für Software- und Systemtechnik; 2003

[Draxler02] Barbara Draxler: "Key Elements of an iterative Software Development Process"; URL: http://www.cosy.sbg.ac.at/~pmm/teaching/SS02/se/papers/0020865.pdf; Abfrage: 18.09.2002

[Ebling03] Andrew Ebling: "Linux Kernel Hacking HOWTO"; URL: http://www.kernelhacking.org/docs/kernelhacking-HOWTO.html.tar.gz; Aktuelle Version: 27.03.2003; Abfrage: 05.06.2003

[EiBa03] Brendan Eich, Mitchel Baker: "Mozilla Super Review"; URL: http://www.mozilla.org/hacking/reviewers.html; Aktuelle Version: 29.05.2003; Abfrage: 25.06.2003

[Eich03a] Brendan Eich: "Mozilla Development Roadmap"; URL: http://www.mozilla.org/roadmap/roadmap-05-Jun-2002.html; Aktuelle Version: 28.03.2003; Abfrage: 25.06.2003

[Eich03b] Brendan Eich: "Mozilla Development Roadmap"; URL: http://www.mozilla.org/roadmap.html; Aktuelle Version: 23.06.2003; Abfrage: 25.06.2003

[Fähnrich02] Klaus-Peter Fähnrich: Vorlesungsunterlagen Softwaretechnik, URL: http://ais.informatik.uni-leipzig.de/download/vorl_softw_tech/SE2.pdf; Abfrage: 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

[GaLaAr01] Christina Gacek, Tony Lawrie, Budi Arief: "The many meanings of Open Source"; URL: http://citeseer.nj.nec.com/485228.html; Verfasst: 2001; Abfrage: 25.06.2003

[GnuGpl91] Free Software Foundation, Inc: "GNU General Public License"; URL: http://www.gnu.org/licenses/gpl.txt; Verfasst: 1991; Abfrage: 25.06.2002

[Godfrey00b] Michael W. Godfrey: "Toward an Understanding of Software Evolution"; URL: http://plg.uwaterloo.ca/~migod/papers/evolution.pdf; Verfasst: 2000; Abfrage: 02.07.2002

[GoLe02] Michael W. Godfrey, Eric Lee: "Secrets from the Monster: Extracting Mozilla's Software Architecture"; URL: http://plg.uwaterloo.ca/~migod/papers/coset00.pdf; Abfrage: 12.011.02

[GoTu00] Michael W. Godfrey, Qiang Tu: "Evolution in Open Source software: A case study" in: Proceedings of the International Conference on Software Maintenance, 2000 (ICSM 2000); San Jose, CA; URL: http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf; Abfrage: 21.06.2002

[Gooch02] Richard Gooch: "Reporting Bugs for the Linux Kernel"; http://www.kernel.org/pub/linux/docs/lkml/reporting-bugs.html; Abfrage: 10.09.2002

[Grassmuck02] Volker Grassmuck: "Freie Software"; Bundeszentrale für politische Bildung; 2002

[GrTa99] Richard Gregory, Ladan Tahvildary: "Architectural Evolution: A Case Study of Apache"; URL: http://citeseer.nj.nec.com/431508.html; Verfasst: 1999; Abfrage: 15.10.2002

[HeNiHe02] Guido Hertel, Sven Niedner, Stefanie Herrmann: "Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel"; URL: http://www.psychologie.uni-kiel.de/linux-study/writeup.html; Abfrage: 08.06.2002

[HttpdApOrg02] Apache Software Foundation: HTTP Server Project; URL: http://httpd.apache.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[IssuesApOrg02] Apache Software Foundation: Issues web site; URL: http://issues.apache.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[JakartaApOrg02] Apache Software Foundation: Apache Jakarta Project; URL: http://jakarta.apache.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[Jordan01] John Jordan: "Detecting Dominant Designs: Applying the theory in Network Economies" (2001); URL: http://citeseer.nj.nec.com/rd/29566629%2C563111%2C1%2C0.25%2CDownload/http://citeseer.nj.nec.com/cache/papers/cs/27173/http:zSzzSzopensource.mit.eduzSzpaperszSzjordan.pdf/detecting-dominant-designs-applying.pdf; Verfasst: 2001; Ontario, Canada; Abfrage: 22.08.2002

