Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Requirements Engineering in Open Source Software-Projekten

Requirements Engineering in Open Source Software-Projects

Bachelorarbeit

im Studiengang Informatik

von

Letian Wang

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. rer. nat. Jil Ann-Christin Klünder
Betreuer: Alexander Specht

Hannover, 22.07.2024

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 22.07.2024

Letian Wang

Zusammenfassung

Der Entwicklungsprozess von Open-Source-Softwareprojekten ist dezentral, offen und wird von Freiwilligen getragen. Dies führt zu einer gewissen Inkompatibilität mit traditionellen Requirements-Engineering-Ansätzen, die auf eine zentrale Steuerung und formale Anforderungen ausgerichtet sind.

Die vorliegende Bachelorarbeit verfolgt das Ziel, die Praktiken des Requirements Engineering in Open-Source-Softwareprojekten zu erfassen, zu bewerten, deren spezifische Herausforderungen zu identifizieren und ihre Umsetzbarkeit zu diskutieren.

Die primäre Forschungsmethodik basierte auf einer Literaturrecherche, der Durchführung von Interviews mit Entwicklern sowie einer Fallstudie zu Firefox, einer etablierten Open-Source-Software.

Auf Basis der zuvor dargelegten Erkenntnisse wurde ein Modell zur systematischen Analyse der Leistungsfähigkeit des Requirements Engineering in der Open-Source-Entwicklung entwickelt. Das Modell analysiert den Ansatz des Requirements Engineering in der traditionellen Softwareentwicklung und seine Nützlichkeit in der Open-Source-Softwareentwicklung in Bezug auf die drei Phasen der Softwareentwicklung: vor, während und nach der Entwicklung.

Die durchgeführte Analyse kommt zu dem Schluss, dass der Zweck des Requirements Engineering in Open-Source-Softwareprojekten zwar teilweise derselbe ist wie in der traditionellen Softwareentwicklung, dass sich jedoch die Vorgehensweise und die Durchführung des Requirements Engineering in Open-Source-Softwareprojekten deutlich von der in traditionellen Softwareprojekten unterscheidet. Vor dem Hintergrund dieser Unterschiede haben sich bei der Entwicklung von Open-Source-Software Methoden herausgebildet, die einen adäquaten Umgang mit den identifizierten Unterschieden ermöglichen.

Abstract

Requirements Engineering in Open Source Software-Projects

The development process of open source software projects is characterised by a high degree of decentralisation, openness and volunteer-driven activity[1]. Consequently, traditional requirements engineering approaches may encounter significant challenges in adapting to this environment.

The objective of this Bachelor's thesis is to gain an understanding of and evaluate the practices of requirements engineering in open source software projects, identify the specific challenges they present and discuss their feasibility.

The primary research methodology comprised a study of the literature, interviews with developers and a case study of Firefox, a large existing open-source software.

A model was developed based on the above data to facilitate a systematic analysis of the performance of requirements engineering in open source development. The model analyses the requirements engineering approach in traditional software development and its usefulness in open source software development in relation to the three phases of software development: before, during and after development.

The analysis concludes that, while the purpose of requirements engineering in open source software is partly the same as in traditional software development, the approach and execution of requirements engineering in open source software projects differs significantly from that in traditional software projects. In view of these differences, methods have emerged in the development of open source software to address these differences.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 2
- 1.2 Problemstellung . . . 3
- 1.3 Lösungsansatz . . . 3
- 1.4 Struktur der Arbeit . . . 4

2 Grundlagen **5**
- 2.1 Requirements Engineering . . . 5
  - 2.1.1 Anforderungen . . . 6
  - 2.1.2 Elicitation . . . 7
  - 2.1.3 Analyse: Interpretation und Negotiation . . . 7
  - 2.1.4 Dokumentation . . . 8
  - 2.1.5 Validieren und Verifizieren . . . 8
  - 2.1.6 Change Management . . . 9
  - 2.1.7 Tracing . . . 9
- 2.2 Open Source Software . . . 9

3 Verwandte Arbeiten **11**

4 Interview **15**
- 4.1 Ziel des Interviews . . . 15
- 4.2 Interviewsleitfaden . . . 15
- 4.3 Ergebnisse des Interviews . . . 16

5 Analyse **19**
- 5.1 Beobachtung Firefox . . . 19
  - 5.1.1 Beschreibung . . . 19
  - 5.1.2 Erklärung . . . 21
- 5.2 Ergebnisse . . . 23
  - 5.2.1 Elicitation . . . 24
  - 5.2.2 Analyse . . . 24
  - 5.2.3 Dokumentation . . . 25
  - 5.2.4 Validieren & Verifizieren . . . 25
  - 5.2.5 Change Management . . . 26

5.2.6 Tracing 26
5.3 Zusammenfassung & Diskussion 26

6 Modell **29**
6.1 Vor dem Projekt 29
6.1.1 Anforderungen 29
6.1.2 Rechtliches, Lizenz und Datenschutz 32
6.1.3 Ziel und Motivation 34
6.1.4 Plannen 35
6.1.5 Kosten 35
6.2 Während des Projekts 36
6.2.1 Anwendungsfeld 36
6.2.2 In Team 36
6.2.3 Implementierung 37
6.2.4 Validierung & Verifizierung 38
6.3 Nachdem Projekt 38
6.3.1 Dokumentation 38
6.3.2 Sicherheit 39
6.3.3 Management 39
6.4 Allgemein & Eigenschaften 40
6.4.1 Herausforderung 40
6.4.2 Eigenschaften & Unterschiede 41
6.4.3 Vorteile und Nachteile 42
6.4.4 Kombinationsmöglichkeit & Kompatibilität 42

7 Diskussion **45**
7.1 Beantwortung von Forschungsfragen 45
7.2 Limitationen 47

8 Zusammenfassung und Ausblick **49**
8.1 Zusammenfassung 49
8.2 Ausblick 49

A Interviewsleitfaden **51**

Kapitel 1

Einleitung

> „Many organizations use open source infrastructure software such as Linux, and open source software (OSS) is generally considered a viable technology. Both professional and academic literature devote much attention to the OSS phenomenon[35]“ (Ven, Verelst, and Mannaert, 2008, p. 54).

Open Source Software Engineering ist eine neue Form der Softwareentwicklung im Rahmen der offenen Innovation. Die Inhaber der Software geben den Benutzern das Recht, die Software und ihren Quellcode für jeden Zweck zu verwenden, zu studieren, zu ändern und zu verbreiten.[10] Mit der zunehmenden Beliebtheit von Open Source Software (OSS) gewinnt der Open-Source-Softwareentwicklung allmählich an Aufmerksamkeit, Da Open-Source-Software in der Regel kostenlos und frei modifizierbar ist und von einer aktiven Gemeinschaft unterstützt wird.[28] Ein Vergleich mit dem traditionellen Software-Engineering, das in der Regel kommerziell ist und über engagierte Manager wie den Projekt Owner und das entsprechende SCRUM-Team verfügt, zeigt, dass das Open-Source-Software-Engineering sich grundlegend unterscheidet. Dies betrifft sowohl die Zusammensetzung der Mitarbeiter als auch die Arbeitsweise.[33] Das Requirements Engineering in der traditionellen Softwareentwicklung ist relativ ausgereift, jedoch besteht in der Open-Source-Softwareentwicklung noch viel Raum und Bedarf dafür.[7] Denn obwohl die Open-Source-Softwareentwicklung relativ frei und informell ist, müssen die Anforderungen dennoch analysiert und bearbeitet werden. Aufgrund der Informalität und des Fehlens eines vollständigen Prozesses wird das Requirements Engineering hier eine wichtige Rolle spielen.[33] Das ist ein Teil des Ziels des Requirements Engineering. Die Bedeutung des Requirements Engineering in der closed-Source-Softwareentwicklung bedarf keiner weiteren Erläuterung, aber bei Open-Source-Software spielt das Requirements Engineering aufgrund der Unterschiede im Entwicklungsteam, der Arbeitsweise, der Form der Anforderungen u.s.w. eine andere Rolle.[25, 1, 18, 19], Diese Unterschiede werfen viele Fragen auf, z. B.:

Welche Probleme ergeben sich aus diesen Unterschieden? Worauf sollten Sie beim Anforderungsmanagement für Open-Source-Software achten? In dieser Bachelorarbeit soll einen Überblick über die Unterschiede und Möglichkeiten des Requirements Engineering in Open Source Software geben.

1.1 Motivation

Aus der Sicht des Requirements Engineering gibt es viele generelle Unterschiede im Entwicklungsprozess von Open Source Software. Zum Beispiel gibt es keine speziellen Entwicklungsteams für Open Source Software im Vergleich mit dem Requirements Engineering. Sie diskutieren Anforderungen und Abhängigkeiten selten in kollektiven Meetings wie z.B. Scrum-Meetings, sondern nutzen häufiger Methoden wie z.B. Foren, um Fehler oder Verbesserungen, die in der Software auftreten, in nicht spezialisierten Statements zu diskutieren[25, 1]. Sowohl der Zweck als auch die Motivation der Entwickler ist hier grundlegend anders[20].

Die Frage der Sicherheit einer Software ist ebenfalls von Belang. Die Frage, ob Open-Source-Software in Computern genauso sicher sein wird wie kommerzielle Software, ist aktuell von großem Interesse.[30] Hier ist ein Beispiel für ein Sicherheitsproblem bei Open Source Software. XZ Utils ist eine freie Software-Sammlung zur verlustfreien Kommandozeilen-Datenkompression für Unix-ähnliche Betriebssysteme. Es bietet auch eine bessere Komprimierung. Am 29. März 2024 wurde von Microsoft-Ingenieuren in der Open-Source-Software XZ Utils[1] in einem Versionsupdate eine kritische Backdoor als Sicherheitslücke entdeckt. Hierbei handelt es sich um einen Supply-Chain-Angriff. Diese Backdoor ermöglicht es einem Angreifer, auf einem verwundbaren Server ohne vorherige Authentifizierung Befehle über SSH auszuführen[21]. Die Schwachstelle wurde in der IT-Bedrohungslage des Bundesamtes für Sicherheit in der Informationstechnik mit 3/Orange bewertet[6] sowie mit einem Base Score von 10.0 im CVSS 3.x, also der höchsten Gefahrenstufe.[22] Die für diese Operation verantwortliche Person handelte unter dem Namen „Jia Tan“ mit einer GitHub-UserID von „JiaT75“. Er übernahm die Kontrolle über das Projekt, nachdem er das Vertrauen der Projektadministratoren gewonnen hatte, und lud gefährlichen Code mit einer backdoor hoch, der zu dieser massiven Informationssicherheitskrise führte. [21]

Auch andere Bereiche des Requirements Engineering werden durch Open-Source-Software erschwert.[20] In einer empirischen Studie gaben ein Drittel der Open-Source-Entwickler an, dass die Entwicklung von Teilen des Requirements Engineering, wie beispielsweise die Negotiation und Priorisierung, schwierig sei.[20]

[1] `https://github.com/tukaani-project/xz`

Die Entwicklung der Open-Source-Softwaretechnik scheint schwierig zu sein. Es gibt jedoch nach wie vor eine Vielzahl an erfolgreiche Open-Source-Software- und Open-Source-Erweiterungslösungen, wie beispielsweise den bekannten Firefox-Browser. Es ist sehr weit verbreitet und wird als Beispiel für diese Arbeit verwendet.

1.2 Problemstellung

Aufgrund der signifikanten Unterschiede zwischen der Open-Source-Softwareentwicklung und der traditionellen Softwareentwicklung wird in dieser Arbeit durchführt, die jeweiligen Merkmale zu zusammenfassen und zu vergleichen, welche Teile des Requirements Engineering in der Open-Source-Softwareentwicklung nur schwer umzusetzen sind bzw. welche Teile durch eine andere, für Open-Source-Software charakteristische Praxisform ersetzt werden können. Im Zuge der Integration dieser Prioritäten werden die folgenden Forschungsfragen beantwortet:

RQ1: Was sind die Ursachen für die Unterschiede zwischen dem traditionellen Requirements Engineering und der Open-Source-Softwareentwicklung?

RQ2: Wie werden bei der OSS-Entwicklung Praktiken des Requirements Engineering behandelt, die sich schwer in OSS implementieren lassen?

RQ3: Welche spezifischen Methoden und Vorgehensweisen werden bei der Behandlung von Anforderungen in der Open-Source-Softwareentwicklung angewendet?

1.3 Lösungsansatz

Es wird ein Modell erstellt, indem man die einschlägige Literatur betrachtet und sich über das bestehende Requirements Engineering informiert. Das Modell analysiert die Unterschiede zwischen dem Requirements Engineering von Open-Source-Software und traditioneller Software in Bezug auf verschiedene Phasen der Softwareentwicklung und integriert sie in einem Durchführung, die Unterschiede zu erklären und die Gründe dafür zu erforschen. Es gibt auch eine Bewertung und Diskussion seiner Stärken und Schwächen. Das Modell wird mit der erfolgreicheren Open-Source-Software Firefox und Open Office verglichen, um die Ergebnisse der Diskussion zu testen und das Modell sowohl in Bezug auf den Prozess als auch auf das Ergebnis zu verbessern. Darüber hinaus werden auch kritische oder riskante Open-Source-Projekte beobachtet. Zu den konkreten Projekten gehören die „Corona Warn App" und die bereits erwähnten „xz Utils", auf deren Grundlage ein Interview geführt wurde. Die Erkenntnisse basieren auf einem anonymen Interview mit einem Back-End-Ingenieur eines großen Internet-Unternehmens, der seine Ansichten zu Open-Source-Software darlegte. Die

Aussagen des Entwicklers, der über eine Professionalität verfügt, werden in den entsprechenden Kapiteln präsentiert, um das bestehende Modell zu untermauern oder zu modifizieren.

1.4 Struktur der Arbeit

Diese Arbeit ist wie folgt strukturiert. In Kapitel 2 werden die Konzepte des Requirements Engineering und der Entwicklung von Open Source Software erläutert, und in Kapitel 3 werden verwandte Informationen aus der gelesenen Literatur erklärt. Kapitel 4 analysiert ein Interview mit einem anonymen Softwareentwickler über die Verwaltung von Open Source Software und Kapitel 5 analysiert die Open Source Software Firefox und ihre Verwaltungsplattform bugzilla. In Kapitel 6 wird ein Modell vorgestellt, das die Unterschiede zwischen Requirements Engineering in Open Source Software und traditioneller Software bei unterschiedlichen Entwicklungszeitplänen analysiert. Abschließend wird in Kapitel 7 eine Diskussion über Open Source Software geführt, in der die eigenen Gedanken und Erfahrungen des Autors mit Open Source Software dargestellt werden. Das Kapitel 8 fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf ausstehende Aufgaben.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen des Requirements Engineering-Prozesses erläutert. Zu diesem Zweck beschreibt das Kapitel die Phasen des Requirements Engineering(RE) Prozesses und die entsprechenden Aktivitäten. Darüber hinaus wird eine Definition von Open-Source-Software vorgestellt. Mehr über die Merkmale und Inhalte der Open-Source-Softwareentwicklung erfahren Sie im nächsten Kapitel 3.

2.1 Requirements Engineering

Nach der ISO/IEC/IEEE 29148:2011 ist das RE definiert wie folgt:

> „interdisciplinary function that mediates between the domains of the acquirer and supplier to establish and maintain the requirements to be met by the system, software or service of interest“[27].

Requirements Engineering hilft den Entwicklern, die Bedürfnisse des Kunden besser zu verstehen, so dass das Produkt besser entwickelt und anschließend gepflegt und überwacht werden kann. Hier ist die Kommunikation innerhalb des Kunden und des Softwareentwicklungsteams sehr wichtig. Dabei gibt es zwei große Kategorien, nämlich die Systemanalyse und die Verwaltung von Anforderungen. Abbildung 2.1 zeigt ein Modell des Requirements Engineering nach Börger et al[3]. Die Anforderungsanalyse gliedert sich in Elicitation, Analyse (Intrepetation, Negotiation), Dokumentation und Validieren und Verifizieren, die dazu dienen, die eingehenden Anforderungen zu analysieren und zu dokumentieren sowie zu überprüfen, ob das Produkt den Anforderungen entspricht. Im Anforderungsmanagement sind Change Management und Tracing für die Verwaltung und Verfolgung von Problemen und Änderungen in den Anforderungen zuständig.

