HOME
HOCHSCHULE
MERSEBURG FH
University of
Applied Sciences

FACHBEREICH
INFORMATIK UND
KOMMUNIKATIONS-
SYSTEME

Studiengang:
Technische Redaktion und Wissenskommunikation (M.A.)

API-Dokumentation aus der Perspektive von Software-Entwicklern: Was macht sie besonders hilfreich?

Masterarbeit von

Julia Dusold

Matrikelnummer: 20846

julia.dusold@gmail.com

Betreuer: Dr. Michael Meng

Zweitbetreuer: M.A. Andreas Schubert

Abgabe: 15. April 2016

Inhaltsverzeichnis

1 Einleitung

Software-Entwickler stehen in der heutigen Zeit immer wieder vor der Aufgabe, sich in neue APIs (kurz für Application Programming Interfaces) einzuarbeiten. Dies führt dazu, dass APIs und auch deren Dokumentationen, die die Arbeit mit den APIs unterstützen und erleichtern sollen, ein relevantes Thema für die Forschung geworden sind.

Diese Masterarbeit zum Thema „API-Dokumentation aus der Perspektive von Software-Entwicklern: Was macht sie besonders hilfreich?“ entstand im Rahmen einer wissenschaftlichen Studie der Hochschule Merseburg, deren Ziel es ist, Informationen zur Optimierung von Software-Entwicklerdokumentationen zu sammeln.

Durch verschiedene Betrachtungen, Fragebögen, Interviews usw. soll herausgefunden werden, welche Bedürfnisse und Ansprüche Entwickler an Dokumentationen beim Erstellen neuer Software haben. Diese Arbeit ist auf API-Dokumentationen fokussiert, da diese eine zentrale Rolle bei der Entwicklung neuer Software spielen. APIs sind Programmierschnittstellen, die Entwicklern dazu dienen, zusätzliche Software für ein bereits bestehendes Betriebssystem oder eine vorhandene Software zu erstellen.

Im Rahmen dieser Masterarbeit wird untersucht, mit welcher Art von API-Dokumentation die Entwickler am liebsten arbeiten und warum dies so ist.

Dazu wurden sechs Entwickler des Internetdienstanbieters 1&1 Internet SE gebeten, eine API-Dokumentation vorzustellen, mit der sie besonders gerne arbeiten. Durch gezielte Nachfragen sollte so herausgefunden werden, was in einer API-Dokumentation besonders wichtig und hilfreich ist. Dabei ging es vor allem um Fragen wie

- Warum wird gerne mit dieser Dokumentation gearbeitet?
- Was ist an der Beispieldokumentation besonders gut gelungen?
- Welche Elemente sind besonders hilfreich?
- Wie werden bestimmte Informationen zu einer bestimmten Aufgabe im Dokument gesucht?

Ebenso sollten die Entwickler jeweils ein Negativbeispiel für eine API-Dokumentation vorstellen. Dabei stand die Herausarbeitung von Mängeln im Vordergrund wie zum Beispiel:

- Warum wird die Dokumentation als schlecht empfunden?
- Welche Elemente der Dokumentation sind dafür verantwortlich?
- Was fehlt dieser Dokumentation?
- Was könnte verbessert werden?

Auf der Grundlage dieser Ergebnisse können so Empfehlungen gegeben werden, um gute, hilfreiche API-Dokumentationen zu erstellen.

Um einen Einstieg in das Thema zu geben, wird im folgenden Kapitel 2 der theoretische Hintergrund erläutert. Dann wird erklärt, was diese APIs überhaupt sind, um deutlich zu machen, warum diese ein relevantes Forschungsthema darstellen.

Es folgt ein Einblick in den aktuellen Stand der Forschung in Kapitel 3 und es wird die Studie der Hochschule Merseburg vorgestellt, zu der diese Arbeit einen weiteren Beitrag leistet.

In Kapitel 4 wird die durchgeführte Untersuchung, deren Auswertung und deren Ergebnisse beschrieben.

Diskutiert werden diese in Kapitel 5. Hier werden auch Empfehlungen für das Erstellen hilfreicher API-Dokumentation gegeben sowie auf weiteren Forschungsbedarf hingewiesen.

2 Theoretischer Hintergrund

Dieses Kapitel soll den notwendigen theoretischen Hintergrund vermitteln. So wird in Kapitel 2.1 erklärt was eine API ist und wozu sie verwendet wird. Im Anschluss wird in Kapitel 2.2 darauf eingegangen was grundsätzlich Zweck von API-Dokumentationen ist, die Thema dieser Arbeit sind. Warum diese API-Dokumentationen eine so wichtige Rolle spielen, erläutert abschließend Kapitel 2.3.

2.1 Was sind APIs? Wozu werden sie verwendet?

API ist die Abkürzung für „Application Programming Interface", also eine Anwendungsprogrammierschnittstelle. Darunter versteht man Schnittstellen für Programmierer, durch die „bestimmte interne Funktionsabläufe abstrahiert werden"[DATACOM Buchverlag (2015)].

Schnittstellen im Allgemeinen dienen zum Austausch von Daten oder zur Nutzung von Funktionalitäten zwischen zwei Programmen[1]. Sie ermöglichen es „große Systeme zu entwerfen, zu bauen und zu betreiben"[Siedersleben (2004), S. 41].

Eine API ist eine Schnittstelle, die auf die Anwendungsprogrammierung spezialisiert ist. Sie ermöglicht es, Entwicklern ein Programm zu schreiben, „das Dienste von einem Betriebssystem oder einer anderen Anwendung anfordert"[TechTarget (2015)].

Das heißt, Entwickler können eigene Programme an ein Softwaresystem anbinden, ohne dabei auf den eigentlichen Code des Systems zurückgreifen zu müssen. Die Programme können die Schnittstelle benutzen und bereitgestellte Bausteine ansprechen, damit das Betriebssystem bestimmte Aktionen durchführt[2].

Man kann also sagen, dass APIs es ermöglichen, Daten, Inhalte und Dienste zwischen Systemen, Programmen, Webseiten und anderen Anwendungen auszutauschen. Die eigenen Programme können durch APIs die Datenpools und Benutzerkreise anderer nutzen, die ihnen sonst verschlossen bleiben[3].

[1] Vergleiche Zörner (2015) , Seite 99
[2] Vergleiche Hoffmann (2016)
[3] Vergleiche Savage (2013)

Es gibt viele Arten von APIs. In komplexen Programmen dienen APIs dazu, diese in einzelne Module aufzusplitten und so überschaubare Programmteile zu erhalten und nicht einen einzigen riesigen Programmcode[4]. Diese APIs nennt man interne APIs. Im Web spielen externe APIs eine große Rolle. Mit deren Hilfe können die Funktionen von Webapplikationen für die eigene Webseite oder App[5] genutzt werden.

Gut veranschaulichen lassen sich diese externen APIs anhand der externen Twitter API. Durch diese ermöglicht es Twitter, dass das Twitter-Backend durch Außenstehende genutzt wird. Durch die zur Verfügung gestellte API ist es möglich, Desktop-Anwendungen, wie zum Beispiel Tweetdeck, oder Smartphone-Apps, wie beispielsweise Tweetbot, zu entwickeln. Diese können über die API zum einen auf die Inhalte der Twitter-Webseite zugreifen und diese anzeigen (Timelines, Profile etc.) und zum anderen die Funktionen nutzen (Tweets senden, anderen Nutzern folgen usw.)[6].

Der Vorteil von solchen Schnittstellen ist es, dass Details über die Implementierung verborgen bleiben. So stellen sie einerseits eine Vereinfachung für den Benutzer der API (zum Beispiel Software-Entwickler oder Webdesigner) dar, da dieser sich nicht mit der eigentlichen Implementierung befassen muss. Andererseits kann der Benutzer bereits mit seiner eigenen Implementierung mithilfe der Bausteine beginnen, bevor die Implementierung auf der anderen Seite steht, solange klar ist, was die Bausteine grundsätzlich tun.