[Kernel02] The Kernelnewbies Project: Linux Kernel Maintainers List; URL: http://kernelnewbies.org/maintainers/; Abfrage: 10.06.2002

[KernelOrg02] Kernel.Org Organization, Inc: "The Linux Kernel Archives"; URL: http://www.kernel.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[Koch00] Stefan Koch: "Entwicklung von Open Source und kommerzieller Software: Unterschiede und Gemeinsamkeiten"; URL: http://wwwai.wu-wien.ac.at/~koch/forschung/sw-eng/vg01-folien.pdf; Verfasst: 2000; Abfrage: 12.06.2002

[Koch01] Stefan Koch: "Produktivität in der Software Entwicklung - Spezialfall Open Source: Produktivität in Open Source Projekten"; URL: http://wwwai.wu-wien.ac.at/~koch/lehre/oss-ws-01/oss_prod/oss_prod-slides.pdf; Verfasst: 2001; Abfrage: 10.11.2002

[LDP02] The Linux Documentation Project: Homepage; URL: http://www.tldp.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[LinuxOrg02] Linux Online, Inc.: “Linux Online!”; URL: http://www.linux.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[LkmlFaq02] Kernel.Org Organization, Inc: “The linux-kernel mailing list FAQ “; URL: http://www.kernel.org/pub/linux/docs/lkml/#s1-9; Abfrage: 2002/2003 (verschiedene Dokumente)

[Ludewig02] Jochen Ludewig: "Modelle im Software Engineering - eine Einführung und Kritik", URL: http://www.iste.uni-stuttgart.de/se/publications/download/Modelle.pdf; Abfrage: 17.09.02

[MaBa03a] Gregorio Robles-Martínez, Jesús M. González-Barahona: "Studying the evolution of libre software projects using publicly available data"; Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE'03; Portland, USA; May 2003

[MaBa03b] Gregorio Robles-Martínez, Jesús M. González-Barahona: "Unmounting the 'code-gods' assumption"; Proceedings of the Workshop 'Making Free/Open-Source Software Work Better' at the XP-2003; Genua, Italien; May 2003

[MoFiHe00] Audris Mockus, Roy Fielding, James Herbsleb: "A Case Study of Open Source Software Development: The Apache Server"; Proceedings of the 22nd International Conference on Software Engineering, 2000 (ICSE 2000); URL: http://citeseer.nj.nec.com/mockus00case.html; Abfrage: 22.05.2002

[MozillaOrg02] The Mozilla Organisation: “mozilla.org”; URL: http://www.mozilla.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[NoSt98] Ludvig A. Norin, Fredrick Stöckl: "Open-Source Software Development Methodology"; URL: http://www.ludd.luth.se/users/no/os_meth.html; Verfasst: 1998; Abfrage: 05.06.2002

[Nüttgens00] Markus Nüttgens: "Open Source: Anwendungsentwicklung im Internetzeitalter"; URL: http://www.iwi.uni-sb.de/nuettgens; Verfasst: 2000; Abfrage: 10.06.2002

[OSOrg02] Open Source Initiative: Homepage; URL: http://opensource.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[Pavlicek00] Russel C. Pavlicek : "Embracing Insanity - Open Source Software Development"; Sams Publishing; September 2000

[Raymond01a] Eric Steven Raymond: "The Cathedral and the Bazar"; O'Reilly UK; 2001

[Raymond01b] Eric Steven Raymond: "How To Ask Questions The Smart Way"; URL: http://www.tuxedo.org/~esr/faqs/smart-questions.html; Verfasst: 2001; Abfrage: 09.11.2002

[ReMa02] Christian Robottom Reis, Renata Pontin de Mattos Fortes: "An Overview of the Software Engineering Process and Tools in the Mozilla Project"; URL: http://www.async.com.br/~kiko/papers/mozse.pdf; Verfasst: 2002; Abfrage: 13.10.2002