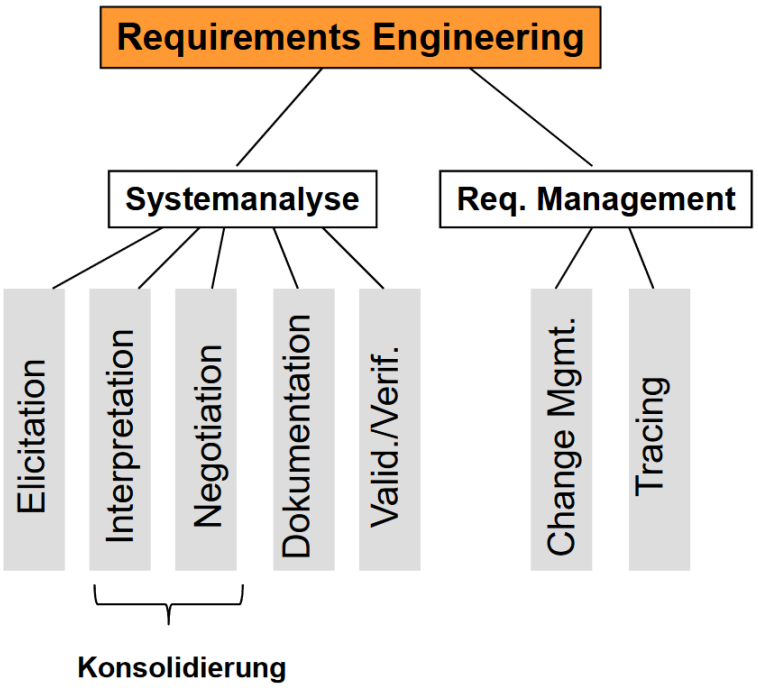

Abbildung 2.1: Das Referenzmodell des Requirements Engineerings von Schneider

2.1.1 Anforderungen

> „A software requirement is a property that must be exhibited by something in order to solve some problem in the real world.[4]“ (Pierre Bourque and Richard E. Fairley, 2014, p. 1-1)

Software-Anforderungen sind im Grunde genommen die Eigenschaften, die etwas haben muss, um ein Problem in der realen Welt zu lösen.[4] Die Anforderungen können nach ihrer Ausrichtung in Funktionale Anforderungen, Nicht-Funktionale Anforderungen, Prozessanforderungen und Projektanforderungen unterteilt werden.[4] Funktionale Anforderungen beschreiben die Funktionen, die eine Software ausführen soll, z. B. die Formatierung eines bestimmten Textes oder die Modulation von Signalen. Sie werden manchmal auch als Funktionen oder features bezeichnet. Nichtfunktionale Anforderungen werden in Qualitätsanforderungen und andere nicht-funktionale Anforderungen unterteilt.Leistungsanforderungen handelt. Nach der Definition von ISO 25010 gehören zu den Qualitätsanforderungen die Themen Flexibilität, Kompatibilität, Zuverlässigkeit usw.. Projektanforderungen sind Einschränkungen für die Softwareentwicklung in Bezug auf das Geld- und Zeitbudget. Prozessanforderungen sind Anforderungen

an den Prozess der Softwareentwicklung, z.B. eine Verifikationsphase mit zusätzlichen Einschränkungen.[4]

2.1.2 Elicitation

Bei der Elicitation von Anforderungen geht es um die Kommunikation mit der Kunden, welcher zum Ziel hat, die Bedürfnisse und Wünsche des Kunden zu ermitteln. Softwareingenieure nutzen diese Informationen, um die Anforderungen zu erfassen. Es ist der erste Schritt zum Verständnis des Problems, das die Software lösen soll. Eines der Grundprinzipien eines guten Prozesses zur Erhebung von Anforderungen ist die effektive Kommunikation zwischen den verschiedenen Stakeholdern[4]. Bevor die Entwicklung beginnt, muss der Anforderungsspezialist zwischen der Domäne der Softwareanwendung und der technischen Welt des Softwareingenieurs vermitteln.

Ein Schlüsselelement der Anforderungserhebung ist die Festlegung des Projektumfangs, d.h. die Beschreibung der spezifizierten Software und ihres Zwecks sowie die Festlegung von Prioritäten für die zu liefernden Ergebnisse, um sicherzustellen, dass die wichtigsten Geschäftsanforderungen des Kunden zuerst erfüllt werden. Auf diese Weise müssen die Anforderungsspezialisten weniger Zeit mit der Erhebung von Anforderungen verbringen, die von geringer Bedeutung sind oder die zum Zeitpunkt der Auslieferung der Software nicht mehr relevant sind.[4] Andererseits muss die Beschreibung zur Anforderung erweiterbar und skalierbar sein, um zusätzliche Anforderungen aufnehmen zu können, die nicht in der ersten formalen Liste ausgedrückt wurden, und sie muss mit früheren Anforderungen kompatibel sein. Übliche Methoden sind Interviews, Workshops und Fragebögen, etc..[4]

2.1.3 Analyse: Interpretation und Negotiation

Im Rahmen der Interpretation erfolgt eine strukturierte, kategorisierte und in Beziehung gesetzte Darstellung der ursprünglichen Anforderungen, wie sie in der Ausschreibungsphase formuliert wurden. Die zuvor beschriebene Klassifikation von verschiedenen Arten von Anforderungen wird hier ebenfalls angesprochen. Dies führt zu einer Optimierung der Klarheit der Anforderungen. Des Weiteren besteht die Möglichkeit, Überschneidungen ähnlicher oder doppelter ursprünglicher Anforderungen zu identifizieren, sodass eine Zusammenführung erleichtert wird. Zudem können nutzlose Erläuterungen aus den ursprünglichen Anforderungen entfernt werden, um die Anforderungen zu erhalten. In dieser Phase können Fehler auftreten. Der Zweck der Negotiation besteht darin, mit Hilfe dieser Verfahren die wahren Anforderungen aus den ursprünglichen Anforderungen abzuleiten und zu konkretisieren. In diesem Schritt werden auch die in der Interpretation identifizierten Fehler korrigiert. Auch hier ist es notwendig, die Anforderun-

gen zu priorisieren, um die Erstellung eines Release-Plans vorzubereiten. Es ist von essentieller Bedeutung, die Anforderungen in einer Weise zu formulieren, die eine leichte Verständlichkeit gewährleistet. Zudem ist es erforderlich, diese bei Bedarf näher zu erläutern. Im Idealfall kann ein Modell erstellt werden, mit dem die Anforderungen mit den Stakeholdern verknüpft werden können, um ein besseres Verständnis der Anforderungen zu erreichen. Ein Beispiel für ein solches Modell ist das i*-Modell. In diesem Modell werden die Abhängigkeiten zwischen dem Stakeholder und den Anforderungen beschrieben, wobei die Anforderungen nicht nur funktional sind, sondern auch nicht-funktionale Anforderungen enthalten sind. In einem solchen Modell kann der Zweck einiger Anforderungen klar analysiert werden. Dieser Prozess ist von entscheidender Bedeutung, da er es ermöglicht, potenzielle Mehrdeutigkeiten zu erkennen und durch Rücksprache mit den Stakeholdern zu korrigieren.[4]

2.1.4 Dokumentation

Die Aufgabe der Dokumentationsphase ist sehr wichtig für den Erfolg des Projekts, da er nicht nur die Anforderungen zu Beginn festhält, sondern auch das Vorhandensein zusätzlicher Anforderungen oder Änderungen während des Entwicklungsprozesses. Das bedeutet, dass die Informationen und Ergebnisse jeder Phase nachvollzogen und wiederverwendet werden können. Der Zweck der Dokumentation besteht darin, eine Spezifikation zu erstellen, in der alle Informationen über das System festgehalten und dokumentiert werden. Es sollte darauf geachtet werden, dass die Dokumentation in einer verständlichen und strukturierten Form unter Verwendung linguistischer Regeln verfasst wird. Für die Entwicklung von Anforderungen können verschiedene Techniken wie das UML Diagramm verwendet werden. Bei der Dokumentationserstellung werden in der Regel einfache aktive Sätze mit einfachem Vokabular verwendet. In der Dokumentation soll zielorientierte Formen verwendet werden, um quantifizierbare, testbare Merkmale zu beschreiben[4]. Dies dient in erster Linie dazu, die Klarheit der Anforderungen zu verbessern und Missverständnisse aufgrund von Doppeldeutigkeiten zu vermeiden. Gleichzeitig ist die Spezifikation nicht nur eine formale Beschreibung des Projekts, sondern auch ein Vertrag zwischen dem Kunden und dem Requirements Engineering.[4]

2.1.5 Validieren und Verifizieren

Die Validierung der Anforderungen stellt sicher, dass die Entwickler das richtige Endprodukt entwickeln. Dabei sollte besonders darauf geachtet werden, dass die vorherige schriftliche Dokumentation (z. B. technische Spezifikationen) die Anforderungen der Stakeholder widerspiegelt. Mit der Verifizierung wird sichergestellt, dass das Entwicklungsteam das Produkt

so entwickelt, wie es von den Stakeholdern definiert wurde. Ein gängigerer Ansatz besteht darin, dass die Benutzer die zuvor geschriebene Dokumentation, z. B. die Spezifikation, noch einmal überprüfen, um das Verständnis des Softwareentwicklungsteams zu überprüfen und sicherzustellen, dass der Inhalt der Dokumentation mit den Anforderungen der Stakeholder übereinstimmt. In ähnlicher Weise sind Visionsvideos eine gute Möglichkeit, die Softwarefunktionalität in einer kurzen, schrittweisen Überprüfung zu demonstrieren, die es dem Kunden ermöglicht, die Anforderungen an die Software eindeutig zu validieren.[4]

2.1.6 Change Management

Der Zweck des Change Managements besteht darin, Anträge auf Änderungen an Anforderungen während der gesamten Dauer eines Softwareprojekts effektiv zu verwalten und negative Auswirkungen von Änderungen zu vermeiden. Change Management umfasst die Überprüfung, Verwaltung, Dokumentation und Autorisierung von Änderungsanfragen. Ein standardisierter und strukturierter Ansatz ist für die erfolgreiche Umsetzung des Change Managements und der damit verbundenen Änderungen von Anforderungen unerlässlich. Change Management umfasst nur die Änderung von Anforderungen, nicht die Erstellung neuer Anforderungen. [4]

2.1.7 Tracing

Der Zweck der Tracing besteht darin, die Quellen der Anforderungen zu ermitteln und ihre Auswirkungen vorherzusagen. Bei der Nachverfolgung können Mängel im Änderungsmanagement festgestellt werden. Bei Problemen, die im Change Management auftreten, kann die Rückverfolgung dazu dienen, das Problem, seine Auswirkungen und Mängel zu identifizieren. Eine Anforderung sollte bis zu den Bedürfnissen und Stakeholdern zurückverfolgt werden können, die sie ausgelöst haben. Umgekehrt sollte eine Anforderung bis zu den Anforderungen und den Designeinheiten, die sie erfüllt haben, und zu den Testfällen, die sie validiert haben, zurückverfolgt werden können.[4]

2.2 Open Source Software

Die Open Source Initiative (OSI) wurde 1998 als gemeinnützige kalifornische Gesellschaft gegründet. Sie gibt eine Definition von Open Source, die von mehreren Regierungen international als Standard oder de facto Open Source anerkannt ist.[29] Open-Source-Software erlaubt sowohl das freie Kopieren als auch das Weitergeben. Gleichzeitig sollte der Quellcode der Software sichtbar sein, damit Benutzer ihn verstehen können und darauf basierend Modifikationen vornehmen dürfen. Wenn Open-Source-Software jedoch weitergegeben wird, müssen alle Empfänger der Software die gleichen

Rechte haben, wie sie in der ursprünglichen Lizenz festgelegt sind. Der Autor kann sich dafür entscheiden, den modifizierten Code in die offizielle Version der Software aufzunehmen, d.h. keinen Code zu behalten, der nicht der Designphilosophie entspricht oder von minderer Qualität ist, um sicherzustellen, dass die Software der Designphilosophie und der Qualität entspricht. Aber auch wenn sie nicht in der offiziellen Version enthalten ist, müssen Änderungen an Open-Source-Software den Quellcode offenlegen. Die obigen Ausführungen sind eine kurze Einführung in Open Source Software. Eine genauere Beschreibung finden Sie auf der offiziellen Website von OSI[1][28].

Heutzutage gibt es viele Open-Source-Software auf dem Markt, wie Firefox, Open Office, Linux und so weiter. Diese Software hat einen offenen Quellcode und erlaubt der Open-Source-Gemeinschaft, sie einzusehen und zu konfigurieren. Die Verwendung dieser Open-Source-Software ist kostenlos und zieht eine große Anzahl von Benutzern an.[30]

[1] `https://opensource.org/osd`

Kapitel 3

Verwandte Arbeiten

Nach der Beobachtung von Noll wurden zehn der insgesamt vierzehn Anforderungen wie „UTF8 encoding“ oder „SMS reminder“ in Firefox 2.0 aktualisierten Funktionen von offenen, benutzerorientierten Dokumenten wie Diskussionsforen, Bugzilla, Wikis usw. übernommen. Drei Anforderungen[1] wurden von konkurrierenden Projekten wie Opera, Internet Explorer usw. übernommen. Zwei Anforderungen wurden nicht veröffentlicht, bevor das Firefox-Produktanforderungsdokument veröffentlicht wurde. Die Gründe für die Anforderungen wurden von den Entwicklern auf der Grundlage ihrer eigenen Erfahrungen und Kenntnisse in sieben von vierzehn Anforderungen vorgeschlagen. Drei Anforderungen wurden aus externen konkurrierenden Projekten entnommen. Zwei Funktionen (Phishing-Schutz und verbesserte Suche) wurden direkt von erfolgreichen Erweiterungsimplementierungen abgeleitet (Google Safe Browsing bzw. Google Search Toolbar). Die restlichen zwei Anforderungen wurden von Benutzern vorgeschlagen, die keine Entwickler sind.[24]

Noll und Liu befassen sich auch mit einer anderen Open-Source-Software, OpenEMR. OpenEMR ist eine medizinische Software, die in den Vereinigten Staaten weit verbreitet ist[15]. Im Gegensatz zum Firefox-Projekt erfordert OpenEMR aufgrund der Überschneidung mit dem medizinischen Bereich ein höheres Maß an Fachwissen seitens der Entwickler, was die Analyse der Anforderungen erschwert. Von den dreizehn Anforderungen für OpenEMR wurden zehn von den Entwicklern auf der Grundlage ihrer Erfahrungen vorgeschlagen, die übrigen drei von den Benutzern. Diese Anforderungen wurden von neun Benutzern oder Entwicklern vorgeschlagen. Vier von ihnen waren Berater und drei waren Systemadministratoren. Einer dieser drei ist ein Kernentwickler, der auch in dem medizinischen Sektor ausgebildet ist. Die beiden anderen sind Mediziner. Im Rahmen dieses Projekts wurden alle Anforderungen in einem offenen Forum oder einem Tracker auf Sourceforge.net geäußert. Zur Veranschaulichung sei an dieser Stelle auf die Abbildung

[1] Einige Funktionen können mehrere Quellen für Anforderungen haben

Source	Firefox	OpenEMR
Asserted by developers		
- from personal experience	4	6
- from knowledge of user needs	3	4
- formally validated	1	0
- by implementation as extension	2	0
	10	10
Contributed by users	3	3
Influenced by competition	1	0
Total	14	13

Abbildung 3.1: Vergleich Firefox und OpenEMR nach Noll[25]

3.1 verwiesen. John Noll merkte an, dass die Dokumentation von OpenEMR im Vergleich zu Firefox, wo die Anforderungen informell in einem Forum festgehalten werden, relativ informell ist. John Noll erkannte zudem die Bedeutung der Benutzer als Mitwirkende bei der Entwicklung von Open-Source-Software an.[25].