Dem Anbieter der API bringt diese Verborgenheit der Implementierung ebenfalls Vorteile. Zum Beispiel kann die Implementierung verändert werden, ohne dass derjenige, der die API verwendet, davon direkt betroffen ist.

Auch dies lässt sich am Beispiel von Twitter veranschaulichen. Jemand verwendet die API von Twitter, um einen Button auf seiner Webseite einzubinden mit dessen Hilfe der Link zu dieser Seite auf Twitter gepostet werden kann. Dazu stellt Twitter eine Hyperlink-Klasse und einige Parameter zur Verfügung. Zum Beispiel kann ein Tweet-Text vordefiniert werden, indem der Benutzer „data-text=”Tweet-Text” seinem Hyperlink hinzufügt[7]. Wie allerdings dieser Text durch das Backend von Twitter in den letztendlichen Tweet gelangt, also die Implementierung dieses Parameters, das weiß der Benutzer nicht und das muss er auch nicht wissen.

All das bedeutet also, dass der Anbieter der API dem Benutzer Funktionen, Daten und so weiter zur Verfügung stellt, die dieser für sich verwenden kann. „Schnittstellen bilden somit einen Vertrag zwischen Anbietern und Verwendern mit Nutzen für beide Seiten”[Zörner (2015), S. 99].

[4] Vergleiche hierzu Hery-Moßmann (2015)
[5] kurz für Applikation, wird häufig im Zusammenhang mit Smartphones und Tablets verwendet.
[6] Die Twitter API ist verfügbar unter https://dev.twitter.com/ [abgerufen am 10.03.2016]
[7] Vergleiche https://dev.twitter.com/web/tweet-button [abgerufen am 10.03.2016]

2.2 Was beinhalten API-Dokumentationen und wofür werden sie verwendet?

Siedersleben (2004) beschreibt als wichtigsten Teil einer Softwaredokumentation die „Außensicht", welche sich an die Benutzer einer Software-Komponente richtet. Darum sollte ein Benutzerhandbuch dies beantworten:

> „Was muss der Benutzer tun, was mute ich ihm zu, welche Arbeit nehme ich ihm ab?"[Siedersleben (2004), S. 70]

Genau diese Fragen sollten auch durch die API-Dokumentation beantwortet werden, denn auch sie wendet sich an die Benutzer, bei denen es sich in der Regel um Programmierer oder Webdesigner handelt.

API-Dokumentation ist dazu da „damit jeder Programmierer weiß, wie ein Modul zu nutzen ist"[Oey (2007), S. 9]. Dies geschieht, indem die bereitgestellten Funktionen einer Schnittstelle aufgezählt sowie erläutert werden und ihre Anwendung erklärt wird.

Ein gutes Beispiel ist die Dokumentation des in Kapitel 2.1 beschriebenen Tweet-Buttons. Es wird zunächst kurz erläutert, was der Tweet-Button ist und wie er funktioniert (siehe Abbildung 2.1). Danach wird beschrieben, wie man einen solchen Button auf seiner eigenen Webseite einbaut und es wird gleichzeitig der Code mitgeliefert, mithilfe dessen dies geschieht. Zuletzt werden die Parameter beschrieben, die verwendet werden können, um den Tweet-Button zu personalisieren (siehe Abbildung 2.2).

Eine klassische API-Dokumentation beinhaltet überwiegend Klassen- und Methodenbeschreibungen, welche meist in Form von Code-Kommentaren geschrieben sind.

Diese Kommentare, die in der Regel in Form von HTML, CSS und Javascript Code vorliegen, können von Software-Dokumentationswerkzeugen in besser lesbaren Formaten, zum Beispiel HTML oder PDF, ausgegeben werden (z.B. Javadoc für die Programmiersprache Java oder Doxygen für die Programmiersprache C++). Die Dokumentationswerkzeuge durchsuchen den Code nach Kommentaren und Tags, die entsprechend den Vorgaben des Werkzeugs gekennzeichnet sind, und erzeugen daraus die Dokumentationsdatei.

Außerdem gibt es User Interfaces für API-Dokumentationen, die ebenfalls aus diesen Code-Kommentaren Dokumentation generieren (z.B. Swagger UI[8] für REST[9]-Schnittstellen). Ein User Interface ist im Allgemeinen dazu da, zwischen dem Benutzer und der Software zu vermitteln[10]. In diesem Fall vermitteln sie also zwischen API und deren Benutzer, indem sie ähnlich wie die Dokumentationstools den Code zum Beispiel

[8] Eine Beschreibung des Swagger UIs ist im Rahmen der Ergebnisse in Kapitel 4.4.8 zu finden.

[9] REST steht für „Representational State Transfer" und bezeichnet einen Software-Architekturstil für Netzwerke, wie zum Beispiel das World Wide Web [Vergleiche Fielding (2000), S. 76].