[Rosenberg00] Donald K. Rosenberg: "Open Source - The Unauthorized White Papers"; IDG Books Worldwide, Inc.; 2000

[Rusling02] David A. Rusling: "The Linux Kernel Book"; URL: http://www.tldp.org/LDP/tlk/tlk.html; Abfrage: 12.07.2002

[Russel00] Paul Russel: "Unreliable Guide To Hacking The Linux Kernel"; URL: http://www.kernelnewbies.org/documents/kdoc/kernel-hacking/lk-hacking-guide.html bzw. http://www.kernelnewbies.org/documents/kdoc/kernel-hacking.pdf; Verfasst: 2000; Abfrage: 03.08.2002

[Salzmann01] Peter Salzmann: "The Linux Kernel Module Programming Guide"; URL: http://www.dirac.org/linux/writing/lgh/linux-gamers-howto.ps; Verfasst: 2001; Abfrage: 03.08.2002

[SambaOrg02] Samba Organisation: “The Samba Web Pages”; URL: http://samba.org; Abfrage: 2002/2003 (verschiedene Dokumente)

[Sandred01] Jan Sandred: "Managing Open Source Projects"; John Wiley & Sons, Inc.; 2001

[Scacchi98] Walt Scacchi: "Comparative Case Analysis for Understanding Software Processes"; URL: http://www.usc.edu/dept/ATRIUM/Papers/New/CCA-Draft.html; Verfasst: 1998; Abfrage: 10.07.2002

[Scacchi01] Walt Scacchi: "ITR/SOC: Understanding Open Software Communities, Processes and Practices: A Socio-Technical Perspective"; Research Proposal; URL: URL: http://www.ics.uci.edu/~wscacchi/Software-Process/Readings/NSF-Open-Software-Proposal.pdf, Abfrage: 20.08.2002

[Scacchi02] Walt Scachi: "Course Materials for ICS 225: Software Process"; URL: http://www.ics.uci.edu/~wscacchi/Software-Process/; Verfasst: 2002; Abfrage: 12.07.2002

[Stallman02] Richard Stallman: "Free Software und Free Manuals"; URL: http://www.fsf.org/philosophy/free-doc.html; Verfasst: 2002; Abfrage: 05.06.2002

[StDe02] Christian Strobel, Kai Degen: "Beschreibung eines UML-Metamodells"; URL: http://www.informatik.uni-essen.de/Lehre/Material/fcbse/studvor/Beschreibung_des_UML_Metamodells_StrobeDegen_.ppt; Abfrage: 02.05.2002

[Tatham02] Simon Tatham: "How to Report Bugs Effectively"; URL: http://www.chiark.greenend.org.uk/~sgtatham/bugs.html; Verfasst/Updated: 2002; Abfrage: 11.08.2002

[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"; URL: http://plg.uwaterloo.ca/~migod/papers/iwpc00.pdf; Verfasst: 2000; Abfrage: 11.08.2002

[Tuomi00] Ilkka Tuomi: "Internet, Innovation and Open Source: Actors in the Network"; URL: http://opensource.mit.edu/papers/Ilkka%20Tuomi%20-%20Actors%20in%20the%20Network.pdf; Verfasst: 2000; Abfrage: 22.06.2002

[Vepstas02] Linas Vepstas: "Call Center, Bug Tracking and Project Management Tools for Linux"; URL: http://linas.org/linux/pm.html; Verfasst/Updated: 2002; Abfrage: 10.06.2002

[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

[Weber00] Steven Weber: "The Political Economy of Open Source Software"; URL: http://e-conomy.berkeley.edu/publications/wp/wp140.pdf; Verfasst: 2000; Abfrage: 17.04.2002

[Williams98] David Williams: "The C++-Portability Guide"; URL: http://www.mozilla.org/hacking/portable-cpp.html; Verfasst: 1998; Abfrage: 02.07.2002