In ihrer Studie weisen Thomas A. Alspaugh und Walt Scacchi darauf hin, dass die klassischen Anforderungen sich mit den typischen CSS-Kontexten in Verbindung bringen lassen. Hierbei handelt es sich um ein System, das von einem Entwicklungsteam für einen Kunden außerhalb dieses Teams erstellt wird. Die meisten oder alle vorgesehenen Benutzer des Systems sind ebenfalls außerhalb dieses Teams. Zudem verfügen die Entwickler entweder über Fachwissen in der Problemdomäne oder auch nicht. Schließlich wird das System gemäß einem Budget und Zeitplan entwickelt[1]. Sie weisen auch darauf hin, dass sich die Beschreibung der Anforderungen an Open Source Software auf die spezifische Funktionalität oder Lösung konzentriert, die das System bietet, und nicht auf die Beschreibung der Probleme oder Herausforderungen, die gelöst werden müssen. In dem Anforderungsdokument werden die Punkte eher allgemein als spezifisch formuliert. In dem angeführten Beispiel aus Bugzilla wurden die spezifischen Anforderungen über die Projekt-Mailingliste und das Diskussionsforum kommuniziert, diskutiert und vereinbart. In ihrer Zusammenfassung kommen sie zu dem Schluss, dass Anforderungen in Open-Source-Software nicht gut gehandelt habt werden und sogar an problematische Teile der traditionellen Softwareentwicklung erinnern. Sie kamen zu dem Schluss, dass ein Teil der Anforderungen, die in der traditionellen Softwareentwicklung gehandhabt

werden, in der Open-Source-Softwareentwicklung in einer anderen und angemessenen Form umgesetzt werden. Außerdem muss bei der traditionellen Softwareentwicklung nur ein einziges Dokument gepflegt werden, während bei Open Source die gesamte Community gepflegt wird. Die Stabilisierung und Erweiterung der Open-Source-Gemeinschaft kann durch die Förderung von Entwicklern mittels Kommunikation innerhalb der Gemeinschaft erreicht werden.[1]

In dem Fragebogen von Coelho et. al. an 118 Entwickler zum Scheitern von Open-Source-Software war der häufigste Grund für das Scheitern das Auftauchen stärkerer Wettbewerber auf dem Markt. An zweiter Stelle stand der Mangel an Zeit oder Interesse. Sie fanden heraus, dass das Fehlen von Richtlinien für die Mitwirkung an der Entwicklung eine der wichtigsten Praktiken ist, die den Erfolg von Open-Source-Software beeinträchtigen.[9]

In der empirischen Studie von Jasion Kuriakose und Jeffrey Parsons wurden Open-Source-Entwickler befragt. In ihrem Fragebogen bewerteten die Open-Source-Entwickler Praktiken im Requirements Engineering. Es wurden die Praktiken ermittelt, die im Requirements Engineering am häufigsten verwendet wurden und die die Entwickler als am effektivsten ansahen. Im Gegensatz dazu wurden auch die am wenigsten genutzten Praktiken und die Praktiken, die als ineffektiv angesehen wurden, ermittelt. Bei der Dokumentation wurden zum Beispiel die leicht lesbare Dokumentation und die Unterstützung der Leser bei der Suche nach den benötigten Informationen getrennt bewertet. Generische Praktiken wie das Setzen von Prioritäten und die Modellierung von Softwarearchitekturen wurden bei der Analyse der Anforderungen hoch bewertet. Bei der Validierung war die Erstellung von Testfällen am beliebtesten. Ebenso wie die eindeutige Identifizierung von Anforderungen im Anforderungsmanagement. In der Phase der Erstellung von Anforderungen sahen sie Schwierigkeiten in der Unternehmenskultur der Mitglieder (65 %), Unterschiede im Bildungshintergrund (59 %), Sprachprobleme (56 %) usw. Bei der Analyse der Anforderungen hingegen traten Probleme bei der Bewältigung von Konflikten (35 %), beim Setzen von Prioritäten (32 %) und beim Erreichen eines gemeinsamen Verständnisses (36 %) auf.[19]

Eine weitere Studie untersuchte die Leistungsfähigkeit der agilen Entwicklungspraxis Rapid Release bei der Entwicklung von Open Source Software. Die Autoren konnten in ihrer Untersuchung nachweisen, dass durch Rapid Release die Attraktivität der Software erhöhen und damit die Zahl der Teilnehmer an Open-Source-Software steigern. Zu den Teilnehmern zählen sowohl Nutzer der Software als auch neue Entwickler in der Gemeinschaft. Das Feedback der Nutzer kann durch Rapid Release häufiger eingeholt und der Marktanteil erhöht werden. Die Vorteile des Rapid Release neuer Funktionen, der schnellen Behebung von Fehlern und der schnellen Behebung von Sicherheitslücken machen Open-Source-Software effizienter, qualitativ besser und sicherer.[12]

Jarke et al. haben vier neue Prinzipien für das Requirements Engineering vorgeschlagen:

> We identify four new principles that underlie contemporary requirements processes, namely: (1) intertwining of requirements with implementation and organizational contexts, (2) dynamic evolution of requirements, (3) emergence of architectures as a critical stabilizing force, and (4) need to recognize unprecedented levels of design complexity. (Jarke et al. 2011) [16]

Die vier oben erwähnten neuen Prinzipien des Requirements Engineering stellen Aktualisierungen des Requirements Engineering dar. Sie schlagen sechs „V's" vor, um die Eigenschaften von Anforderungen zu beschreiben, nämlich Volume, Veracity, Volatility, Vagueness, Variance und Velocity. Die Attraktivität von Open-Source-Software wurde als Beurteilungsfaktor gewählt, da Open-Source-Projekte eine gewisse Attraktivität benötigen, um das Projekt aktiv und erfolgreich zu halten. Im Rahmen des Hypothesentests wurden die positiven Auswirkungen von Veracity und Vagueness auf die Entwicklung von Open-Source-Software, die negative Auswirkung von Volatility auf die Entwicklung von Open-Source-Software sowie die krummlinige Auswirkung von Volume als akzeptiert erachtet. Demgegenüber wurden die positive Auswirkung von Velocity und die negative Auswirkung von Variance auf die Entwicklung von Open-Source-Software abgelehnt.[32]

Kapitel 4

Interview

Ein Interview wurde durchgeführt, um ein besseres Verständnis für das Anforderungsmanagement und die Verwaltung von Open Source Software zu erhalten. Im Rahmen dieses Interviews wird ein Softwareentwicklungsingenieur eines großen Technologieunternehmens interviewt. Das Unternehmen zeichnet sich durch eine hohe Disziplin bei der Entwicklung von Software aus. Der Interviewte hat selbst nicht an der Entwicklung von Open-Source-Software gearbeitet, jedoch in einem Team von Open-Source-Software-Managern gearbeitet. Er spricht über seine eigenen Erfahrungen und Lektionen, die er bei der Entwicklung von Open-Source-Software gelernt hat.

4.1 Ziel des Interviews

Ziel der Interviews war es, die Ansichten und Meinungen von Entwicklern, die im Rahmen des Requirements Engineering arbeiten, in Bezug auf Open Source Software zu beobachten und ihre Akzeptanz von Open Source Software zu beobachten. Auf dieser Grundlage versuchen wir, ein tieferes Verständnis für die Entwicklung von Open-Source-Software zu gewinnen und die Schwierigkeiten bei der Verwaltung von Anforderungen herauszufinden. Diese Schwierigkeiten werden mit den Vor- und Nachteilen der Ansätze des Requirements Engineering verglichen. In einem Abschnitt mit offenen Fragen werden die Unterschiede zwischen Open-Source-Software und kommerzieller Software aus der Sicht des Entwicklers diskutieren und prüfen, ob es Möglichkeiten zur gegenseitigen Zusammenarbeit gibt.

4.2 Interviewsleitfaden

Im Rahmen des vorliegenden Interviews wurden insgesamt drei Einführungsfragen für die Einführung der Frage zwölf Hauptfragen gestellt, darunter eine Folgefrage. Die Fragen zielen auf die Erfassung von Problembereichen,

die im Kontext der üblichen Softwareentwicklung häufig auftreten. Für die Untersuchung von Open-Source-Software wäre es hilfreich, wenn die Befragten Beispiele anführen würden, um ihre Aussagen zu untermauern. Im einleitenden Teil des Interviews wurden drei Fragen gestellt, um zunächst die derzeitigen Tätigkeiten der Befragten zu ermitteln und sodann in das Thema der Open-Source-Softwareentwicklung einzuführen. Im Hauptteil des Interviews wurden die Anfangsfragen verwendet, um das Requirements Engineering in der traditionellen Entwicklung zu verstehen, mit dem der Befragte derzeit konfrontiert ist. Im Folgenden werden die Ansichten des Befragten zum Open-Source-Software-Engineering erfragt, wobei ein besonderer Fokus auf Projektmanagement, Dokumentation und ähnliche Aspekte gelegt wird. In der abschließenden Diskussion werden Herausforderungen von Open-Source-Software, Unterschiede zur traditionellen Entwicklung sowie Möglichkeiten des gegenseitigen Lernens erörtert. Es wird erwartet, dass die Antworten der Befragten einen Überblick über diese Diskussionen ermöglichen. Neue Ideen, die die abschließende Diskussion inspirieren, wären sehr willkommen. Ein ins Deutsche übersetzter Interviewleitfaden wurde in den Anhang A.1 aufgenommen.

4.3 Ergebnisse des Interviews

Aus das geführtes Interview ging hervor, dass die befragte Person an einem Unternehmensprojekt beteiligt waren, welches später in die Open-Source-Community gestellt wurde. In der Folge wurde er Mitglied des Administratorenteams dieses Open-Source-Projekts. In dem Interview stellte er der Open-Source-Gemeinschaft eine Open-Source-Software-Schnittstelle zur Verfügung, die nach Belieben von der Open-Source-Gemeinschaft hinzugefügt und erweitert werden konnte. Er wies darauf hin, dass die Berücksichtigung von Zeitzonen bei seiner Arbeit von großer Bedeutung ist. Aufgrund der räumlichen Distanz zwischen ihm und seinen Kollegen sowie der daraus resultierenden unterschiedlichen Zeitzonen ist die Wahrscheinlichkeit, dass sich die Bearbeitung eines Problems über einen ganzen Tag erstreckt, signifikant erhöht. Ein Beispiel für die erwähnten Zeitzonen ist die Fertigstellung eines Stücks Code, welches in den frühen Morgenstunden in seiner Region überprüft wurde. In der Konsequenz ist er gezwungen, am darauffolgenden Tag erneut zu erscheinen, um das Problem zu lösen. Er führte aus, dass sich diese Art von Problem im Open-Source-Modell nur schwer besser lösen lässt, da in Unternehmen in der Regel engere Fristen für die Bearbeitung von Aufgaben gelten. Zudem stellt das unbeständige Lieferdatum von Open-Source-Software keine Lösung des Problems dar. Die einzige Alternative besteht darin, alle potenziellen Probleme im Voraus in der Dokumentation aufzulisten, was jedoch ebenfalls mit einem hohen Aufwand verbunden ist.

Die dritte Frage des Hauptteils thematisiert die Abhängigkeiten von

Aufgaben. Abhängigkeitskonflikte, die bei der täglichen Arbeit mit dem Code einer anderen Gruppe auftreten, werden dieser Gruppe mitgeteilt. Wenn es Zeit oder Raum gibt, die Priorität in der betroffenen Gruppe zu ändern, wird gewartet. Ist die Anpassung zu schwierig, wird der Projektleiter informiert. In den befragten Unternehmen erfordert der Einsatz von Open-Source-Software, dass der Vorgesetzte zusätzliche Sicherheitsprüfungen durchführt, damit die Sicherheit der Open-Source-Software nicht die Sicherheit der kommerziellen Software gefährdet. Zu diesem Zweck verfügt ihr Unternehmen über eine Whitelist, in der zugelassene Software oder Codebases von Drittanbietern aufgeführt sind, deren Sicherheit das Unternehmen vertraut und die in der täglichen Arbeit eingesetzt werden dürfen. Nach dem Verständnis der Befragten wird Open Source Software in zwei Hauptbereiche unterteilt. Die erste ist Open-Source-Software, die vollständig von der Gemeinschaft organisiert wird, ohne kommerziellen Input. Der andere ist kommerzielle Software, die der Open-Source-Gemeinschaft zur Verfügung steht und an der die Open-Source-Gemeinschaft Änderungen usw. vornehmen kann.

Die Beantwortung der neunten Frage zum Thema Verwaltung von Open-Source-Software durch die Befragten erfolgte in erster Linie auf technischer Ebene, wobei die Interaktion zwischen den Teilnehmern eher gering ausfiel. Was den Code betrifft, so gibt es im Gegensatz zum Code in einem Unternehmen, der einem festen Muster folgt, kein festes Muster für den Code von Open-Source-Software, und jeder Entwickler hat ein anderes Niveau und einen anderen Stil des Codes. Aus diesem Grund muss der Verwaltungsprozess den gepushten Code und die Merge-Anfrage überprüfen und den Code bis zu einem gewissen Grad ändern oder inakzeptablen Code direkt zurückgeben. Was die Sicherheit betrifft, so gibt es neben der Überprüfung durch den Administrator auch einen speziellen Überprüfungsprozess in ihrer Gruppe, um Backdoors oder andere Sicherheitslücken zu vermeiden. Der Befragte wies darauf hin, dass es immer noch viele Schwierigkeiten bei dem Versuch gibt, eine Sicherheitslücke in der offiziellen Version zu veröffentlichen. Er sagte auch, dass Sicherheitsbedrohungen während seiner Verifikationserfahrungen nicht wirklich aufgetreten seien. Er merkte an, dass es mit zunehmender Größe von Open Source Software immer mehr Anfragen nach Funktionalität geben wird und dass Priorisierung hier sehr wichtig ist und es kaum einen Unterschied zur traditionellen Softwareentwicklung gibt. Er sagte, dass bei Open Source Software die Administratoren eine sehr wichtige Rolle spielen. Sie verbringen regelmäßig Zeit damit, Feature Requests und Merge Requests zu verwalten und bleiben sogar bis spät in die Nacht auf, um diese Arbeit zu erledigen, was einen großen Teil ihrer Welt einnimmt. Da diese Administratoren Open-Source-Software als Hobby oder für persönliche Zwecke begonnen haben, kommt es häufig vor, dass sie ihre Rolle als Administrator aufgeben, weil sie das Interesse oder die Zeit verlieren. In einem größeren Projekt gibt es daher in der Regel mehrere Administratoren. Gleichzeitig birgt die Übergabe

von Administratoren gewisse Risiken, und in der Diskussion über die Sicherheitsprobleme, die bei XZ Utils auftraten, sagten die Befragten auch, dass solche Risiken schwer zu vermeiden sind.

In Bezug auf das Entwicklerteam von Open-Source-Software hatte der Befragte einen interessanten Standpunkt: Bei Open-Source-Software ist das Kernteam für die Richtung der Software verantwortlich. Es ist von Natur aus schwierig oder unwahrscheinlich, die Entwicklung der Software vollständig in die Hände der Open-Source-Gemeinschaft zu legen. Dieses Kernteam kann mit dem Projekteigentümer im Requirements Engineering verglichen werden, wo jeder für seine Software verantwortlich ist und kontrolliert, wohin sie geht. Die Befragten gaben an, dass Open-Source-Software in einigen speziellen Bereichen im Wesentlichen der traditionellen Softwareentwicklung ähnelt. Die Geschäftslogik in der Software liegt nach wie vor in der Verantwortung von Fachleuten. Allerdings sind die Kommunikationsanforderungen bei Open-Source-Software wesentlich höher als bei der agilen Entwicklung vor Ort beim Kunden.

Der Befragte war nicht optimistisch, was die Zusammenarbeit zwischen Open Source und traditioneller Software betrifft und sagte, dass er sich die einzige Zusammenarbeit vorstellen könne, wenn Open Source Software kommerziell gekauft werde. Für die Open-Source-Entwickler sei es nicht gut, wenn sie an der Entwicklung der Software mitarbeiten, für die Nutzung bezahlt werden oder eine Mitgliedschaft abschließen müssten. Die Früchte ihrer Arbeit würden dem Unternehmen zufallen. Dies kann für die Entwickler schwer zu akzeptieren sein. Insgesamt steht der Befragte der Entwicklung von Open Source Software positiv gegenüber. Er glauben, dass Open-Source-Software eine größere Gemeinschaft und mehr Nutzer mit sich bringt. Die Open-Source-Gemeinschaft sei freundlicher als die kommerzielle, entspreche dem Geist der Zusammenarbeit und Innovation im Internetzeitalter und sei es wert, gefördert zu werden.

Kapitel 5

Analyse

Im Folgenden wird das Anforderungsmanagement der großen Open-Source-Software Firefox analysiert. Bugzilla ist eine kostenlose Open-Source-Software, die In der Programmiersprache Perl geschrieben wurde und am 26. August 1998 erschien. Die Software ist unter der Mozilla Public License veröffentlicht. Zahlreiche freie Software-Projekte werden von Bugzilla verwaltet, wie beispielsweise OpenOffice[1], Eclipse[2] und Firefox[5]. Im nachfolgenden Abschnitt wird dargelegt, wie Firefox Bugzilla zur Erfassung, Analyse, Dokumentation, Verwaltung und Nachverfolgung von Anforderungen eingesetzt werden kann. Aufgrund der Vielzahl an Funktionen, wie beispielsweise Logins, Konten und Verzeichnisse, ist Firefox ein vielseitiges Programm. In Bugzilla sind diese Funktionen in separate Abschnitte unterteilt. Die hier präsentierte Analyse basiert auf den Daten, die am 1. Juli 2024 von bugzilla-bezogenen Websites erhoben wurden.