[10] Vergleiche Clements (2003), S. 12

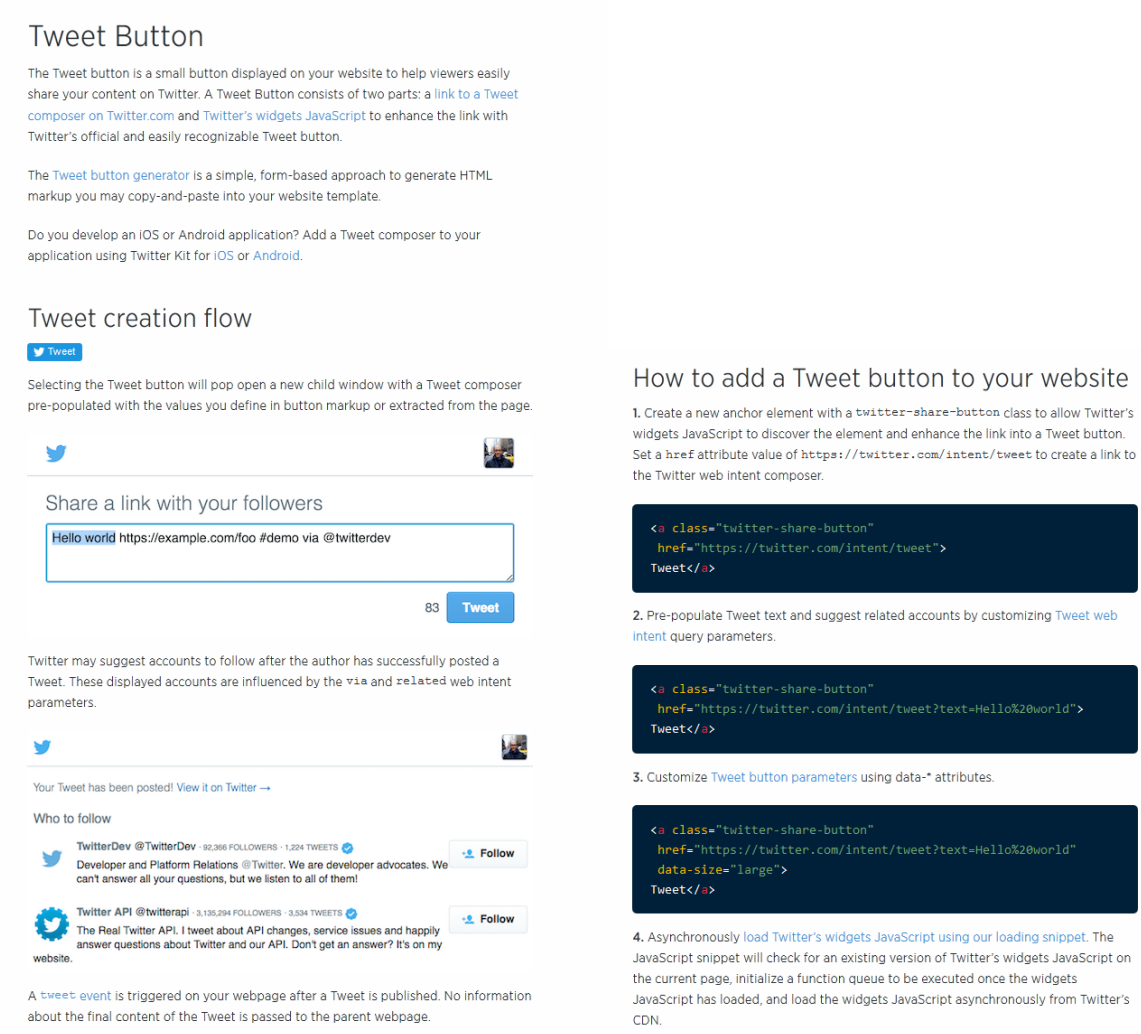
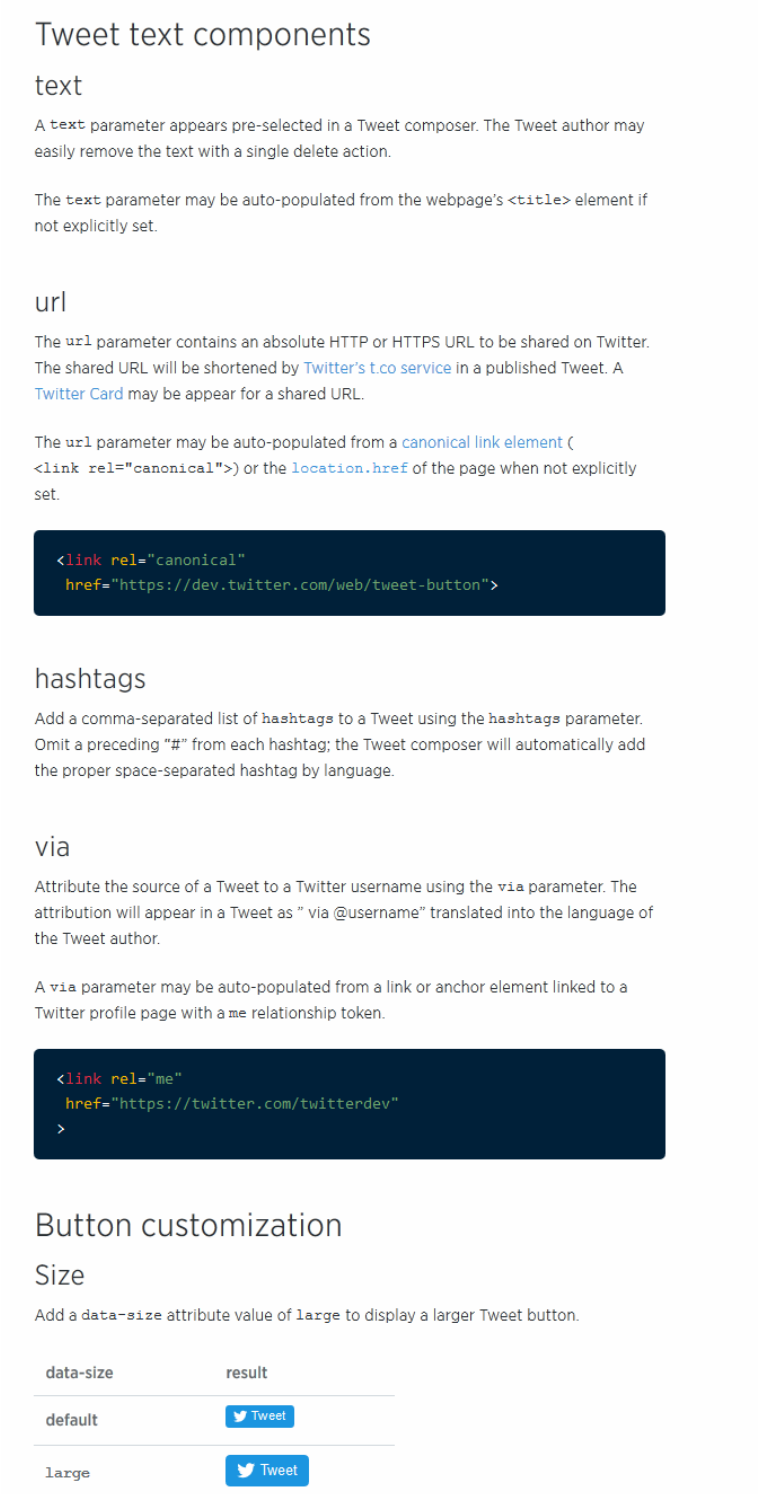


Abbildung 2.1: Dokumentation des Tweet-Buttons: Das rechte Bild zeigt die Erläuterung der Funktionsweise und das linke Bild beschreibt wie der Button einer Webseite hinzugefügt wird inklusive Beispielcode. (Quelle: Twitter Developers Documentation, https://dev.twitter.com/web/tweet-button, abgerufen am 02.04.2016)



Tweet text components

text

A `text` parameter appears pre-selected in a Tweet composer. The Tweet author may easily remove the text with a single delete action.

The `text` parameter may be auto-populated from the webpage's `<title>` element if not explicitly set.

url

The `url` parameter contains an absolute HTTP or HTTPS URL to be shared on Twitter. The shared URL will be shortened by Twitter's t.co service in a published Tweet. A Twitter Card may be appear for a shared URL.

The `url` parameter may be auto-populated from a canonical link element (`<link rel="canonical">`) or the `location.href` of the page when not explicitly set.

```
<link rel="canonical"
 href="https://dev.twitter.com/web/tweet-button">
```

hashtags

Add a comma-separated list of `hashtags` to a Tweet using the `hashtags` parameter. Omit a preceding "#" from each hashtag; the Tweet composer will automatically add the proper space-separated hashtag by language.

via

Attribute the source of a Tweet to a Twitter username using the `via` parameter. The attribution will appear in a Tweet as " via @username" translated into the language of the Tweet author.

A `via` parameter may be auto-populated from a link or anchor element linked to a Twitter profile page with a `me` relationship token.

```
<link rel="me"
 href="https://twitter.com/twitterdev"
>
```

Button customization

Size

Add a `data-size` attribute value of `large` to display a larger Tweet button.

data-size	result
default	Tweet
large	Tweet

Abbildung 2.2: Dokumentation der Datentypen und Parameter des Tweet-Buttons. (Quelle: Twitter Developers Documentation, https://dev.twitter.com/web/tweet-button, abgerufen am 02.04.2016)

nach Kommentaren durchsuchen und eine Dokumentation erstellen. Der Unterschied ist, dass hier keine statischen Dokumente erzeugt werden, sondern eine interaktive Schnittstelle zwischen Benutzer und Dokumentation.

Zurück zu den Inhalten einer API-Dokumentation. Neben den oben erwähnten Klassen und Methodenbeschreibungen sollten noch weitere Informationen in der Dokumentation enthalten sein, damit diese vollständig ist:

> „Zur Beschreibung einer Schnittstelle gehören Syntax (Formales, technische Details) und Semantik (fachliche Bedeutung)"[Zörner (2015), S. 100].

Zu diesen semantischen Informationen gehört auch das Verhalten der Schnittstelle, wie zum Beispiel zeitliche Reihenfolgen, Laufzeitverhalten oder Informationsfluss innerhalb der Schnittstelle[11].

Aus diesen unterschiedlichen Informationsarten resultiert die Aufteilung der API-Dokumentation in die zuvor bereits erwähnte klassische API-Dokumentation und eine API-Referenzdokumentation, die dann eben jene weiterführenden Informationen enthält. Wird in dieser Arbeit von API-Dokumentation gesprochen, so ist der Verbund dieser beiden Dokumentationen gemeint.

Laut Gruenbaum (2010) sollte API-Dokumentation aus vier Teilen bestehen:

1. Ein Überblick, der den Entwicklern die Schnittstelle und deren Architektur erklärt,
2. einen Soforteinstieg in Form von Schritt-für-Schritt-Anleitungen oder einfache Walkthroughs,
3. Code-Beispiele, die von den Software-Entwicklern zum Bauen ihres eigenen Codes verwendet werden können, und
4. Referenzmaterial, welches Informationen über alle Klassen, Parameter, Funktionen oder auch XML-Elemente liefert.

Diese wesentlichen Inhalte einer API-Dokumentation werden in einigen weiteren Quellen beschrieben[12] und werden in vielen Dokumentationen auch so umgesetzt (z.B. Android API). Welche Teile der API-Dokumentation besonders wichtig sind, hängt größtenteils vom Entwicklertyp und vom Anwendungskontext ab[13].

Einen genauen und allgemein gültigen Leitfaden für den Inhalt und den Aufbau von API-Dokumentationen gibt es allerdings (noch) nicht. Daraus ergibt sich die Notwendigkeit, sich auf Forschungsebene mit dem Thema auseinanderzusetzen, um herauszufinden, wie eine besonders hilfreiche API-Dokumentation aussehen soll. Doch warum sollte man sich diese Mühe machen?

[11] Vergleiche Clements (2003), S. 183
[12] Vergleiche zum Beispiel Clements (2003), S. 229 oder parson (2013)
[13] Vergleiche parson (2013)

2.3 Warum sind APIs und deren Dokumentationen wichtig?

Wie bereits in Kapitel 2.1 erwähnt, ermöglichen APIs den Software-Entwicklern eigene Komponenten für bereits bestehende Softwaresysteme zu programmieren, ohne dieses System im Ganzen zu kennen. Um die Wichtigkeit solcher APIs und damit auch ihrer Dokumentation zu erklären, ist es hilfreich, unsere heutige digitale Welt zu betrachten.

Wir sind vernetzt, benutzen das Internet auf vielen verschiedenen Geräten und haben Apps für so ziemlich alle Bereiche in unserem Leben. Doch wie können diese Apps mit den Geräten kommunizieren? Woher bekommen sie ihre Daten? Wie kann jemand App-Inhalte in sozialen Netzwerken teilen? All dies wird durch APIs ermöglicht.

Sie „dienen also zum Austausch und der Weiterverarbeitung von Daten und Inhalten zwischen verschiedenen Webseiten, Programmen und Content-Anbietern. Darüber hinaus ermöglichen sie so Dritten den Zugang zu vorher verschlossenen Datenpools und Benutzerkreisen"[Hoffmann (2016)].

Zum Beispiel stellt Google seine Android API öffentlich zur Verfügung, sodass es Entwicklern rund um die Welt möglich ist, eigene Apps für dieses mobile Betriebssystem zu schreiben. Ein anderes Beispiel sind die Facebook-Buttons[14], die mittlerweile auf vielen Webseiten zu finden sind. Auch diese werden über eine Schnittstelle an Facebook angebunden. Genauso erfolgt das Einbetten eines YouTube-Videos auf der eigenen Webseite über eine API.

Man stellt fest: APIs sind überall.

Welchen Nutzen es bringt, die eigenen Dienste anderen zur Verfügung zu stellen, kann gut am Beispiel von YouTube erklärt werden. Wie bereits erwähnt, kann der Videoplayer von YouTube über die angebotene API in Webseiten, Apps etc. eingebunden werden. Da dies durch die API einfacher ist, als einen eigenen Videoplayer zu bauen, laden viele ihre Videos auf YouTube, um diese auf ihrer eigenen Webseite einbinden zu können. Auch YouTube profitiert davon, da dadurch zum einen die Zahl der YouTube Nutzer steigt und zum anderen mehr Besucher auf die Plattform gelockt werden. Und mehr Besucher bedeuten mehr Werbeeinnahmen und so weiter.

APIs bieten demnach die Möglichkeit, eigene Dienste, Daten und Softwaresysteme anderen zur Verfügung zu stellen. So können für den Benutzer gut verständliche APIs helfen zu vernetzen, Dienste für sie attraktiv zu machen und Inhalte zu verbreiten[15].

Um eine API leicht verständlich zu machen, benötigt man eine gute Dokumentation. Diese soll es einfach machen, die Programmierschnittstelle zu verstehen. Sie erleichtert somit die Arbeit mit der API und macht das Softwaresystem oder den Webdienst,

[14] Mit Facebook-Buttons sind Schaltflächen auf Webseiten gemeint, die es ermöglichen, den Seiteninhalt auf Facebook zu teilen oder eine Seite mit „Gefällt Mir" zu markieren.
[15] Vergleiche Nijim u. Pagano (2014), S. 5f.

an den diese anbindet, gegebenenfalls beliebter. Dies wiederum führt dazu, dass mehr neue Anwender, Kunden usw. gewonnen werden können[16].

Durch gute Dokumentationen können auch die Support-Kosten einer API reduziert werden. Wenn die Benutzer die Antworten zu ihren Problemen in der Dokumentation finden können, kommen weniger Nachfragen[17].

Mit dieser großen Relevanz von APIs und deren Dokumentation in der digitalisierten Welt von heute geht die Notwendigkeit von Forschung einher, die sich mit dem Thema der Optimierung von API-Dokumentationen beschäftigt.

[16] Vergleiche Nijim u. Pagano (2014), S. 9ff.
[17] Vergleiche Gruenbaum (2010), S. 70 und Bloch (2006), S. 2

3 Aktueller Stand der Forschung bezüglich der Optimierung von API-Dokumentationen

Die vorliegende Arbeit deckt mit ihrer Studie einen kleinen Teil des Forschungsfeldes bezüglich der Optimierung von API-Dokumentationen ab. Die steigende Relevanz von APIs im digitalen Zeitalter führt dazu, dass deren Dokumentation auch immer mehr an Bedeutung gewinnt.

Dieses Kapitel gibt einen Überblick über den bisherigen Stand der Forschung.

In Kapitel 3.1 werden einige bedeutsame Studien zum Thema API-Dokumentation vorgestellt und deren Ergebnisse kurz erläutert. Kapitel 3.2 stellt die Studie der Hochschule Merseburg vor, die auch die vorliegende Arbeit beinhaltet. Es werden die verschiedenen Phasen der Studie beschrieben.

In Kapitel 21 wird der aktuelle Stand der Forschung diskutiert und in Kapitel 3.4 die für diese Arbeit grundlegenden Forschungsfragen dargestellt.

3.1 Überblick über bisherige Studien zu API-Dokumentationen

Beispielhaft werden an dieser Stelle einige Studien besprochen, die sich mit API-Dokumentationen beschäftigen und in der Forschungsfrage und/oder der Vorgehensweise ähnlich dieser Arbeit und der Studie der Hochschule Merseburg sind. Man erhält so einen guten Überblick über das Forschungsfeld als solches.

3.1.1 David G. Novick und Karen Ward: What users say they want in documentation

Eine Studie, die Benutzer befragte, um herauszufinden, was diese sich von einer Software-Dokumentation wünschen, ist die von Novick u. Ward (2006). Hier wurden Interviews durchgeführt, um Meinungen zusammen zu tragen, wie die Erwartungen der Testpersonen aussehen.