5.1 Beobachtung Firefox

5.1.1 Beschreibung

In Abbildung 5.1 wird ein Screenshot der Benutzeroberfläche von Bugzilla[3] präsentiert. In dieser Abbildung sehen wir, dass jedes Feature Request eine eindeutige und einzigartige Nummer erhält(A). Darüber hinaus wird jedes Feature Request kategorisiert(B) und eine kurze Beschreibung des Problems(C) gegeben. Darüber hinaus wird für jedes Feature Request das Projekt, zu dem es gehört, sowie der spezifische Teil des Projekts und das Datum, an dem das Feature Request zuletzt im Forum geändert wurde(D), festgehalten. Darüber hinaus gibt es die zwei Schlüsselwörter Assignee(E), Status(F). Die Abbildung 5.2[4] präsentiert die Seite eines

[1] `https://www.openoffice.org/`

[2] `https://www.eclipse.org/`

[3] `https://bugzilla.mozilla.org/buglist.cgi?product=Firefox&component=Bookmarks%20%26%20History&resolution=---` Stand: Mon Jul 1 2024 02:37:58 PDT

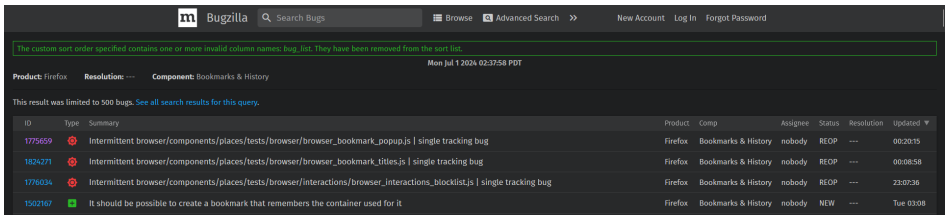

Abbildung 5.1: GUI von Bugzilla

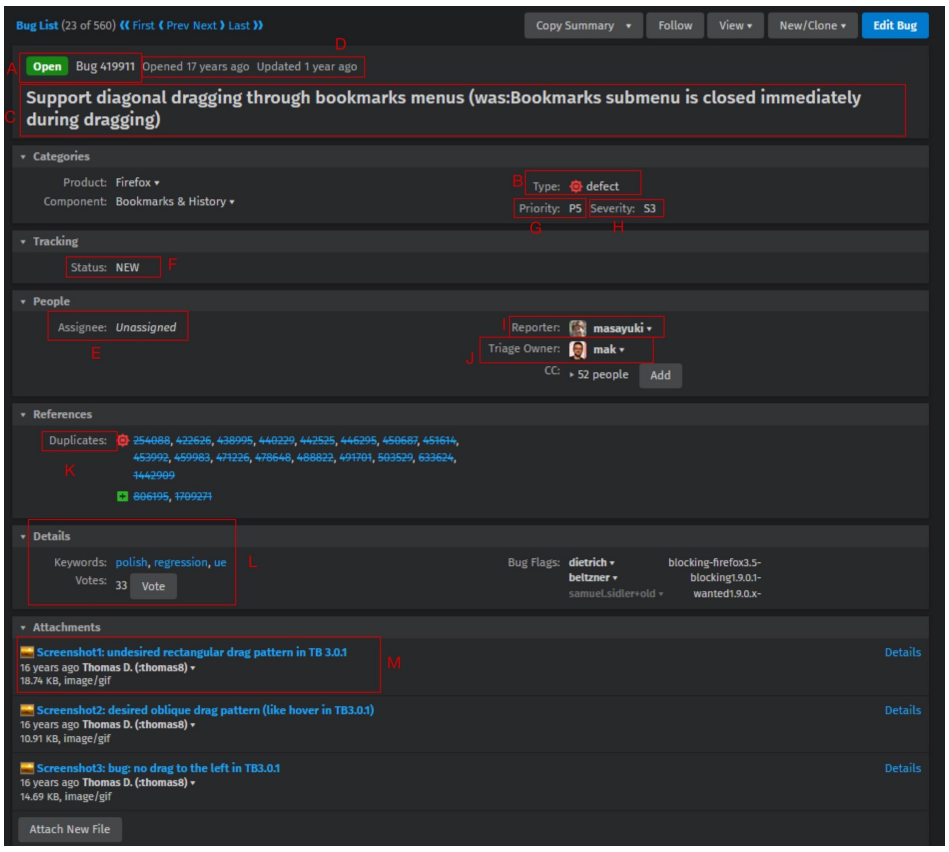

Abbildung 5.2: GUI von Bugzilla

ausgewählten Feature Requests. Da bei der Einführung möglichst viele Schlüsselwörter auf der Seite vorkommen sollten, wurde für die Beschreibung ein Feature Request gewählt, der schon lange existiert und relativ viel diskutiert wurde. Auf der Seite ist zu sehen, dass jeder Anfrage eine Priorität(G) und ein Schweregrad(Severity)(H) zugewiesen wurde, der auf der vorherigen Seite basiert. Danach folgen der Status der Bearbeitung und die Personen, die an der Anfrage beteiligt sind. Zwei neue Schlüsselwörter, „Reporter"(I) und „Triage Owner"(J), werden hier eingeführt. Die Spalte „Referenzen" beschreibt die Abhängigkeiten und Vorgänger der Anfrage sowie ähnliche Probleme(Duplicates)(K), die ausschließlich in dieser Anfrage vorkommen(blocks). Wenn eine Feature-Anfrage in den „block" einer anderen Feature-Anfrage auftaucht, wird die Feature-Anfrage in der Liste außerdem rot dargestellt. Und der Schweregrad wird auf „Blocker" gesetzt. In der Spalte „Details"(L) finden Sie die Schlüsselwörter und die Anzahl der Stimmen zu dieser Anfrage. Schließlich kann die Funktionsanfrage auch konkret beschrieben werden, indem Sie einen Screenshot der Datei hochladen(M) usw..In Abbildung 5.3 [5] sehen Sie einen Reporter, der eine Funktionsanfrage in den Foren diskutiert.

5.1.2 Erklärung

In diesem Unterabschnitt erfolgt eine Erläuterung und Analyse der in der vorherigen Abbildung erwähnten Schlüsselwörter. Die Art einer Feature-Anfrage wird in drei Kategorien eingeteilt, nämlich Defekt(defect), Verbesserung(enhancement) und Aufgabe(task). Eine Defekt-Anfrage bezieht sich auf einen Fehler oder ein Sicherheitsproblem, das während der Entwicklung behoben oder korrigiert werden muss. Eine VerbesserungsAnfrage zielt auf die Optimierung des Projekts ab, beispielsweise durch Hinzufügen neuer Funktionen oder durch Leistungsverbesserungen. Eine Aufgabe-Anfrage umfasst Refactoring des Projekts, Löschen und Ersetzen von Code. Eine Verbesserung des Projekts kann beispielsweise in Form neuer Funktionen oder einer Leistungssteigerung erfolgen. Eine Aufgabe umfasst hingegen Refactoring des Projekts sowie das Löschen und Ersetzen von Code. Im Gegensatz zu Enhancement ist Task eher auf die Entwicklergemeinschaft ausgerichtet, während Enhancement auf die Optimierung für den Kunden abzielt. Der in Abbildung 5.1 erwähnte Beauftragte ist die für die Anfrage zuständige Person, während der aktuelle Status und das Ergebnis der Anfrage in Status bzw. Resolution festgehalten werden. Der Übergang des Status unterliegt gewissen Regeln. So kann der Status „assigned" lediglich in „Neu" geändert werden, wenn ein Wechsel des Beauftragten stattgefunden hat, während der Status „resolved" nur dann erreicht wird, wenn die Anfrage

[4] `https://bugzilla.mozilla.org/show_bug.cgi?id=419911`
[5] `https://bugzilla.mozilla.org/show_bug.cgi?id=429392`

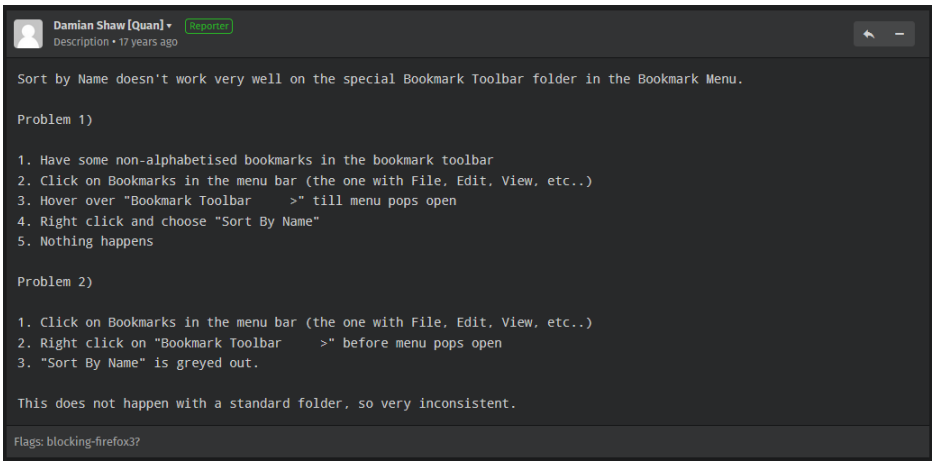

Abbildung 5.3: Diskussionsforum

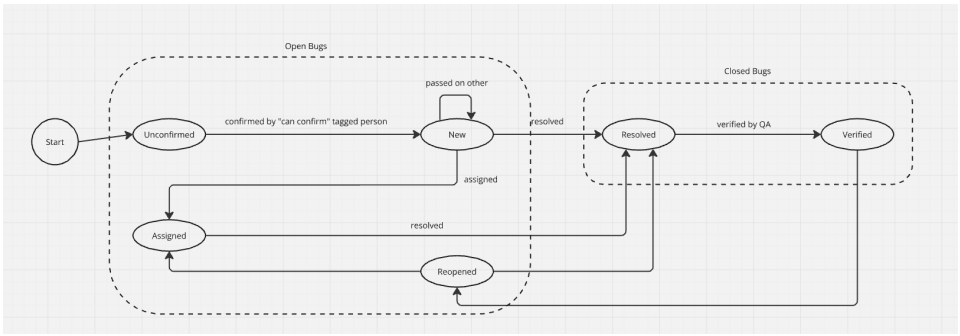

Abbildung 5.4: Status Zustandsdiagramm

abgeschlossen ist. Abbildung 5.4[6] veranschaulicht das Zustandsdiagramm des Übergangs der Feature Request. Zusätzlich zur unentschiedenen Standardpriorität gibt es fünf Prioritätsstufen, die bestimmen, ob und wann ein Feature Request bearbeitet wird. P1 bis P3 legen fest, wann die Anfrage für dieses oder das nächste Update oder später bearbeitet wird, P4 ist speziell für den Web Platform Test Bot gedacht und P5 wird nicht bearbeitet, sondern von der Open Source Community geprüft, wenn diese ein Update einreicht. Der Schweregrad des Problems wird ebenfalls in vier Stufen eingeteilt, wobei die schwerste Stufe S1 ein katastrophales Problem bezeichnet, das die Entwicklung behindern oder mehr als 25% der Nutzer oder Datenverluste betreffen könnte. Die Stufen s2 bis s4 basieren auf den Auswirkungen des Problems, der Verfügbarkeit eines Workarounds und dem Ort, an dem das Problem auftritt, z.B. in einer wichtigen Funktion. Der Schweregrad wird hier hauptsächlich verwendet, um eine Rückmeldung über die Auswirkung

[6] `https://wiki.mozilla.org/BMO/UserGuide` Accessed: 2024-07-02

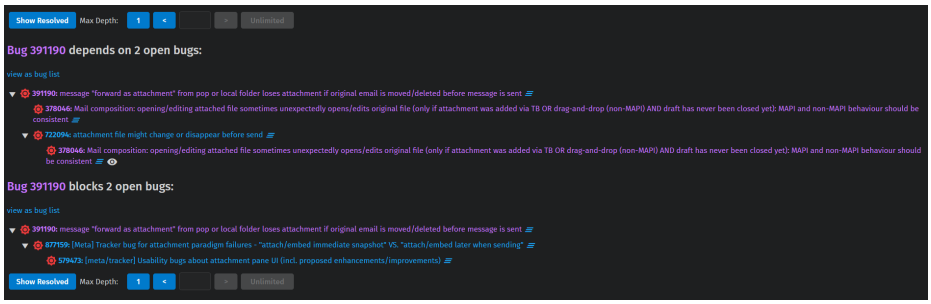

Abbildung 5.5: Dependency Tree & Block Tree

des Fehlers zu geben, während für Dinge wie Verbesserung oder Aufgabe N/A verwendet wird und es einen Standardschweregrad gibt, der durch -ängezeigt wird. Der Reporter ist die Person, die diesen Fehlerbericht schreibt. Der Triage-Prozess wird verwendet, um sicherzustellen, dass jede Funktionsanfrage kategorisiert und priorisiert wird, um sicherzustellen, dass kritische Probleme nicht übersehen werden. Insbesondere identifiziert der Triage-Eigentümer nicht klassifizierte Feature-Anfragen, legt deren Priorität und Schweregrad fest und aktualisiert einige der bestehenden Prioritäten nach vorgegebenen Regeln. Der Engineering Manager und der Direktor sind dafür verantwortlich, jedem Board Triage-Eigentümer zuzuweisen.

Die Verwaltung von Abhängigkeiten umfasst nicht nur die Auflistung doppelter Feature-Anfragen, sondern auch die Darstellung von Fällen, in denen eine Abhängigkeit von einer anderen Feature-Anfrage besteht. Zudem wird die Blockierung der Entwicklung einer Feature-Anfrage durch eine andere Feature-Anfrage verdeutlicht. Das Firefox-Team verwendet in Bugzilla einen Baum oder ein Diagramm, wie in Abbildung 5.5[7] dargestellt, um diese Zusammenhänge zu veranschaulichen. Der Versuch des Autors, auf das Diagramm zuzugreifen, führte jedoch zu einem schlechten Gateway-Fehler.

5.2 Ergebnisse

Aus den obigen Beobachtungen geht hervor, dass Bugzilla bei der Verwaltung der Anforderungen an die Open-Source-Software Firefox die Anforderungen definiert und in der Dokumentation festgehalten hat. Als nächstes vergleichen wir das Bugzilla Requirements Management Modell mit dem Requirements Engineering Modell. Wir beginnen mit den sechs Phasen 2.1 des Requirements Engineering. Vergleichen Sie, ob ihre Rollen in Bugzilla implementiert sind, und wenn ja, auf welche Weise. Wenn nicht, ob es Workarounds oder Alternativen gibt.

[7] `https://bugzilla.mozilla.org/show_bug.cgi?id=391190`

5.2.1 Elicitation

In Bugzilla stellt der Reporter, der die Funktionsanfrage im Forum beschreibt, die Hauptquelle für Anfragen dar. Im Forum präsentiert der Berichterstatter das Ergebnis des Problems, wie in Abbildung 5.3 dargestellt, welche eine Lesezeichensortierung beschreibt. Es sei darauf hingewiesen, dass das Problem keine Beschreibung einer spezifischen Lösung oder eine Diskussion über den Code beinhaltet. Im Folgenden werden die Schritte detailliert beschrieben, die erforderlich sind, um die Probleme zu reproduzieren. In Bezug auf den zuvor erläuterten Zweck der Elicitation sei an dieser Stelle noch einmal darauf verwiesen, dass Die Elicitation im Requirements Engineering dient der Identifikation der Stakeholder, der Sammlung von Anforderungen sowie deren Priorisierung. In der Regel obliegt diese Aufgabe dem Product Owner und dem Kunden. Es sei an dieser Stelle angemerkt, dass die Richtung der Open-Source-Software maßgeblich durch das Open-Source-Entwicklungsteam bestimmt wird und nicht durch den Kunden diktiert wird, da keine kommerzielle Beziehung besteht. Auch der Berichterstatter ist nicht der Kunde, und die Prioritäten bei der Elicitation werden intern vom Entwicklungsteam und nicht in Absprache mit dem Kunden festgelegt. Die Priorisierung und der Schweregrad werden durch das oben erwähnte Triage-System berücksichtigt. Es fehlt eine der Hauptrollen des Kunden bei der Entscheidung über den Stakeholder. Dies erschwert die Beurteilung, ob die Ziele der Elicitation bei Open-Source-Software erreicht werden.

5.2.2 Analyse

Die Anforderungen werden zunächst im Forum analysiert und validiert. In den Kommentaren unter „Feature Request" diskutieren triage owners und Reporter die Systemumgebung, wie das Betriebssystem, die Version des Programms usw., sowie den Prozess der Reproduktion des Bugs, die Beschreibung neuer Features und die Rate der Bewertung. Vorläufige Lösungen können ebenfalls einen wertvollen Teil der Anforderungsanalyse darstellen. Die zuvor dargelegte Analyse des Defektteils bildet die Grundlage für die Erstellung eines Vorschlags zur Erweiterung durch einen Berichterstatter. Dieser Vorschlag umfasst eine kurze Beschreibung der gewünschten Funktion sowie eine Einschätzung der Machbarkeit der Funktion durch den Eigentümer der Triage. Im Vergleich zur Negotiation und Interpretation im Requirements Engineering sind abgesehen von den unterschiedlichen Kommunikationsmethoden und den verschiedenen Kommunikationsrollen, die im vorigen Unterabschnitt erwähnt wurden, alle Diskussionen in Bugzilla offen und transparent für die Gemeinschaft. Zudem ist die in der Kommunikation verwendete Wörter und Sätze informeller er als bei geschäftlichen Treffen.