Bei den untersuchten Dokumentationen handelte es sich allerdings nicht um API-Dokumentationen, sondern um Software-Dokumentationen im Allgemeinen. Dennoch

zeigt die Studie einige allgemeine Ansatzpunkte, wenn es um die Dokumentation von Software geht. Diese sind[18]:

- Navigation - Für einen Großteil der Befragten ist ein gut organisiertes, leicht zu navigierendes und leicht zu durchsuchendes Dokument das Wichtigste.
- Problem-Orientierung - Da die meisten Personen die Dokumentation erst nutzen, sobald Probleme auftauchen, sollte diese auf die Problemlösung ausgerichtet sein.
- Angemessene Erklärungen - Die in der Dokumentation gegebenen Erklärungen sollten an den Wissensstand der Benutzer angepasst sein, um sowohl Langeweile als auch Überforderung vorzubeugen.
- Präsentation - Die Darstellung der Dokumentation sollte ihren Problemlösungscharakter als auch die Suche nach relevanten Informationen unterstützen.
- Vollständigkeit und Fehlerfreiheit - Es ist für den Benutzer von besonderer Bedeutung, alle benötigten Informationen zur Verfügung gestellt zu bekommen und sich auf deren Richtigkeit verlassen zu können.

Dies alles sind Punkte, die von Entwicklern angesprochen wurden und die gegebenenfalls auch auf API-Dokumentation zutreffen. Dies gilt es mit einer passenden Studie zu überprüfen.

3.1.2 Martin P. Robillard und Robert DeLine: A field study of API learning obstacles

Die Studie von Robillard u. DeLine (2011) beschäftigt sich mit dem Lernprozess von APIs. Die Forschungsfrage war dabei „Was macht APIs schwer zu erlernen?”. Im Hinblick darauf wurde eine Kombination aus Umfragen und Interviews mit Software-Entwicklern von Microsoft durchgeführt. Diese Vorgehensweise ist ähnlich der Studie, in deren Rahmen die vorliegende Arbeit angefertigt wurde.

Sie bestand aus drei Untersuchungen: Zunächst gab es eine explorative Studie, um eine große Breite an Lernhindernissen in APIs zu finden und Einsichten in die Ansichten der Entwickler zu gewinnen, um die weiteren Studien zu planen.

Daraufhin folgten qualitative Interviews, mit denen genaue Ansichten und Meinungen von Entwicklern hinsichtlich des Erlernens neuer APIs am Arbeitsplatz eingeholt wurden. Es folgte außerdem eine Follow-Up-Studie, die dazu diente, die bisherigen Ergebnisse zu bestätigen und weitere Daten zu sammeln.

In der Studie wurden sowohl Software-Entwickler als auch Software-Architekten und andere Mitarbeiter im Bereich der Software-Entwicklung befragt.

[18] Vergleiche Novick u. Ward (2006), S. 4ff.

Das relevanteste Ergebnis der Studie war, dass das am Häufigsten auftretende Problem beim Erlernen einer neuen API eine nicht ausreichende Dokumentation ist.

Einer der an der Studie teilnehmenden Software-Entwickler bestätigte die Wichtigkeit der API-Dokumentation folgendermaßen:

> „The biggest hurdle when learning APIs is the documentation. If the documentation for an API is good, it solves 99% of your problems."[Robillard u. DeLine (2011), S. 704]

Wenn also eine gute Dokumentation dazu führt, dass APIs problemloser erlernt werden können, so ist sie ein besonders wichtiger Bestandteil und es muss das Ziel sein, eine Dokumentation so gut wie möglich zu gestalten.

Die Studie ergab fünf besonders wichtige Faktoren, die beim Erstellen von guten API-Dokumentationen beachtet werden sollten. Diese sind die Dokumentation des Zwecks der API („intent documentation"), Code Beispiele („code examples"), passende Szenarios zu der API („matching APIs with scenarios"), die Transparenz der API („penetrability") sowie das Format und die Präsentation[19].

Durch die Befragungen von Robillard und De Line über diese Punkte ergaben sich diese Schlussfolgerungen:

- Die Dokumentation über die Benutzung und den Zweck der API sollte nur im Bedarfsfall zur Verfügung gestellt werden, nämlich da, wo die richtige Benutzung nicht selbst-erklärend ist oder wo es um die Betriebseigenschaften geht[20].
- Code-Beispiele, die umfassende Gebrauchsmuster darstellen sind hilfreicher als solche, die nur die Verwendung einzelner Klassen und Methoden zeigen. Dabei sollten die in einer API-Dokumentation aufgeführten Beispiele die optimale Vorgehensweise darstellen[21].
- Als besonders hilfreich wird es empfunden, wenn die API-Elemente mit den passenden Szenarien verbunden werden, da dies es vereinfacht, die Struktur der API zu verstehen[22].
- Im Hinblick auf die Transparenz der API ist es wichtig abzuwägen, welche Informationen über das Innere der API, sprich deren Implementierung, für die Benutzer relevant sind und welche nicht. Aus der Studie geht hervor, dass die relevanten Information die sind, die sich auf die Leistungsfähigkeit der API Elemente, das Behandeln von Fehlerfällen und die Erklärung abstrakter Operationen beziehen[23].

[19] Vergleiche Robillard u. DeLine (2011), S. 704f.
[20] Vergleiche Robillard u. DeLine (2011), S. 717
[21] Vergleiche Robillard u. DeLine (2011), S. 718f.
[22] Vergleiche Robillard u. DeLine (2011), S. 720
[23] Vergleiche Robillard u. DeLine (2011), S. 721f.

- Auch bezüglich des Formats und der Präsentation wurden einige Wünsche der Befragten zusammengetragen. Dabei stellte sich heraus, dass es unter den Entwicklern unterschiedliche Lerntypen gibt. Doch unabhängig von den Lerntypen wünschen sich viele eine klare, einheitliche und lineare Präsentation der Dokumentation, da eine zusammengewürfelte Darstellung aus Links zu Verwirrung führen kann[24].

Bei der Studie von Robillard u. DeLine (2011) ist allerdings zu beachten, dass sie nur mit Mitarbeitern einer einzigen Organisation (Microsoft) durchgeführt wurde. Es wird jedoch von den Autoren betont, dass die Studie trotzdem sehr repräsentativ ist, da Microsoft über 30.000 Entwickler beschäftigt, die unterschiedliches Hintergrundwissen und verschiedene Erfahrungsstände haben.

3.1.3 Sae Young Jeong, Yingyu Xie u.a.: Improving Documentation for eSOA APIs through User Studies

Eine weitere Studie, die es sich zum Ziel gesetzt hat, API-Dokumentation zu verbessern, ist die von Jeong u. a. (2009). Sie beschäftigt sich mit APIs für „enterprise Service-Orientated Architecture" (eSOA)[25] und deren Dokumentation. Die Untersuchung fokussierte sich auf die Usability der Online-Dokumentation, die von SAP für ihr SOA Produkt zur Verfügung gestellt wird.

In dieser Studie wurde eine andere Herangehensweise gewählt: Die Teilnehmer bekamen eine Aufgabenstellung bezüglich der Dokumentation, die sie zu erledigen hatten. Während sie die Aufgabe bearbeiteten, wurde die Methode des Lauten Denkens[26] angewendet, um die notwendigen Einsichten zu bekommen.

Die Teilnehmer waren Software-Entwickler, die alle aus der Zielgruppe für eSOAs stammten. Es wurde zwischen zwei Gruppen unterschieden: Die eine Hälfte hatte einen betriebswirtschaftlichen Hintergrund, die andere Hälfte nicht.

Besonders auffällig war, dass die Testpersonen mit betriebswirtschaftlichem Hintergrund die Aufgabe wesentlich besser bearbeiten konnten als diejenigen ohne. Auch unterschieden sich die beiden Gruppen durch die Art und Weise, wie sie durch die Dokumentation navigierten.

Eine Hauptaussage der Studie ist, dass nicht die Programmierkenntnisse, sondern das Hintergrundwissen eine große Rolle spielen.

[24] Vergleiche Robillard u. DeLine (2011), S. 723

[25] SOA ist eine Software-Architektur, die sich besonders auf die Implementierung von Dienstleistungen konzentriert. Bei eSOA handelt es sich um eine Version, die sich auf die Geschäftsprozesse innerhalb eines Unternehmens konzentriert. (Quelle: The Open Group: The SOA Source Book, https://www.opengroup.org/soa/source-book/soa/soa.htm, abgerufen am 17.12.2015)

[26] Die Methode des Lauten Denkens gibt Einblicke in die Gedanken und Absichten der beobachteten Person und so kann der Verarbeitungsprozess untersucht werden. Zum Anwenden der Methode wird die Testperson dazu aufgefordert alles laut auszusprechen, was ihr beim Bearbeiten der gestellten Aufgabe in den Sinn kommt. (Vergleiche Konrad (2010), S. 476f.)

Daraus resultiert die Empfehlung, dass in Dokumentationen verschiedene Möglichkeiten der Information und Navigation angeboten werden sollten, um diese an verschiedene Nutzer mit unterschiedlichen Hintergründen anzupassen[27].

Neben dieser Empfehlung wurden im Rahmen der Studie auch weitere Richtlinien ausgearbeitet, die beim Verfassen von API-Dokumentation befolgt werden sollten[28]:

- Die Dokumentation sollte ein einheitliches Format haben, damit die Entwickler sofort erkennen können, welche Teile zur eigentlichen Dokumentation gehören und wann sie auf gegebenenfalls externe Informationsquellen verwiesen werden.
- Am Anfang sollte eine gut gestaltete Übersicht der gesamten Dokumentation stehen und eine Erklärung der möglichen Herangehensweisen an die API. Auch sollte sichergestellt werden, dass es einen roten Faden gibt, der die Benutzer durch die Dokumentation führt.
- Relevante Terminologie sollte beschrieben werden. Auch die Beschreibung von Parametern und Services sollten hervorgehoben werden. Im Bezug auf Services ist es notwendig, diese eindeutig zu benennen.
- How-To-Anleitungen, Code-Beispiele, eine online Test-Funktion und eine leistungsfähige Suche sind wichtige Bestandteile in einer API-Dokumentation.

Vorgenannte Richtlinien dienten der in dieser Arbeit durchgeführten Studie als Grundlage für die Erstellung des Interviewleitfadens.

Bei der Studie von Jeong u. a. (2009) gilt es zu beachten, dass sie eine kleine Anzahl an Teilnehmern (acht) hatte und daher keine statistisch relevanten Ergebnisse liefern kann. Auch entspricht die Gruppe der Teilnehmer nicht zwangsläufig der gesamten Zielgruppe der API-Dokumentationen und die Aufgabe, die gestellt wurde, war einfacher als reale Aufgabenstellungen üblicherweise sind. Doch wie bei vielen Usability Studien ist diese Art der Beobachtungsstudie trotzdem gut geeignet, um nützliche Informationen zu erhalten.

3.1.4 Barthélémy Dagenais und Martin P. Robillard: Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors

Dagenais u. Robillard (2010) beschäftigten sich mit der Entwickler-Dokumentation von Open-Source Software. Im Vergleich zu den bisherigen Studien wurden hier sowohl die Benutzer als auch einige Autoren von Open-Source Dokumentationen befragt. Zusätzlich wurde der Änderungsverlauf von Dokumentationen analysiert, um so die Entscheidungen, die zur Änderung führten, verfolgen zu können.

[27] Vergleiche Jeong u. a. (2009), S. 87
[28] Vergleiche Jeong u. a. (2009), S. 101ff.

Ein wichtige Beobachtung der Studie ist, dass die Überarbeitung der Dokumentation häufig auch die Qualität des Codes verbessert. Eine andere ist, dass die ständige Interaktion unter den am Projekt Beteiligten einen positiven Effekt auf die Dokumentation hat.

Durch diese Studie wird deutlich, dass Software-Dokumentation ein dynamisches Produkt ist und konstant überarbeitet werden muss. Spätestens mit jedem Update der Software[29] oder mit Fragen, die mehrfach von Benutzern der API gestellt werden [30], sollte die Dokumentation geändert werden. So kann das Erlernen einer Software optimal unterstützt werden und die Software-Dokumentation durch die Nutzergemeinschaft verbessert werden.

3.1.5 Robert Watson u.a.: API Documentation and Software Community Values: A Survey of Open Source API Documentation

Mit API-Dokumentationen im Open Source Bereich beschäftigt sich die Studie von Watson u. a. (2013). Sie hebt sich dadurch ab, dass hier keine Interviews oder Beobachtungsstudien durchgeführt wurden, sondern API-Dokumentationen aus Open Source Communities untersucht wurden. So sollten die Thesen überprüft werden, die in bisherigen Studien aufgestellt wurden, wie in den vorherigen Kapiteln beschrieben. Eine Hauptthese dabei war:

> „The documentation of open-source software will contain the elements that software developers want."[Watson u. a. (2013), S. 166]

33 API-Dokumentationen aus beliebten Open Source Projekten wurden hinsichtlich ihrer Elemente, der Gestaltung und dem Schreibstil analysiert. Dazu wurden Tabellen mit Eigenschaften und Elementen von API-Dokumentationen erstellt, die dann beim Durcharbeiten der einzelnen Dokumentationen ausgefüllt wurden.

Die Analyse ergab, dass die bearbeiteten API-Dokumentationen alle oder fast alle derjenigen Elemente enthielten, die in früheren Studien als wünschenswert beschrieben wurden. So waren zum Beispiel eine Übersicht über die gesamte Dokumentation und Code-Beispiele in über 80% der untersuchten API-Dokumentationen vorhanden.

Die Ergebnisse bestärken die in der bisherigen Literatur beschriebenen Anforderungen und zeigen, dass das Design und die Qualität des Schreibstils weitere wichtige Punkte in den Anforderungen sind und weiterer Forschung bedürfen, um genauere Ergebnisse und Richtlinien zu diesen Punkten zu erhalten.

[29] Vergleiche Dagenais u. Robillard (2010), S. 6
[30] Vergleiche Dagenais u. Robillard (2010), S. 7

3.1.6 Walid Maalej und Martin P. Robillard: Patterns of Knowledge in API Reference Documentation

Für diese Arbeit ebenso von Bedeutung ist die Studie von Maalej u. Robillard (2013), die sich mit API-Referenzdokumentation beschäftigt. Einige der im Rahmen dieser Arbeit befragten Software-Entwickler erwähnten häufig die Wichtigkeit der Unterscheidung zwischen API Dokumentation und Referenzdokumentation. Referenzdokumentation ist dazu da, zusätzliches Wissen zur API zur Verfügung zu stellen, welches nicht direkt aus dem Code resultiert. Ziel ist es, Autoren von API Dokumentation zu helfen, ihre Dokumente zu evaluieren und besser zu organisieren.

Um herauszufinden, welche Art von Wissen in den Dokumentationen vermittelt wird und wie dieses über die Dokumentation verteilt ist, wurden über 5000 Dokumentationseinheiten untersucht. Dabei standen folgende Fragen im Vordergrund[31]:

- „Welche verschiedenen Arten von Wissen gibt es in API-Referenzdokumentationen?"
- „Können die verschiedenen Wissensarten durch die Formatierung erkannt werden?"
- „Wie sind diese unterschiedlichen Arten von Wissen auf die verschiedenen API Elemente verteilt?"

Die durch die Untersuchung gefundenen Wissensarten sind (sortiert nach Häufigkeit des Auftretens)[32]:

- Funktionalität - Beschreibung des Verhaltens der API.
- Nullinformation - Text oder Textteile, die keine Information enthalten, sondern nur Füllstoff sind.
- Struktur - Beschreibung der inneren Struktur der API (Klassen, Methoden, etc.) und deren Zusammenhang.
- Richtlinien - Erklärungen dessen, was der Benutzer mit bestimmten API Elementen machen darf und was nicht.
- Code-Beispiele - Bereitstellung von Beispielcode, der zum Erfüllen bestimmter Aufgaben genutzt werden kann.
- Muster - Erläuterungen dazu, wie bestimmte Ergebnisse mithilfe der API erreicht werden können, zum Beispiel How-To-Anleitungen.
- Steuerfluss - Beschreibung wie die Steuerung innerhalb der API abläuft oder eine Aufzählung der Methoden, die durch einen Aufruf ausgeführt werden.
- Zweck - Erklärung des Zwecks der zur Verfügung gestellten Elemente oder der Gründe für Design-Entscheidungen.