5.2.3 Dokumentation

In Bugzilla existiert keine kanonische Dokumentation, die jeden Fehler oder jede Funktionsanfrage abdeckt. Stattdessen erfolgt eine transparente Diskussion in den Foren. Die Diskussion wird als Fehlerbeschreibung dokumentiert. Des Weiteren werden Informationen wie die beteiligten Personen, Prioritäten usw. in gleicher Weise wie in Abbildung 5.2 dokumentiert. Im Gegensatz zum Requirements Engineering, wo die Dokumentation stärker formalisiert, Teil eines Vertrags und gesetzlich geschützt ist, ist sie in Bugzilla eher wie eine Vereinbarung oder ein Protokoll zu betrachten. Sie erfüllen unterschiedliche Funktionen. Daher ist die Dokumentation in Bugzilla transparent, offen, prägnant und flexibel im Vergleich zur Dokumentation der traditionellen Softwareentwicklung gemäß Requirements Engineering. Eine Anfragevorlage in Bugzilla umfasst in der Regel eine Übersicht, eine Beschreibung des Problems, eine Anleitung zur Reproduktion, erwartete Ergebnisse sowie tatsächliche Ergebnisse. In einigen Fällen wird zudem ein Screenshot des Problems hinzugefügt. Diese Darstellungsform ist wesentlich prägnanter als die traditionelle Dokumentation des Requirements Engineering. Die Softwarearchitektur, Testfälle etc. sind nicht im Bugzilla-Text enthalten. Dies ist darauf zurückzuführen, dass die Kernbestandteile der Software, wie beispielsweise die Architektur, bereits im Bugzilla definiert sind. Obgleich sich eine Vielzahl von Feature-Anfragen in der Datenbank befindet, erfolgt eine Zuweisung der entsprechenden Anfragen zu den zuständigen Boards, sobald der Hauptteil der Software identifiziert wurde. Die Kernentwickler verfügen über ein detaillierteres Wissen bezüglich des Inhalts der von ihnen verantworteten Boards als die Entwickler in der Open-Source-Gemeinschaft. Das Entwicklungsteam ist für die Dokumentation von Sicherheits- und anderen Problemen wie zum Beispiel Abhängigkeiten, mit denen es selbst konfrontiert ist, verantwortlich. Infolgedessen kann die fehlende Dokumentation des Requirements Engineering an dieser Stelle erklärt werden.

5.2.4 Validieren & Verifizieren

Wie bereits dargelegt, dient die Validierung der Sicherstellung der korrekten Entwicklung der Projekte. In ähnlicher Weise bezieht sich das richtige Projekt bei der Entwicklung von Open-Source-Software auf den Zustand, in dem das Kernteam der Open-Source-Software seine Software haben möchte. Es ist bemerkenswert, dass der Autor eine stichprobenartige Untersuchung von 20 abgeschlossenen Feature-Requests für ihre Auftraggeber durchgeführt haben. Dabei stellte sich heraus, dass die meisten dieser Requests (nämlich 16) mit dem Eigentümer oder Berichterstatter der Triage identisch waren. Dies bedeutet, dass in vielen Fällen dieselbe Person den Fehler findet und ihn korrigiert. Dadurch werden Fehler in der Kommunikation und im Verständnis der Anforderungen weitgehend vermieden. In den übrigen Fällen berichten

die Beauftragten auch im Forum über den aktuellen Fortschritt und ihre Lösungen. In allen Fällen wird der eingereichte Code von einem Code-Reviewer überprüft, der den Code auf Korrektheit, Qualität, Stil, Sicherheit usw. überprüft.[8] Dadurch wird sichergestellt, dass die Feature-Anfrage richtig verstanden und abgeschlossen wird. Es sei darauf hingewiesen, dass es einen großen Unterschied zwischen diesem Verfahren und dem traditionellen Requirements Engineering in der Softwareentwicklung gibt. Da der Kunde in diesem Prozess keine Rolle spielt, wird durch das Verifizieren und Überprüfen die Kommunikation bis zu einem gewissen Grad reduziert. Die Überprüfung des Codes und das Testen sind jedoch weiterhin wesentliche Bestandteile des Prozesses.

5.2.5 Change Management

Die Änderungsverwaltung von Firefox sowie der Prozess für Funktionsanfragen weisen eine hohe Ähnlichkeit auf. Für eine detaillierte Betrachtung des spezifischen Prozesses sei auf den 5.2.2 verwiesen. Der Prozess einer Feature-Anfrage umfasst die folgenden Schritte: Zunächst meldet ein Reporter eine Feature-Anfrage, welche im Anschluss durch einen Triager analysiert wird. Daraufhin erfolgt eine Zuweisung der Anfrage an einen Entwickler sowie eine Überprüfung durch einen Tester oder Reviewer, bevor schließlich der Status der Anfrage durch den jeweiligen Entwickler geändert wird. Die Ergebnisse des Vergleichs mit dem Requirements Engineering sind ähnlich wie die vorherigen.

5.2.6 Tracing

Die Nachverfolgung auf Bugzilla erfolgt primär über die Statusmeldung. Die Entwickler sind dazu befugt, diesen Status zu ändern, um den aktuellen Stand der Funktionsanfrage zu beschreiben. Dazu zählen beispielsweise die Statusmeldungen „needinfo" oder „verified". Darüber hinaus bietet die Auflistung der verwandten Feature Requests und der Abhängigkeitsstrukturen/Grafiken der Feature Requests in der Referenz eine Hilfestellung bei der Verfolgung. Dadurch lassen sich die Auswirkungen einer Feature Request identifizieren. Die Gründe für Abhängigkeits- und Anforderungsänderungen sowie für Statusänderungen sind in Bugzilla gut dokumentiert.

5.3 Zusammenfassung & Diskussion

Die vorangehende Analyse lässt den Schluss zu, dass das Open-Source-Projekt Firefox im Allgemeinen gut organisiert ist. Allerdings zeigen sich in Bezug auf das Requirements Engineering traditioneller Software signifikante

[8] `https://firefox-source-docs.mozilla.org/contributing/reviewer_checklist.html` Accessed: 2024-07-08

Unterschiede. In diesem Kontext ist der vollständige, leicht zu durchsuchende, einfache und leicht verständliche Bugzilla-Benutzerleitfaden von entscheidender Bedeutung. Die Benutzerleitfaden beinhaltet die wesentlichen Richtlinien und Empfehlungen für die Entwicklung sowie Anforderungen und Vorlagen. Zudem bietet es eine Vielzahl an Hilfestellungen für die Durchführung von Tests und weiteren technischen Aspekten. Des Weiteren wurde seitens des Entwicklerteams eine Suchmaschine implementiert, welche es Neulingen in der Open-Source-Gemeinschaft erleichtert, Informationen zu finden. Des Weiteren wird bei jedem der relativ speziellen Schlüsselwörter in der Funktionsanforderung eine Erklärung eingeblendet. Die Entwicklung einer gut strukturierter Benutzerleitfaden stellt eine exzellente Praxis dar.[19]

Bei einem Vergleich der Open-Source-Softwareentwicklung mit der traditionellen Softwareentwicklung lässt sich feststellen, dass die Entwickler und Benutzer in der Open-Source-Community mit den Kunden in der traditionellen Softwareentwicklung verglichen werden können, während der Projekteigentümer den Kern des Open-Source-Softwareentwicklungsteams bildet. Der Projekteigentümer stellt das zentrale Mitglied des Open-Source-Software-Entwicklungsteams dar. Gleichzeitig obliegt den Kernentwicklern auch die Arbeit der Entwickler in der Softwareentwicklung. Der wesentliche Unterschied besteht darin, dass die Rolle der Kunden in der Open-Source-Softwareentwicklung ohne finanzielle Unterstützung eine deutliche Verschiebung zugunsten des Projekteigentümers erfährt. Die Entwicklungsabsichten der OSS-Kernmitarbeiter selbst sind von entscheidender Bedeutung für die Bestimmung der Entwicklungsrichtung. Dies ist der Hauptgrund für die zuvor analysierten Unterschiede.

Bugzilla stellt eine exzellente Plattform für das Anforderungsmanagement von Firefox sowie für die Anforderungsanalyse, Dokumentation usw. dar. Diese Plattform ist für Firefox als große Open-Source-Software ein wesentliches Medium zur Kommunikation mit der Community. Sie zieht Benutzer und Techniker an, sich Firefox anzuschließen, wodurch Firefox gegenüber starken Konkurrenten wie Google Chrome, Opera und anderen Browsern im Wettbewerb bestehen kann und seine eigenen Merkmale entwickelt. Der Erfolg von Firefox als Open-Source-Software ist unbestreitbar.

Kapitel 6

Modell

Der Vergleich der Unterschiede in der Anforderungsbearbeitung zwischen Open-Source-Softwareentwicklung und traditioneller Softwareentwicklung erfolgt anhand eines Modells (siehe Abbildung 6.1), welches die Unterschiede zwischen den beiden Entwicklungsansätzen während der Planungsphase, der Implementierungsphase und der anschließenden Problembearbeitung der Softwareentwicklung analysiert. Ziel ist die Gewinnung eines tieferen Verständnisses für die Anforderungsbearbeitung von Open-Source-Software. Im Anschluss erfolgt eine Gesamtbeschreibung, in der die Vorteile, Nachteile und Schwierigkeiten der verschiedenen Entwicklungsansätze analysiert werden. Zudem werden die eigenen Gedanken des Autors dargelegt.

6.1 Vor dem Projekt

Im ersten großen Teil des Modells erfolgt eine Analyse der Unterschiede zwischen Open-Source- und traditioneller Softwareentwicklung sowie des Requirements Engineering bei der Vorbereitung der Softwareentwicklung. Gegenstand der Betrachtung sind dabei die Anforderungsanalyse, rechtliche und lizenzrechtliche Fragen, Ziele und Motivation, Planung und Kosten.

6.1.1 Anforderungen

Im Rahmen der Anforderungserhebung erfolgt zunächst die Anforderungserfassung, anschließend die Anforderungsermittlung und schließlich die Anforderungserfassung[4]. Im Rahmen dessen ist es erforderlich, die an der Softwareentwicklung Beteiligten zu identifizieren und die Quellen von Anforderungen in der Softwareentwicklung zu ermitteln. Zudem ist es von Bedeutung, unbearbeitete Anforderungen zu sammeln. Zu den Stakeholdern gehören Benutzer, Kunden, Entwicklungsteams, Marketing, Management und weitere Akteure. Eine besondere Bedeutung kommt dabei der Kommunikation zu. Üblicherweise werden hierfür Interviews, Workshops und ähnliche

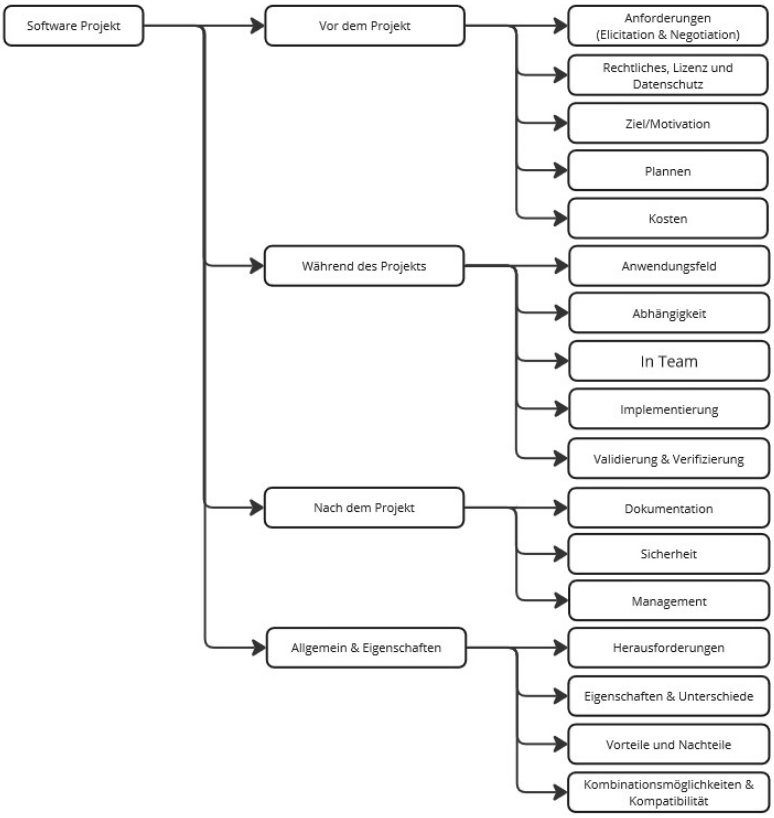

Abbildung 6.1: Modell

Methoden eingesetzt. Im Anschluss an die Erhebung erfolgt eine Analyse der Anforderungen, in deren Rahmen eine Bestätigung derselben erfolgt, um etwaige Missverständnisse zu vermeiden. Auf dieser Grundlage erfolgt schließlich eine Kategorisierung der Anforderungen sowie eine gemeinsame Priorisierung mit dem Kunden.

Aus früheren Interviews aus Kapitel 4 lässt sich ableiten, dass eine Form der Open-Source-Softwareentwicklung ist die Erstellung von Erweiterungen auf der Grundlage bestehender Software. Ebenso ist es möglich, dass ein Entwicklerteam eine Open-Source-Softwareentwicklung von Grund auf beginnt. Die Quelle der Anforderungen kann dabei sowohl aus der Open-Source-Community stammen, beispielsweise in Form von Fehlerberichten auf Bugzilla, als auch von der Person, die sie anfordert, erfolgen. Dabei kann es sich um einen Benutzer oder einen anderen Entwickler handeln. Alternativ können die Anforderungen auch von den Kernentwicklern stammen. Die Kernentwickler können ebenfalls Anforderungen haben, die auf ihren eigenen Bedürfnissen, Erfahrungen usw. beruhen, oder sie haben hervorragende Funktionen in Produkten mit ähnlicher Ausrichtung auf dem Markt beobachtet und möchten davon lernen.[25] Bei Open-Source-Software werden die Anforderungen in erster Linie von der Benutzergemeinschaft selbst formuliert, während bei kommerzieller Software die Anforderungen in der Regel von den Kunden stammen.[11] Bei der Entwicklung von Open-Source-Software sind verschiedene Akteure involviert, darunter Entwickler, Projektverantwortliche, Tester und Endbenutzer.[23] Ein wesentlicher Unterschied zur traditionellen Softwareentwicklung besteht darin, dass bei der Entwicklung von Open-Source-Software marktbedingte Einflüsse seltener sind[13].

In einer früheren Fallstudie zu Firefox in Kapitel 5 wurde festgestellt, dass die Priorisierung von Anforderungen in Open-Source-Software durch das Kernteam erfolgt und dass die Priorisierung von Anforderungen in Open-Source-Software in der Regel bestimmt, in welchen Updates die Änderungen veröffentlicht werden. In der traditionellen Softwareentwicklung erfolgt die Priorisierung von Funktionen für die nächste Runde von Sprints. Das Entwicklungsteam erhält dann ein nach Prioritäten geordnetes Product Backlog. Auch hier sind die Funktionen der Prioritäten ähnlich.

In der Phase des Verstehens der Anforderungen ist es bei der traditionellen Softwareentwicklung von zentraler Bedeutung, die Anforderungen aus der Sicht des Kunden zu verstehen und gleichzeitig dafür zu sorgen, dass das Entwicklungsteam die Anforderungen richtig versteht. Hierbei ist es von essenzieller Bedeutung, die Anforderungen aus der Sicht des Kunden zu verstehen. Im Rahmen des Interviews wurde jedoch ersichtlich, dass die Richtung der Open-Source-Softwareentwicklung maßgeblich von den Hauptnutzern bestimmt wird. Die Möglichkeit, die Anforderungen zum Zeitpunkt der Entwicklung subjektiv zu ändern oder sogar ein Veto einzulegen, ist ein wesentlicher Aspekt der Open-Source-Softwareentwicklung. Ein Beispiel

hierfür sind die Status-Tags für Wontfix in Firefox. Dies verdeutlicht, dass die Open-Source-Softwareentwicklung einen maßgeblichen Einfluss auf die Anforderungen und die Arbeit ausübt.

6.1.2 Rechtliches, Lizenz und Datenschutz

Im Rahmen des traditionellen Requirements Engineering für Software werden die rechtlichen Aspekte in der Regel im Vertrag festgehalten. Diesbezüglich sei auf den von der IHK München zur Verfügung gestellten Mustervertrag für die Erstellung von Software[1] verwiesen, welcher die folgenden Inhalte aufweist: Vertragsparteien, Softwarequalität, funktionsbezogene Inhalte, Änderungs- und Lieferbedingungen, Quellcode, Haftung usw.

In Bezug auf Open-Source-Software werden diese durch die Lizenz der Open-Source-Software bestimmt. Die Abbildung 6.1.2 präsentiert eine Analyse mehrerer gängiger Open-Source-Softwarelizenzen. Im Rahmen dieser Analyse werden die Unterschiede zwischen den Lizenzen im Hinblick auf die Distribution von Closed-Source-Software nach Veränderung, Extension, Patent, Weitergabe, einschließlich der Weitergabe des Quellcodes, untersucht. Die Begriffe „Veränderung“, „Extension“ und „Verkauf“ beziehen sich auf die Frage, inwiefern eine Closed-Source-Distribution mit Modifikationen, eine Erweiterung der Lizenz sowie der Verkauf der Software unter der Lizenz zulässig ist. Der Begriff „Patent“ betrifft die Frage, ob die Lizenz eine Patentlizenz oder damit verbundene Bedingungen enthält. Der Begriff „Weitergabe“ schließlich betrifft die Frage, ob der Quellcode in die Distribution aufgenommen werden muss. Die Darstellung der Lizenzen erfolgt in einer Reihenfolge, die von einer gewissen Flexibilität bis hin zu einer strengeren Auslegung reicht. Entwicklerteams für Open-Source-Software können die für ihre Projekte relevanten Lizenzen frei wählen, um die Offenheit der Software gegenüber der Gemeinschaft zu steuern. Die Offenlegung der Quellcodes von Open-Source-Software ermöglicht die öffentliche Einsicht in die Art und Weise der Datenverarbeitung und -analyse. Dadurch unterliegt ein Verstoß gegen datenschutzrechtliche Bestimmungen der Aufsicht der Open-Source-Community. Auf dieser Grundlage ist es möglich, die Benutzer gemäß ihren individuellen Anforderungen zu konfigurieren. Im Falle von Closed-Source-Software ist der Quellcode hingegen nicht einsehbar, sodass die Nutzer lediglich auf die Datenschutzerklärung des Anbieters vertrauen können. Im Vergleich dazu zeichnet sich Open-Source-Software durch eine hohe Transparenz aus, die einer Kontrolle durch die Gemeinschaft unterliegt. Bei Closed-Source-Software hingegen ist eine eigenständige Kontrolle durch den Nutzer nicht möglich. Aus diesem Grund kann davon ausgegangen werden, dass der Datenschutz bei OSS vertrauenswürdiger ist als bei der traditionellen Softwareentwicklung.

[1] `https://www.ihk-muenchen.de/de/Service/Recht-und-Steuern/Vertragsrecht/mustervertraege/software-erstellungsvertrag.html`

Lizenz	Close-Source -Distribution nach Veränderung	Extension	Patent	Weitergabe enthält Quellcode?
MIT	Ja	Ja	Nein	Nein
3-Clause BSD	ja	Ja	Nein	Nein
Apache Ver. 2.0	Ja	Ja	Ja	Nein, aber Notizen für Änderungen
LGPL Ver. 3	Ja für dynamisch	Teilweise unter LGPL	Nein	Ja
MPL 2.0	Nein	Ja	Nein	Ja
GPL Ver. 3	Nein	Unter GPL	Nein	Ja

Bei der Entwicklung von Open-Source-Software wird jedoch nicht immer der Inhalt der Lizenzen beachtet. Eine Untersuchung der Lizenzen ergab Verstöße gegen die Lizenzbestimmungen im Fall der Luca-Anwendung, die bei Verwendung der GPL v3.0 zur Sammlung von Daten während einer Coronavirus-Pandemie hätte eingesetzt werden können. In diesem Kontext wird aus der LICENSE-Datei in Gitlab vom Luca-App-Entwicklungsteam zitiert.[2]

> Der Quellcode wird ausschließlich zum Zwecke der Betrachtung für persönliche, nicht-kommerzielle Zwecke auf nicht-ausschließlicher, nicht-unterlizenzierbarer und nicht-übertragbarer Basis bereitgestellt.
>
> Der Betrachter darf insbesondere nicht (und darf nicht gestatten, dass Dritte dies tun) mittelbar oder unmittelbar: (a) den Quellcode anderweitig vervielfältigen oder verarbeiten; (b) den Quellcode teilen oder öffentlich wiedergeben; (c) den Quellcode auf ein öffentliches oder verteiltes Netzwerk kopieren; (d) den Quellcode modifizieren, anpassen, übersetzen oder abgeleitete Werke erstellen, die auf dem Quellcode basieren; oder (e) den Quellcode in sonstiger Weise kommerziell oder nicht-kommerziell nutzen.

Der vorliegende Text aus dem LICENSE-Dokument von Luca App steht in wesentlichen Punkten im Widerspruch zu den Bedingungen der GPL v3.0, die von der Urheberin bzw. dem Urheber offenbar verwendet wird. In den Punkten a bis e wird seitens Luca App jede Form des Kopierens

[2] `https://gitlab.com/lucaapp/android/-/blob/master/LICENSE`

und Manipulierens des Quellcodes, das Teilen und öffentliche Zeigen des Quellcodes, das Kopieren des Quellcodes in ein öffentliches oder verteiltes Netzwerk, das Ändern, Anpassen, Übersetzen oder Erstellen abgeleiteter Werke auf der Grundlage des Quellcodes sowie jede Form der kommerziellen oder nichtkommerziellen Nutzung des Quellcodes abgelehnt. Diese Einträge verletzen sowohl den Inhalt der GPL v3.0-Lizenz als auch teilweise den Inhalt der Open-Source-Definition, wie sie von der Open Source Initiative vorgegeben wird.[28] Die genannten Einträge beschränken die Freiheit der Verbreitung, Veränderung usw. von Open-Source-Software. Folglich kann das Vorgehen von Luca App nicht als adäquater Versuch einer Open-Source-Software betrachtet werden.

6.1.3 Ziel und Motivation

Die Ziele und Motivationen werden in zwei Teilen analysiert, wobei zunächst das Entwicklungsteam und anschließend die Entwickler betrachtet werden.

Im Rahmen der traditionellen Softwareentwicklung ist das primäre Ziel des Entwicklungsteams die Erfüllung der Inhalte des Vertrags. Dies umfasst beispielsweise die Anforderungen in Bezug auf Abnahmetests, Sicherheit und weitere Elemente. Diese sind in der Regel im Vertrag festgehalten, sodass das Ziel bei der offenen traditionellen Softwareentwicklung relativ klar definiert ist. Im Gegensatz dazu muss Open-Source-Software attraktiv sein, um die Zahl der Teilnehmern zu erhöhen und mehr Feedback von den Nutzer:innen zu erhalten[12], damit sie sich weiterentwickeln kann.

Aus persönlicher Sicht lässt sich festhalten, dass in der traditionellen Softwareentwicklung Entwicklerinnen und Entwickler als Softwareingenieurinnen und Softwareingenieure angestellt werden. Die Softwareentwicklung stellt folglich ihr berufliches Tätigkeitsfeld dar. Selbstredend existieren darüber hinaus weitere Motivationen, beispielsweise persönliche Hobbys. Die persönlichen Motivationen, die den Antrieb für die Beteiligung an der Open-Source-Softwareentwicklung darstellen, sind dabei von Individuum zu Individuum unterschiedlich. Die Teilnehmerinnen und Teilnehmer sind sich einig, dass insbesondere Spaß, Altruismus und die Zusammenarbeit mit unbekannten Entwicklern die wichtigsten Motivationen sind.[14] In der Studie von Marco et al. stimmen 91 %, 85 % und 80 % der Befragten im Durchschnitt zu (oder stimmen stark zu), dass sie aus diesen Motiven heraus zu OSS beitragen.[14] Das Lernen und Wiedererlernen stellen ebenfalls wesentliche Faktoren dar. Bemerkenswerterweise waren 93 % der Befragten der Ansicht, dass sie einen Beitrag geleistet haben, da OSS ihnen die Möglichkeit bot, zu lernen und ihre Fähigkeiten zu erweitern.[14]

6.1.4 Plannen

In ähnlicher Weise wie bei der Open-Source-Softwareentwicklung und dem Requirements Engineering erfolgt auch hier eine Planung. Nach dem Erwerb von Kenntnissen im Bereich des Requirements Engineering sowie der Beobachtung des Bugzilla-Systems planen beide die nächste Entwicklungsphase, wobei sie die Anforderungen nach Prioritäten ordnen. Selbstverständlich sind auch in diesem Kontext Unterschiede festzustellen. In Bezug auf Open-Source-Software lässt sich festhalten, dass das Intervall zwischen den Versions-Updates als unsicherer einzustufen ist. Ein Beispiel für eine Open-Source-Software, deren Aktualisierungsrhythmus als unbeständig bezeichnet werden kann, ist Firefox[3]. Zwar erfolgt eine Aktualisierung einmal pro Monat, jedoch ist der Zeitpunkt der Veröffentlichung nicht präzise vorhersehbar. Es lässt sich annehmen, dass dies auf die Prioritätseinstufung zurückzuführen ist. So bedeutet beispielsweise die Priorität T1, dass sie in der nächsten Version veröffentlicht wird. Der Zeitplan für die Aktualisierung der Firefox-Version orientiert sich folglich vermutlich an der Erfüllung der genannten Anforderungen. In der traditionellen Softwareentwicklung werden jedoch in der Regel Einschränkungen wie ein klares Zeitbudget festgelegt, welches vom Projektmanager in einen detaillierteren Plan umgesetzt werden muss. Im Vergleich zu traditioneller Software, bei der die Zeitvorgaben im Vertrag festgelegt sind, wird bei Open-Source-Software eine schnelle Entwicklung mit häufigen inkrementellen Releases angestrebt, um die Attraktivität zu erhalten[12, 19].

6.1.5 Kosten

Die durchgeführten Interviews legen nahe, dass Entwicklerinnen und Entwickler, die als Mitglied des Kernentwicklungsteams an Open-Source-Projekten arbeiten, einen beachtlichen Zeit- und Arbeitsaufwand in die Entwicklung und Verwaltung ihrer Projekte investieren. Zeitmangel und mangelndes Interesse stellen ebenfalls wesentliche persönliche Gründe für das Scheitern bei der Open-Source-Softwareentwicklung dar[9]. Folglich sind Zeit und Motivation der Entwickler als Kostenfaktoren bei der Entwicklung von Open-Source-Software zu nennen. Im Gegensatz dazu erfordert die Entwicklung einer Software in einem Unternehmen einen hohen personellen und technischen Aufwand sowie zahlreiche komplexe Vorgänge. Es lässt sich festhalten, dass die Entwicklung von Open-Source-Software im Vergleich zur traditionellen Software-Entwicklung in Form von kostenloser und haftungsfreier Software erfolgt.

[3] `https://www.mozilla.org/en-US/firefox/releases/` Accessed: 2024-06-24

6.2 Während des Projekts

Im vorliegenden Abschnitt erfolgt eine Analyse der Unterschiede im Entwicklungsprozess zwischen Open-Source- und traditioneller Software. Der Fokus liegt hierbei auf dem Anwendungsfeld, der Implementierung, der Validierung sowie der Verifizierung.

6.2.1 Anwendungsfeld

Innerhalb der traditionellen Softwareentwicklung manifestiert sich das Problem divergierender Verständnisse zwischen dem Entwicklungsteam und dem Fachgebiet, welches auch als „Symmetry of Ignorance" bezeichnet wird. Bei der Entwicklung von Open-Source-Software besteht die Möglichkeit, dass die Hauptentwickler ebenfalls über das entsprechende Fachwissen verfügen[25]. Dies führt zu einer Reduktion der Kommunikationskosten. Des Weiteren wurde in den geführten Interviews ersichtlich, dass die Fachleute bei der Formulierung von Anforderungen die entsprechende Geschäftslogik vorlegen, um eine adäquate Analyse durch die Entwickler zu gewährleisten. Im Rahmen der Beobachtung von Firefox wurde festgestellt, dass das Entwicklerteam von Firefox ein vollständiges und klares Hilfedokument sowie eine Schnittstelle für die Community-Entwickler verfasst hat. Dies kann auch Fachleuten auf dem Gebiet der Konfiguration von Open-Source-Software und anderen Arbeiten zugutekommen.

6.2.2 In Team

Im Rahmen der traditionellen Softwareentwicklung erfolgt eine klare Abgrenzung zwischen dem Entwicklungsteam und der Außenwelt. Ein Beispiel für eine solche Aufteilung ist das Scrum-Verfahren, bei dem das Scrum-Team aus einem Scrum Master, einem Product Owner und einem Entwicklungsteam besteht. Die übrigen involvierten Akteure werden als „extern" bezeichnet. Im Kontext der Open-Source-Entwicklung ist eine klare Abgrenzung nicht gegeben. Die Entwicklungsingenieure von Open-Source-Software nutzen die Software zudem selbst. Die Anforderungen an die Software werden nicht ausschließlich von den Nutzern der Software definiert, sondern auch von Mitwirkenden in der Open-Source-Gemeinschaft, die über entsprechende Entwicklungsfähigkeiten oder Fachwissen verfügen.[25] Zudem fließen auch Vorschläge des Kernentwicklungsteams in die Definition der Anforderungen ein. Aus den durchgeführten Interviews und Beobachtungen lässt sich ableiten, dass das Kernentwicklungsteam von Firefox erkannt hat, dass die Kernentwicklergruppe neben der eigentlichen Entwicklungsarbeit auch den erhaltenen Code einchecken muss. Darüber hinaus sind einige strukturelle Arbeiten erforderlich, beispielsweise die Priorisierung von Anforderungen.

Der Autor postuliert, dass das Konzept eines Entwicklungsteams bei

Open-Source-Software mit dem Konzept der „kleinschen Flasche“ vergleichbar ist, da es keine eindeutige Abgrenzung zwischen intern und extern gibt. Ein Benutzer, der beispielsweise bei der Nutzung von Open-Source-Software eine Schwachstelle entdeckt und diese auf der entsprechenden Plattform meldet, leistet damit einen Beitrag zur Software, der als Mitwirkung betrachtet werden kann. Ein Entwickler, der die Software als Benutzer nutzt, verhält sich ähnlich. Das Kernteam kann in diesem Kontext als die „Flasche“ selbst von der „kleinschen Flasche“ betrachtet werden. Letztere stellt einen Schlüsselfaktor für die Steuerung der Richtung der Software dar.

6.2.3 Implementierung

In der traditionellen Softwareentwicklung existiert eine Vielzahl an Praktiken, die eine Steigerung der Effizienz und Qualität der Entwicklung zum Ziel haben. Dazu zählen beispielsweise Scrum, eXtreme Programming und Pair Programming, welche im Kontext der agilen Entwicklung Anwendung finden. Scrum wird insbesondere in der traditionellen Softwareentwicklung eingesetzt, um die Effizienz, Flexibilität und Kundenzufriedenheit bei der Softwareentwicklung zu erhöhen[4]. Das tägliche Scrum-Meeting dient der Analyse der aktuellen Situation sowie der Identifikation potenzieller Herausforderungen. Zudem wird ein On-Site-Customer eingesetzt, um eine effektive Kommunikation mit den Kunden sicherzustellen[4]. Die Kommunikation stellt einen wesentlichen Aspekt in der Open-Source-Softwareentwicklung dar. Dabei werden Mailinglisten von Entwicklern häufig genutzt, um Architekturinformationen zu teilen und auszutauschen[31, 8].

Im Rahmen der durchgeführten Interviews wurde ersichtlich, dass in den Unternehmen der Befragten Codevorlagen für die Entwickler existieren, sodass sich der Stil des Codes nicht allzu sehr unterscheidet. Bei der Open-Source-Entwicklung sind die Entwickler jedoch unterschiedlichen Hintergründen, Muttersprachen und Bildungserfahrungen ausgesetzt, was die Entwicklung von Open-Source-Software erschwert. Dies führt zu einer divergierenden Entwicklung des Stils des Codes.