[31] Vergleiche Maalej u. Robillard (2013), S. 2
[32] Vergleiche Maalej u. Robillard (2013), S. 5 und S. 12

- Referenzen - Verweise auf weiterführende Informationen.
- Konzepte - Beschreibung der verwendeten Begriffe oder Beschreibungen zum Design und den von der API verwendeten Konzepten.
- Qualität - Beschreibung nicht funktionaler Anforderungen, zum Beispiel Konsequenzen für die Performance der API, oder Erklärungen zur internen Implementierung.
- Umgebung - Beschreibung der Umgebung, in der die API verwendet wird, zum Beispiel die Versionsgeschichte oder Lizenzinformationen.

Die Ergebnisse der Studie liefern außerdem ein Vokabular, das Autoren von API-Referenzdokumentation dabei helfen kann, sich mit ihren Inhalten auseinanderzusetzen und die verschiedenen Wissensarten geschickt zu kombinieren. Des Weiteren kann die Dokumentation dadurch verbessert werden, indem die durch die Studie identifizierten Nullinformationen entfernt werden.

3.1.7 Weitere Studien

Weitere erwähnenswerte Studien sind „The Role of Conceptual Knowledge in API Usability" von Ko u. Riche (2011) und „An Exploratory Study of API Usage Examples on the Web" von Wang u. a. (2012).

Ko u. Riche (2011) beschäftigten sich mit der Relevanz von konzeptionellem Wissen und kamen zu dem Schluss, dass Entwickler ohne solches Wissen große Probleme beim Lösen von Aufgaben mit der API haben. Daher sollte die zugehörige Dokumentation nicht nur anwendungsbezogene Informationen enthalten, sondern auch konzeptionelle Informationen und Hintergrundwissen.

Wang u. a. (2012) hingegen beschränkten ihre Studie auf die Relevanz und Qualität von Code-Beispielen hinsichtlich der Benutzung einer API. Dabei fanden sie heraus, dass das Internet zwar eine gute Quelle für Code-Beispiele ist, aber dass damit zusammenhängend die Notwendigkeit besteht, das Auffinden der für den einzelnen Entwickler relevanten Beispiele zu erleichtern.

3.2 Die Studie der Hochschule Merseburg

Die im vorherigen Kapitel vorgestellten Studien zeigen, dass fehlende beziehungsweise unzureichende Informationen in API-Dokumentationen meist die größten Hindernisse sind, wenn Software-Entwickler sich neu einarbeiten. Daher ist es wie bereits in Kapitel 2.3 beschrieben wichtig, dass die Anbieter von APIs Dokumentationen bereitstellen, die das Einarbeiten und das Nutzen von APIs optimal unterstützen.

Da in diesem Bereich bisher nur wenig geforscht wurde und dies hauptsächlich in den USA, hat es sich die Studie der Hochschule Merseburg, geleitet von Prof. Dr. Michael Meng, zur Aufgabe gemacht, diese Forschungslücke zu schließen.

Dabei stehen die folgenden zentralen Fragen, die zum Erreichen der Ziele führen sollen, im Vordergrund:

- Wie sollte die Dokumentation für die Zielgruppe der Software-Entwickler aussehen, damit der Lernprozess optimal unterstützt wird?
- Welche Inhalte sollten in einer effektiven Entwicklerdokumentation enthalten sein, und wie sollten diese Informationen dargestellt und strukturiert werden?
- Wie können die Anbieter der Dokumentation das Wissen optimal für die Nutzung durch Benutzer von APIs, die auch außerhalb des Unternehmens bereitgestellt werden, aufbereiten?
- Wie kann der Informationsaustausch zwischen Experten durch die Entwicklerdokumentation verbessert werden?

Die Ziele des Projekts der Hochschule Merseburg sind

- durch quantitative und qualitative Untersuchungen zu ermitteln, welche Lernstrategien und Informationsbedürfnisse die Software-Entwickler haben,
- Strategien und Umsetzungsempfehlungen im Hinblick auf Inhalt, Struktur und Gestaltung zu erarbeiten,
- eine erste Umsetzung dieser Empfehlungen durchzuführen sowie
- die Wirksamkeit der Verbesserungen zu testen.

Daraus resultiert, dass die Studie in mehrere Phasen gegliedert werden muss, in denen auf die einzelnen Ziele hingearbeitet wird.

Nach einer Analyse der Zielgruppe und einer Ermittlung des Kommunikationsbedarfs in Phase 1, geht es in Phase 2 an die eigentlichen umfassenden Untersuchungen.

Diese beinhalten diverse Verfahren, um die Erwartungen und Lernstrategien der Entwickler zu analysieren. Diese Verfahren umfassen Querschnittserhebungen in Form von Fragebogenuntersuchungen für statistisch relevante Ergebnisse, qualitative Untersuchungen wie Interviews und Beobachtungsstudien, um einen guten Überblick

über die Ansichten der Zielgruppe zu erhalten und eine Longitudinalstudie, welche die zeitliche Entwicklung des Lernprozesses beobachten soll.

Nach der Auswertung dieser Untersuchungen folgt Phase 3, die sich mit den Gestaltungsempfehlungen und deren erster Umsetzung beschäftigt. In Phase 4 werden dann die vorgenommenen Verbesserungen geprüft, um deren Wirksamkeit zu validieren.

Diese Arbeit wird zu dem Zeitpunkt erstellt, an dem sich die Studie der Hochschule Merseburg in der Phase der qualitativen Datenerhebung durch Interviews befindet, deren Ziel es ist, das Herangehen der Entwickler an APIs und deren Dokumentation zu untersuchen. Hierzu wird das Lernverhalten und die Erwartungen und Vorlieben der Entwickler näher betrachtet.

Neben den in dieser Arbeit geführten Interviews mit Entwicklern der 1&1 Internet SE, wurde im Vorfeld bereits eine Befragungen mit Entwicklern aus verschiedenen anderen Unternehmen durchgeführt.

In der Untersuchung von Steinhardt u. a. (2015) wurden 17 Software-Entwickler aus verschiedenen Firmen, mit unterschiedlich langer Programmiererfahrung befragt. Außerdem arbeiteten die Entwickler mit differenten Plattformen und Sprachen, das heißt die Hintergründe der einzelnen Teilnehmer waren sehr heterogen.

Als Gesprächsgrundlage diente, wie auch bei dieser Arbeit, ein Interviewleitfaden, der darauf abzielte, die relevanten Themen gezielt anzusprechen. Enthalten waren 45 Fragen zu 11 verschiedenen Themenbereichen, die die Erfahrungen und Erwartungen der Befragten an die Software-Dokumentation abdeckten. Hierbei ging es zum Beispiel um das Vorgehen beim Einarbeiten in eine neue API oder um Ansprüche an die Informationsdarstellung.

Die Auswertung erfolgte mittels einer qualitativen Inhaltsanalyse. Diese Methode wurde auch für diese Arbeit genutzt. Sie wird in Kapitel 4 im Bezug auf die durchgeführte Untersuchung genauer beschrieben.

Die Ergebnisse der Untersuchung von Steinhardt u. a. (2015) beinhalten

- die ersten Fragen an eine API,
- die Probleme und Strategien beim Einarbeiten in eine neue API,
- übergeordnete und inhaltliche Anforderungen sowie Funktionalitäten, die den Erwartungen an gute API-Dokumentation entspringen, und
- Informationen zur Beliebtheit von Foren.