Die Durchführung von Reviews stellt eines der wesentlichen Instrumente zur Gewährleistung der Qualität von Open-Source-Software dar[17], wie bereits in der vorherigen Betrachtung des Firefox-Browsers erörtert wurde. Auf dieser Grundlage bietet Firefox eine Vielzahl von Tests an, darunter den mochitest, ein Firefox-Frontend für den Addressbar-Test. Der Zweck dieses Tests besteht in der Prüfung des Firefox-Browsers selbst. Darüber hinaus stehen in Firefox zahlreiche ähnliche automatisierte Tests zur Verfügung[4]. Die Durchführung automatisierter Tests ist erforderlich, da die Entwickler aus verschiedenen Zeitzonen stammen und Open-Source-Software in der Regel rund um die Uhr entwickelt wird.[36]

[4] `https://firefox-source-docs.mozilla.org/testing/automated-testing/index.html`

6.2.4 Validierung & Verifizierung

Die Validierung und Verifizierung traditioneller Software erfolgt durch die Bereitstellung von Spezifikationen, Prototypen, Visionsvideos und ähnlicher Materialien für den Kunden. In Open-Source-Software hingegen existiert die Rolle des Kunden nicht, sodass der in traditioneller Software verfügbare Ansatz nicht anwendbar ist. In der von Noll et al. durchgeführten Fallstudie zu OpenEMR konnte festgestellt werden, dass die in den Forumskommentaren aufgeführten Anforderungen effektiv validiert wurden[25]. Die verbleibenden, nicht validierten Anforderungen wurden als Wunschliste auf Sourceforge.net hochgeladen. In der Fallstudie von Noll et al. zu Firefox wurde festgestellt, dass die von den Entwicklern vorgeschlagenen Anforderungen nur selten durch einen formalen Prozess validiert wurden[24] . Bei den eigenen Beobachtungen der Autoren zu Firefox fand die Validierung im Kommentarbereich der Foren statt, wobei die meisten Anforderungen von den Kernentwicklern vorgeschlagen und diesen auch selbst zugewiesen wurden. Dies reduziert zudem die Kosten der Kommunikation.

6.3 Nachdem Projekt

In diesem Abschnitt werden die Unterschiede zwischen Open-Source-Software und herkömmlicher Software im Hinblick auf die Fertigstellung nach der Entwicklung analysiert. Der Schwerpunkt liegt auf der Dokumentation, Aspekten der Softwaresicherheit und dem Anforderungsänderungsmanagement.

6.3.1 Dokumentation

Die Aufgabe der Dokumentation im Rahmen des Requirements Engineering besteht in der Strukturierung und klar verständlichen Beschreibung der zuvor erfassten Anforderungen. Dies ist erforderlich, um eine Kompatibilität mit der anschließenden Verifizierung und Validierung sicherzustellen. Zudem wird dadurch das Verständnis der Anforderungen durch das Entwicklerteam erleichtert. Als gängige Methoden können beispielsweise UML-Diagramme genannt werden. Bei der Beobachtung des Browsers Firefox lässt sich feststellen, dass zu einem gegebenen Zeitpunkt eine Vielzahl von Dokumenten existiert. Dazu zählen unter anderem Foren, die der Verifizierung und Validierung als dynamische Dokumentation dienen[34], sowie Anleitungen für Entwickler und Wiki-Dokumentationen, in denen das Schreiben von Testspezifikationen erläutert wird. In den meisten OSS-Projekten existieren keine vertraglichen Verpflichtungen und keine formale Dokumentation der Anforderungen[7].

Die Autoren sind der Auffassung, dass sich der Unterschied aus dem divergierenden Zweck der Dokumentation ergibt. Während diese in Firefox

den Prozess des Verstehens und der Kategorisierung der Anforderungen dokumentiert, wird gleichzeitig der transparente Charakter des Open-Source-Engineerings bewahrt. Der Zweck der Dokumentation ist hier nicht mit der geschlossenen, formalen und geschäftsmäßigen Dokumentation der traditionellen Softwareentwicklung vergleichbar. Zudem ist die Dokumentation in Open-Source-Software und die Kommunikation in Foren informeller und deklarativer Natur[7, 1], was ebenfalls einen Unterschied zur Dokumentation in der traditionellen Softwareentwicklung darstellt.

6.3.2 Sicherheit

In vorangehenden Untersuchungen des Browsers Firefox haben wir festgestellt, dass Open-Source-Software potenzielle Sicherheitsrisiken aufweist, die durch die Transparenz des Quellcodes und der Dokumentation schnell identifiziert werden können. Wie bereits im ersten Absatz dargelegt, stellen Sicherheitsprobleme wie xz-Utils jedoch nach wie vor eine Bedrohung dar. Die von dem neu ernannten Administrator JiaT75 hochgeladene Sicherheitslücke sowie damit zusammenhängende Probleme wurden im Rahmen des Interviews thematisiert. Die befragten Experten gaben zu Protokoll, dass es keine leichte Aufgabe ist, sicherzustellen, dass die Administratoren des Kernteams keine versteckten Hacker sind. Langjähriges Vertrauen ist nicht immun gegen die Möglichkeit, dass Administratoren schlechte Absichten haben. In der traditionellen Entwicklung kann dies aufgrund des kommerziellen Charakters und der Verträge zum Teil vermieden werden.

6.3.3 Management

Der Begriff „Management“ bezeichnet in diesem Kontext das Management von Änderungen. Dazu zählt das Requirements Engineering bei der Änderung und die Verfolgung von Anforderungen. Bei der Beobachtung des Firefox-Projekts wurde festgestellt, dass der Inhalt des Change Managements zusammen mit neuen Anforderungen in Bugzilla eingestellt wird. Die Darstellung der zusammenhängenden Anforderungen erfolgt in Form von Bäumen und Diagrammen, wodurch den Entwicklern die Auffindung von Zusammenhängen zwischen Änderungen sowie der Zugriff auf ähnliche Änderungen erleichtert wird. Die Verwaltung setzt Triage ein, um die Entwickler richtig einzusetzen und sicherzustellen, dass jede Anforderung korrekt kategorisiert wird. Im Forum können Gründe für nachzuerfassende Änderungen an Anforderungen über Hyperlinks im Beziehungsbaum schnell aufgegriffen werden. Unter diesen Gesichtspunkten erbringt große Open-Source-Software wie Firefox bessere Leistungen beim Change Management und der Verfolgung von Anforderungen im Requirements Engineering.

6.4 Allgemein & Eigenschaften

Im Rahmen der Untersuchung werden die Herausforderungen, mit denen Open-Source-Software im Requirements Engineering konfrontiert ist, umfassend analysiert. Dabei werden die Charakteristika von Open-Source-Software im Vergleich zur traditionellen Entwicklung beleuchtet, die jeweiligen Vorteile herausgearbeitet und schließlich der Entwicklungsstil analysiert, um festzustellen, ob Möglichkeiten für einen gegenseitigen Lernprozess und eine Zusammenarbeit bestehen.

6.4.1 Herausforderung

Obgleich sich eine Vielzahl erfolgreicher, großer und geordneter Betreiber von Open-Source-Software etabliert hat, sind die Herausforderungen im Requirements Engineering und in der Qualitätssicherung für Open-Source-Software nach wie vor evident. Wie bereits dargelegt, erfolgt die Entwicklung von Open-Source-Software rund um die Uhr in verschiedenen Zeitzonen.[36] Die Anforderungen an automatisierte Tests haben sich dahingehend verändert, dass sie dazu dienen, die Arbeitsbelastung der Reviewer zu reduzieren und die Geschwindigkeit der Rückmeldungen zu erhöhen. Die Implementierung neuer Funktionen bedingt die Durchführung neuer Tests zu deren Verifikation. Daher sind Aktualisierungen der automatisierten Tests ebenso von hoher Relevanz. Zudem hat der Autor bei der eigenen Entwicklung kleiner Open-Source-Software festgestellt, dass die Dokumentation der Schnittstellen, welche die Open-Source-Software bereitstellt, sowie der Code selbst nicht übereinstimmen. Folglich stellt die rechtzeitige Aktualisierung der Handbücher für die Entwickler einen wichtigen Faktor dar, um deren Motivation aufrechtzuerhalten.

In dem vorausgehenden Kapitel 3 wurde bereits darauf verwiesen, dass mehr als 50 % der befragten Entwickler Unterschiede in Bildung, Sprache und Kultur als eine der Schwierigkeiten bei der Entwicklung von Open-Source-Software ansehen[19]. Da es bei Open-Source-Software keine Einschränkungen hinsichtlich des Hintergrunds der Entwickler gibt, kann die Überprüfung der Arbeit durch die divergierende Codequalität erschwert werden. Aus den Interviews ging hervor, dass im Gegensatz zu Unternehmen, die ein vorgeschriebenes Codeformat haben, unterschiedliche Codestile Teil des Prozesses sind, den die Befragten bei der Verwaltung ihrer Projekte zu bewältigen haben. Dies kann potenziell die Notwendigkeit zur Neuschreibung des Codes mit sich bringen.

An anderer Stelle äußerte etwa ein Drittel der Entwickler, dass sie Schwierigkeiten bei der Bewältigung von Konflikten, der Prioritätensetzung und der Analyse von Anforderungen haben[19]. In Firefox wird die Priorität als Entscheidungskriterium herangezogen, um zu bestimmen, wann ein Update für eine Anforderung veröffentlicht wird. Allerdings fehlt eine genaue

Definition der Priorität und Schweregrad.Im Folgenden findet sich die Original-Firefox-Dokumentation, welche die Definitionen der Prioritäten P1 und P2 sowie der Schweregrade S1, S2 und S3 beinhaltet.

> P1 Fix in the current release cycle
>
> P2 Fix in the next release cycle or the following (nightly + 1 or nightly + 2)
>
> S1 (Catastrophic) Blocks development/testing, may impact more than 25% of users, causes data loss, likely dot release driver, and no workaround available
>
> S2 (Serious) Major functionality/product severely impaired or a high impact issue and a satisfactory workaround does not exist
>
> S3 (Normal) Blocks non-critical functionality or a work around exists[5]

Bei der Definition des Schweregrads in S1 fällt auf, dass bestimmte Situationen oder Fehlertypen sowie quantitative Beschreibungen berücksichtigt werden. Demgegenüber mangelt es in S2 und S3 an präzisen Definitionen von Begriffen wie „MAJOR", „SEVERELY" und „NON-CRITICAL". Diese können aufgrund der subjektiven Entscheidung des Triage-Owners ungenau sein. Allerdings birgt genau dieser Aspekt eine gewisse Schwierigkeit, denn der Versuch, diese Themen exakt zu beschreiben, ist mit einigen Herausforderungen verbunden. Dies kann dazu führen, dass das Dokument eine beträchtliche Länge und Spezialisierung aufweist, was wiederum die Lesbarkeit und Verständlichkeit beeinträchtigt.

6.4.2 Eigenschaften & Unterschiede

In der vorliegenden Untersuchung wurden bereits eine Vielzahl von Merkmalen und Unterschieden zwischen traditioneller und Open-Source-Software erörtert. Im Folgenden sollen daher einige wesentliche Merkmale und ihre Implikationen näher beschrieben werden.

Aus Perspektive des Requirements Engineering stellt das Fehlen vertraglicher Zwänge sowie die Abwesenheit der Kundenrolle einen der signifikantesten Unterschiede dar. Da es keine vertraglichen Einschränkungen gibt, existiert bei Open-Source-Software kein extern festgelegtes Zeitbudget, keine vorgeschriebenen Funktionen, die fertiggestellt werden müssen, keine Endkontrolle des Abnahmetests und so weiter. Dies gibt dem Entwicklungsteam die Freiheit, über die Funktionen der Software und das Entwicklungstempo zu entscheiden. Da es keine Rolle des Kunden gibt, kann das Entwicklungsteam die Ziele und die Richtung der Änderungen der Anforderungen während der Validierung und Verifizierung selbst bestimmen.

[5] `https://wiki.mozilla.org/BMO/UserGuide/BugFields#priority`

Des Weiteren sind bei der Entwicklung von Open-Source-Software die Entwickler auch die Benutzer, was eine bessere Verständnisbasis für die vorgeschlagenen Anforderungen schafft. Gleichzeitig werden viele der Anforderungen von den Entwicklern selbst vorgeschlagen, was nicht nur die Kosten für die Kommunikation reduziert, sondern auch die Möglichkeit von Missverständnissen verringert, da der Kommunikationsprozess zwischen dem Kunden und dem Projekteigner im Vergleich zur Arbeit an den Anforderungen in traditioneller Software entfällt.

Ein weiteres Merkmal von Open-Source-Software ist die Transparenz von Dokumentation und Quellcode. Open-Source-Software stellt öffentlich zugänglichen Quellcode und Entwicklungshandbücher zur Verfügung, um Rückmeldungen zu Problemen und Codebeiträge der Open-Source-Gemeinschaft zu erleichtern. Darüber hinaus wird der Prozess des Verstehens und der Verwaltung von Anforderungen in den Analysen öffentlich gemacht. Dies ist bei der Softwareentwicklung für geschäftliche Zwecke nur schwer zu erreichen.

6.4.3 Vorteile und Nachteile

Zusammenfassend lässt sich festhalten, dass die Vorteile von Open-Source-Softwar in der größeren Freiheit, den geringeren Kommunikationsaufwand und der Tatsache liegen, dass durch den transparenten und offenen Code Sicherheitsprobleme schnell erkannt und an die Gemeinschaft zurückgemeldet werden können. Des Weiteren besteht die Möglichkeit, dass Anregungen aus der Community in den Entwicklungsprozess von Open-Source-Software einfließen. Ein weiterer Vorteil von Open-Source-Software ist ihre Vielfalt[37]. Für die Benutzer ist Open-Source-Software oft kostenlos, was ebenfalls als Vorteil zu nennen ist.

Auch Open-Source-Software weist gewisse Nachteile auf, insbesondere im Vergleich zur traditionellen Softwareentwicklung. Da Open-Source-Software-Ingenieure in erster Linie durch ihre eigenen Interessen motiviert sind, unterliegen sie keinerlei arbeitsplatz- oder vertragsbezogenen Zwängen. In einer Befragung wurden Zeitmangel und mangelndes Interesse als die beiden größten nicht umweltbedingten Faktoren identifiziert, die ein Stoppen von Open-Source-Software zur Folge hatten[19]. Dies bedeutet, dass eine kleine Open-Source-Software aufgrund von Zeitproblemen eines Hauptentwicklers plötzlich nicht mehr aktualisiert werden kann. Darüber hinaus ist die wirtschaftliche Quelle von Open-Source-Software nicht nachhaltig[26], was für die Entwickler kleiner Software eine Herausforderung darstellen wird.

6.4.4 Kombinationsmöglichkeit & Kompatibilität

Im Rahmen der durchgeführten Interviews konnte festgestellt werden, dass es zwei Prozesse der Zusammenarbeit zwischen Open-Source-Software und

traditioneller Entwicklung gibt. Einerseits stellt kommerzielle Software einen Teil ihrer Boards der Open-Source-Community zur Verfügung und verwaltet diesen Teil der Open-Source-Entwicklung. Andererseits wird Open-Source-Software von Unternehmen erworben. Bei der ersten Möglichkeit stellt das Unternehmen der Gemeinschaft Open-Source-Software zur Verfügung und verkauft kostenlose Dienste, um die Innovation zu steigern und die Entwicklungskosten zu senken[2].

Kapitel 7

Diskussion

Im Folgenden wird zunächst die Forschungsfrage unter Zuhilfenahme der Resultate der Fallstudie, des Interviews sowie des Modells beantwortet. Darüber hinaus erfolgt eine Interpretation der Ergebnisse im Kontext dieser Arbeit. Zudem werden die Limitationen dieser Arbeit analysiert.

7.1 Beantwortung von Forschungsfragen

RQ1: Was sind die Unterschiede zwischen dem traditionellen Requirements Engineering und der Open-Source-Softwareentwicklung?

Im Rahmen der Untersuchung der Open-Source- und der traditionellen Softwareentwicklung in Kapitel 6 konnten mehrere zentrale Unterschiede festgestellt werden, die zu Unterschieden zwischen Open-Source- und traditionellen Entwicklungsprozessen führen.

Open-Source-Software ist nicht vertraglich gebunden. Verträge sind hier Softwarebestellungen zwischen Unternehmen und Arbeitsverträge zwischen Unternehmen und Entwicklern. Das Fehlen von Verträgen zwischen Unternehmen ermöglicht es den Entwicklern von Open-Source-Software, frei über die Richtung der Entwicklung zu entscheiden. Bei Open-Source-Software werden die Eigenschaften der Open-Source-Software durch das Kernentwicklungsteam der Software bestimmt. Da es kein Zeitbudget gibt, kann das Open-Source-Software-Entwicklungsteam auch das Tempo der Aktualisierungen selbst bestimmen. Der Nachteil des Fehlens eines kommerziellen Vertrags ist das Fehlen einer stabilen Finanzierungsquelle Das Gleiche gilt für Entwickler von Open-Source-Software, da es keinen Arbeitsvertrag gibt.

Die Stakeholder von Open-Source-Software unterscheiden sich von denen der traditionellen Softwareentwicklung, bei der es keine Kunden oder ähnliche Abteilungen wie Marketing und Management im ursprünglichen Unternehmen gibt. Darüber hinaus ist die Identität der Stakeholder bei Open-Source-Software nicht eindeutig, und die Bezeichnungen Entwickler

und Benutzer können auf ein und dieselbe Person zutreffen. Dies führt zu vielen Unterschieden bei Phasen wie Negotiation und Validierung im Requirements Engineering.

Ein weiterer wesentlicher Aspekt ist die Transparenz der Open-Source-Softwareentwicklung. Die öffentliche Zugänglichkeit des Quellcodes und der Dokumentation erlaubt es die Mitarbeitern der Open-Source-Gemeinschaft, sich an der Entwicklung oder Überprüfung zu beteiligen. Zudem macht transparenter Quellcode Open-Source-Software vertrauenswürdiger in Bezug auf die Datensicherheit und ermöglicht eine private Konfiguration, was ebenfalls zu den Vorteilen von Open-Source-Software zählt.

Die obige drei Punkte stellen aus Sicht des Autors die wesentlichen Unterschiede zwischen der Entwicklung von Open-Source-Software und traditioneller Software dar. Selbstverständlich existieren darüber hinaus weitere Einflussfaktoren wie beispielsweise die Beteiligung der Open-Source-Community, die zu spezifischen Abweichungen führen können. Allerdings erlangen die genannten drei Aspekte eine übergeordnete Bedeutung für die Anwendung von Requirements Engineering im Kontext von Open-Source-Software.

RQ2: Wie werden bei der OSS-Entwicklung Praktiken des Requirements Engineering behandelt, die sich schwer in OSS implementieren lassen?

Die Umsetzung der Scrum-Praktiken in Open-Source-Software ist aufgrund der globalen Verteilung der Entwickler und der damit einhergehenden Zeitverschiebungen mit erheblichen Schwierigkeiten verbunden. Dies führt dazu, dass eine gemeinsame Besprechung der aufgetretenen Schwierigkeiten selten möglich ist. Aus ähnlichen Gründen ist es ebenfalls nicht möglich, Praktiken wie Brainstorming oder Workshops während des Erhebungsprozesses der Anforderungen durchzuführen.

Des Weiteren wurde bereits in der vorhergehenden Analyse dargelegt, dass die Dokumentation der Priorisierung und Schweregradbestimmung ungenau ist. Auch diesbezüglich sind Schwierigkeiten zu vermerken. In der traditionellen Entwicklung obliegt es dem Kunden, Prioritäten zu definieren. In der Open-Source-Entwicklung erfolgt die Festlegung von Priorität und Schweregrad eher subjektiv. Des Weiteren ist von Relevanz, ob das Kernentwicklungsteam ein gemeinsames Verständnis sowie eine gemeinsame Einschätzung der Prioritäten aufweist.

RQ3: Welche spezifischen Methoden und Vorgehensweisen werden bei der Behandlung von Anforderungen in der Open-Source-Softwareentwicklung angewendet?

Im Rahmen einer Untersuchung des Firefox-Entwicklungsteams wurde ersichtlich, dass Bugzilla als Verwaltungssystem dient, welches eine Vielzahl von Aufgaben im Requirements Engineering erfüllt. Die Bearbeitung der Dokumentation, die Analyse der Anforderungen, das Change Management sowie weitere Aufgaben werden in Bugzilla parallel durchgeführt. Auf jeder Anforderungsseite befindet sich ein Forum, wobei die Kommunikation

im Forum auch als Aufzeichnung der Softwaredokumentation sowie der Analyse der Anforderungen dient. Foren werden auch in anderen Bereichen der Open-Source-Softwareentwicklung als Werkzeuge eingesetzt. Zudem ist Bugzilla selbst eine Open-Source-Software und steht zahlreichen Open-Source-Projekten zur Verfügung. Insgesamt kann ein Forum als ein Merkmal von Open-Source-Projekten betrachtet werden.

7.2 Limitationen

Das Interview in 4 unterliegen einigen Einschränkungen. Da der Befragte nur ein Entwickler ist, sind viele seiner Ansichten persönlich und nicht mit akademischer Genauigkeit definiert. An einigen Stellen verfügt er auch nicht über spezifische Erfahrungen, sondern hat seine eigenen Annahmen zum Thema gemacht. Außerdem hat der Befragte keine Erfahrung in der Entwicklung von Open-Source Software, sondern nur als einer der Manager eines Open-Source-Projekts. Daher ist auch sein Wissen über Open Source Software begrenzt. Außerdem wurde nur ein Interview geführt und seine Meinung spiegelt nicht die Meinung aller Entwickler über Open Source Software wider.

Im Rahmen der Beobachtungen von Firefox verfügte der Autor nicht über die erforderlichen Ressourcen, um Elemente wie die interne Kommunikation des Kernteams zu beobachten. Infolgedessen konnte durch diese Beobachtung kein Einblick in das Verständnis des Kernteams für Anforderungen und Entscheidungsfindung gewonnen werden.

Der Umfang der Bachelorarbeit sowie die zeitlichen Beschränkungen lassen vermuten, dass die im Modell enthaltenen Aussagen nicht durch entsprechende Experimente oder Arbeiten gestützt werden. Zudem ist anzumerken, dass nur ein Interviewpartner zur Verfügung stand, was die Ergebnisse des Interviews beeinträchtigt haben könnte.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Um das Requirements Engineering in Open Source Software zu analysieren, wurden mehrere Artikel recherchiert und verglichen. Außerdem gibt es ein Interview mit einem Entwickler, der einige Erfahrung in der Verwaltung von Open-Source-Software hat, und eine Beobachtung von Firefox, einer großen Open-Source-Software.

Im Rahmen des Interviews wurde ein Entwickler zu seinem Wissen und seinen Ansichten bezüglich der Entwicklung von Open-Source-Software befragt. Der Befragte gab Auskunft über seine Erfahrungen mit der Verwaltung der Open-Source-Softwareentwicklung und die Schwierigkeiten, die den von ihm bekannten Personen dabei begegneten.

Im Rahmen der Betrachtung von Firefox erfolgt eine Analyse des Umgangs mit Anforderungen im Open-Source-Software-Engineering anhand der Schritte des Requirements Engineering. Darauf aufbauend wird ein Vergleich mit dem Umgang mit Anforderungen in der traditionellen Softwareentwicklung angestellt.

Die Auswewdrtung von Interviews und Beobachtungen erfolgte in Kombination mit der zuvor gelesenen Literatur, um das Requirements Engineering in Open-Source-Software anhand eines Modells zu analysieren und mit der traditionellen Softwareentwicklung zu vergleichen. Im Folgenden werden die Merkmale von Open-Source-Software sowie die Vor- und Nachteile von Open-Source-Software im Vergleich zu traditioneller Software dargestellt.

8.2 Ausblick

In späteren Studien kann eine erneute Validierung des Modells erfolgen. Des Weiteren besteht die Möglichkeit, Experimente durchzuführen, um

die Praktiken bei Open-Source-Software in Bezug auf ihre Effizienz zu quantifizieren. Die Beobachtungen zu großflächiger Open-Source-Software sind derzeit zahlreicher und es gibt mehr Untersuchungen zu erfolgreicher und reibungslos funktionierender Software. Durch die Durchführung von Beobachtungsstudien zu kleiner und mittelgroßer Open-Source-Software kann ein umfassenderes Verständnis der Stärken und Schwächen der Open-Source-Entwicklung erreicht werden. Zudem kann ein vollständigeres Modell entwickelt werden, wenn mehr Entwickler mit Erfahrung sowohl in der Open-Source-Entwicklung als auch in der Closed-Source-Entwicklung und akademische Experten in verwandten Bereichen befragt werden können. Im Rahmen des Modells sollten zudem Untersuchungen zum Teammanagement sowie zur Ausrichtung des GUI- und Softwarearchitekturdesigns durchgeführt werden. Diese Untersuchungen konnten jedoch aus Zeit- und Genehmigungsgründen nicht abgeschlossen werden. Bezüglich des Teammanagements ist zu berücksichtigen, dass in der Open-Source-Softwareentwicklung entsprechende Genehmigungen erforderlich sind, um den spezifischen Managementprozess zu beobachten. Zudem gibt es weniger Beispiele für GUI- und Softwaredesign von Grund auf. Es wäre wünschenswert, wenn entsprechende Untersuchungen durch eine andere Person durchgeführt werden könnten.

Anhang A

Interviewsleitfaden

Interviewsleitfaden:
Einführungsfragen:
1. Welche Teil von Software Entwicklung beschäftigst du meistens mit?
2. Hast du früher mit Open Source Software Projekt beschäftigst?
3. Was machst du meist Kopfschmerzen in deine Arbeit außer Implementationsphase? Sollte es einfacher sein, falls die Projekt in OSS sein?

Hauptteil:
1. Hast du die Fehlern durch Missverständnisse von der Anforderung erfahren?
2. Was gibt es in der Softwaredokument von der Arbeit?
3. Wie handelst du die Abhängigkeit in verschiedene Aufgaben?
4. Falls es Konflikt mit andere Modul gibt's, was machst du damit?
5. Glaubst du dass es schwierig wird in OSS Entwicklung?
6. Hast du früher mit OSS oder OSS Entwickler gearbeitet?
7. Was wäre die Vor- und Nachteile, wenn du dein aktuelle Projekt in Open Source arbeiten?
8. Was ist dein Motivation in deiner OSS Projekt Erfahrung?
9. Gibt's ein Admin für die Projekt?
9.1. Falls ja, was hast du mit dem Admin kommuniziert? Wie kommunizierst du mit dem Admin?
9.2. Falls du der Admin bist, gibt's Schwierigkeit bei der Administration in der Projekt?
10. Glaubst du dass alle Dokument soll veröffentlichen soll?
11. Hast du (in deiner Arbeit) OSS verwendet? Wie hält du über die Sicherheit von OSS?
12. Was denkst du, wenn die OSS Entwickler in ein andere Domäne die er nicht kennt arbeiten? Z.B. medizinische Software.

Schlussfragen:
1. Was denkt du als die größte Schwierigkeit in OSS Entwicklung in dem Bereich Anforderung?
2. Woran liegt die größte Unterschied zwischen OSS und gewerbliche Software?
3. Gibt's Möglichkeit, dass in gewerbliche Software Entwicklung auch Einsatz von OSS Entwicklung hat? Was ist deine Meinung dazu?

Abbildung A.1: Interviewsleitfaden

Literaturverzeichnis

[1] T. A. Alspaugh and W. Scacchi. Ongoing software development without classical requirements. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 165–174. IEEE, 2013.

[2] M. Andersen-Gott, G. Ghinea, and B. Bygstad. Why do commercial companies contribute to open source software? *International Journal of Information Management*, 32(2):106–117, 2012.

[3] E. Börger, B. Hörger, D. Parnas, and D. Rombach. Requirements capture, documentation, and validation. In *Dagstuhl Seminar*, volume 99241, 1999.

[4] P. Bourque and R. E. Fairley, editors. *Guide to the Software Engineering Body of Knowledge, Version 3.0*. IEEE Computer Society, 2014.

[5] Bugzilla.org. Installation list. Accessed: 2024-07-01.

[6] Bundesamt für Sicherheit in der Informationstechnik. Kritische backdoor in xz für linux. `https://www.bsi.bund.de/SharedDocs/Cybersicherheitswarnungen/DE/2024/2024-223608-1032.pdf?__blob=publicationFile&v=5`, 2024.

[7] J. W. Castro Llanos and S. T. Acuña Castillo. Differences between traditional and open source development activities. In *Product-Focused Software Process Improvement: 13th International Conference, PROFES 2012, Madrid, Spain, June 13-15, 2012 Proceedings 13*, pages 131–144. Springer, 2012.

[8] S.-W. Chou and M.-Y. He. Understanding oss development in communities: the perspectives of ideology and knowledge sharing. *Behaviour & Information Technology*, 30(3):325–337, 2011.

[9] J. Coelho and M. T. Valente. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, pages 186–196, 2017.

[10] J. E. Corbly. The free software alternative: Freeware, open source software, and libraries. *Information Technology and Libraries*, 33(3):65–75, Sep. 2014.

[11] K. Crowston, K. Wei, Q. Li, and J. Howison. Core and periphery in free/libre and open source software team communications. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 6, pages 118a–118a, 2006.

[12] A. C. B. G. da Silva, G. de Figueiredo Carneiro, A. C. M. de Paula, M. P. Monteiro, and F. B. e Abreu. Agility and quality attributes in open source software projects release practices. In *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 107–112. IEEE, 2016.

[13] V. Del Bianco, L. Lavazza, V. Lenarduzzi, S. Morasca, D. Taibi, and D. Tosi. A study on oss marketing and communication strategies. In *Open Source Systems: Long-Term Sustainability: 8th IFIP WG 2.13 International Conference, OSS 2012, Hammamet, Tunisia, September 10-13, 2012. Proceedings 8*, pages 338–343. Springer, 2012.

[14] M. Gerosa, I. Wiese, B. Trinkenreich, G. Link, G. Robles, C. Treude, I. Steinmacher, and A. Sarma. The shifting sands of motivation: Revisiting what drives contributors in open source. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1046–1058. IEEE, 2021.

[15] P. Groen. Openemr continues to grow in popularity and use. `https://www.openhealthnews.com/hotnews/openemr-continues-grow-popularity-and-use`, Dec. 2012. Retrieved 2012-12-20.

[16] M. Jarke, P. Loucopoulos, K. Lyytinen, J. Mylopoulos, and W. Robinson. The brave new world of design requirements. *Information Systems*, 36(7):992–1008, 2011.

[17] A. Khatami and A. Zaidman. State-of-the-practice in quality assurance in java-based open source software development. *Software: Practice and Experience*, 2024.

[18] H. M. Kiran and Z. Ali. Requirement elicitation techniques for open source systems: a review. *International Journal of Advanced Computer Science and Applications*, 9(1), 2018.

[19] J. Kuriakose and J. Parsons. How do open source software (oss) developers practice and perceive requirements engineering? an empirical study. In *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 49–56. IEEE, 2015.

[20] J. Kuriakose and J. Parsons. How do open source software (oss) developers practice and perceive requirements engineering? an empirical study. In *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 49–56, 2015.

[21] M. Lins, R. Mayrhofer, M. Roland, D. Hofer, and M. Schwaighofer. On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from xz. *arXiv preprint arXiv:2404.08987*, 2024.

[22] National Vulnerability Database. CVE-2024-3094 Detail , 05 2024.

[23] A. Nguyen Duc, D. S. Cruzes, C. Ayala, and R. Conradi. Impact of stakeholder type and collaboration on issue resolution time in oss projects. In *IFIP International conference on open source systems*, pages 1–16. Springer, 2011.

[24] J. Noll. Requirements acquisition in open source development: Firefox 2.0. In *Open Source Development, Communities and Quality: IFIP 20 th World Computer Congress, Working Group 2.3 on Open Source Software, September 7-10, 2008, Milano, Italy 4*, pages 69–79. Springer, 2008.

[25] J. Noll and W.-M. Liu. Requirements elicitation in open source software development: a case study. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 35–40, 2010.

[26] A. Nowogrodzki. How to support open-source software and stay sane. *Nature*, 571(7763):133–135, 2019.

[27] I. of Electrical and E. Engineers. Iso/iec/ieee international standard - systems and software engineering – life cycle processes –requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, pages 1–94, 2011.

[28] Opensource.org. The open source definition, July 2006. Page created on July 7, 2006, last modified on February 16, 2024, retrieved 24 June 2024.

[29] Opensource.org. International authority & recognition, Apr. 2015. Retrieved 24 June 2024, last modified on September 7, 2022.

[30] C. Payne. On the security of open source software. *Information systems journal*, 12(1):61–78, 2002.

[31] M. Pinzger and H. C. Gall. Dynamic analysis of communication and collaboration in oss projects. *Collaborative Software Engineering*, pages 265–284, 2010.

[32] W. Robinson and R. Vlas. Requirements evolution and project success: An analysis of sourceforge projects. *Requirements Engineering*, 2015.

[33] W. Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings-Software*, 149(1):24–39, 2002.

[34] W. Scacchi, C. Jensen, J. Noll, and M. Elliott. Multi-modal modeling of open source software requirements processes. In *Proceedings of the International Conference on Open Source Software*, pages 1–8. Citeseer, 2005.

[35] K. Ven, J. Verelst, and H. Mannaert. Should you adopt open source software? *IEEE software*, 25(3):54–59, 2008.

[36] Q. Xie and A. M. Memon. Model-based testing of community-driven open-source gui applications. In *2006 22nd IEEE International Conference on Software Maintenance*, pages 145–154. IEEE, 2006.

[37] T. Yamakami. A three-layer view model of oss: Toward understanding of diversity of oss. In *13th International Conference on Advanced Communication Technology (ICACT2011)*, pages 1190–1194. IEEE, 2011.