Durch die Vielfältigkeit der abgefragten Themengebiete wurde eine breit gefächerte Menge an Informationen gewonnen, aus denen bereits erste Anforderungen an API-Dokumentationen abgeleitet werden können. Auch wurden zusätzliche Themen erkannt, die weitere Ansatzpunkte für die Studie bieten.

Nun galt es, die Ergebnisse dieser Studie durch weitere Befragungen zu ergänzen und einige Themengebiete weiter zu vertiefen. Eine dieser Ergänzungen ist diese Arbeit.

3.3 Diskussion des Forschungsstands

Viele der bisherigen Studien untersuchten die Usability von API Dokumentationen beim Erlernen einer bestimmten API anhand der Befragung oder Beobachtung von Entwicklern. So wurden oft Listen mit häufig gestellten Fragen erstellt oder Richtlinien für eine höhere Qualität bei der Dokumentation für die betreffende API.

Diese Studien konzentrierten sich meist auf eine einzige API und deren Dokumentation oder auf die in einem bestimmten Arbeitsumfeld verwendeten APIs; die Studie von Jeong u. a. (2009) konzentrierte sich zum Beispiel wie bereits erwähnt auf eSOA APIs im Bezug auf SAP Anwendungen.

Aus diesen Studien gibt es bereits grobe Übersichten über von Entwicklern erwünschte Inhalte in den Dokumentationen und Fragen, die diese beantwortet haben möchten. Doch durch die Beschränkung der Befragungen auf spezielle APIs, sind die Studien nur eingeschränkt repräsentativ und benötigen weiterer Nachforschungen.

Ein anderer Ansatzpunkt vieler Studien zum Thema API Dokumentation ist die Untersuchung von Open Source Software Dokumentationen, wie die von Watson u. a. (2013) und Dagenais u. Robillard (2010). Dabei wurden hauptsächlich die Dokumentationen selbst untersucht und zusammengetragen, welche Informationen häufig vorhanden sind und welche nicht. So wurde bereits ein guter Überblick geschaffen, welche Informationen in API-Dokumentationen enthalten sind. Es gilt allerdings zu klären, ob dies auch den Wünschen der Entwickler entspricht.

3.4 Resultierende Forschungsfragen für diese Arbeit

Die in Kapitel 3.1 und 3.2 dargestellten Studien zeigen wie auch Kapitel 3.3 diskutiert die Notwendigkeit weiterer Nachforschungen zum Thema der API-Dokumentationen.

Es geht dabei vor allem darum, weitere Einsichten in den Umgang der Software-Entwickler mit API-Dokumentationen zu erhalten.

Als eine Ergänzung der Untersuchung von Steinhardt u. a. (2015) geht diese Arbeit darauf ein, was API-Dokumentation für Software-Entwickler besonders hilfreich macht. Dabei stellt sich die Frage nach dem Informationsbedarf der Entwickler, aber auch danach, welche Aspekte eine Dokumentation für den Benutzer der API gut machen.

Des Weiteren soll die vorliegende Arbeit dazu dienen, die bisher gewonnen Ergebnisse der Studie der Hochschule Merseburg zu untermauern und eine weitere Grundlage für die als nächstes anstehenden Beobachtungsstudien zu bilden.

Dabei liegt der Fokus in dieser Studie nicht auf der Untersuchung einer bestimmten API und deren Dokumentation, wie es in vielen der bereits erwähnten Untersuchungen der Fall war, sondern um eine bunte Mischung von API-Dokumentationen und um allgemeine Ansichten von Software-Entwicklern. Das Augenmerk bei der Befragung soll darauf liegen, wie Software-Entwickler API-Dokumentation im Allgemeinen sehen und wie ihre ganz persönlichen Vorlieben aussehen.

Durch das Zusammentragen verschiedenster Lieblingsdokumentationen und Dokumentationen, die als besonders schlecht empfunden werden, können die verschiedensten Eigenschaften, Inhalte etc. zusammengetragen werden, die für die Software-Entwickler relevant sind. So können die bisher aus vorangegangenen Studien bereits existierenden Empfehlungen weiter ergänzt und bestätigt werden.

Außerdem ergibt sich durch die Untersuchung, die Möglichkeit in die einzelnen Themenbereiche, die von Entwicklern bezüglich API-Dokumentationen angesprochen werden, tiefer einzutauchen und umfassendere Informationen zu Unterthemen wie zum Beispiel dem erwünschten Umfang, der Struktur und Navigation oder dem Einsatz von Codebeispielen zu sammeln.

4 Eigene Untersuchung: Entwicklerinterviews zur Analyse beliebter API-Dokumentationen

Der in Kapitel 3 erläuterte Stand der Forschung bietet viele Ansätze für weitere Untersuchungen. In dieser Arbeit geht es wie bereits in Kapitel 1 beschrieben darum, herauszufinden, was eine API-Dokumentation für Software Entwickler besonders hilfreich macht und was nicht.

Dieses Kapitel beschreibt die Durchführung einer Studie anhand von Gesprächen mit Software-Entwicklern der 1&1 Internet SE. In Kapitel 4.1 wird die Untersuchungsmethode (Experteninterview) beschrieben. Anschließend erläutert Kapitel 4.2 die Durchführung der Interviews.

In Kapitel 4.3 wird das zur Analyse der Interviews verwendete Auswertungsverfahren (qualitative Inhaltsanalyse nach Mühlfeld) beschrieben und abschließend werden in Kapitel 4.4 die Ergebnisse dieser Untersuchung vorgestellt.

4.1 Die Untersuchungsmethode: Das Experteninterview

In dieser Arbeit wurde das Experteninterview als Untersuchungsmethode gewählt. An dieser Stelle werden zunächst die Grundlagen der Befragung als Forschungsinstrument und die Methode des Interviews (siehe Kapitel 4.1.1) erklärt. Im Anschluss wird in Kapitel 4.1.2 die Befragung, die im Rahmen dieser Arbeit in Form von Experteninterviews durchgeführt wurde, genauer erläutert.

4.1.1 Die Befragung als Forschungsinstrument

Menschen sammeln Erfahrungen, entwickeln ihre eigenen Überzeugungen und Einstellungen und nutzen diese, um sich in den verschiedensten Situationen zurecht zu finden. Diese sind dabei immer subjektiv, sie entwickeln sich aus Alltagserfahrungen in Abhängigkeit von dem sozialen Hintergrund und der eigenen Persönlichkeit. So existieren viele verschiedene Vorstellungen und Meinungen nebeneinander.

Um aus vielen einzelnen Meinungen eine wissenschaftlich gesicherte Erkenntnis zu erhalten, ist es notwendig, diese Überzeugungen und Einstellungen wertfrei und mit einer neutralen Distanz zu beurteilen[33].

Zur Erforschung von Einstellungen und Meinungen ist die Methode der Befragung ein gutes Instrument[34]. Dabei gibt es drei unterschiedliche Befragungstypen, die sich durch die Kommunikationsart unterscheiden[35]:

- das persönliche Interview,
- das telefonische Interview und
- die schriftliche Befragung.

Schriftliche Befragungen finden in Form von Fragebögen statt und sind stark standardisiert. Alle Befragten erhalten die gleichen Fragen und gegebenenfalls sogar dieselbe Auswahl an vorgegebenen Antwortmöglichkeiten. Dies sichert eine höchstmögliche Objektivität bei der Auswertung.

Das Problem bei solchen standardisierten Befragungen ist allerdings, dass „man bei geschlossenen Fragen keine Informationen jenseits des Spektrums der vorgelegten Antwortkategorien"[Diekmann (2007), S. 438] erhält und sie somit nur sinnvoll sind, wenn man bereits ein umfangreiches Vorwissen über das Forschungsobjekt besitzt.

Weniger strukturierte Interviews, wie zum Beispiel narrative Interviews oder Leitfadeninterviews, ermöglichen es ein solches Vorwissen zu schaffen, indem ein umfassenderes Meinungsbild eingeholt wird.

Strukturierte Befragungen wie Fragebögen und stark standardisierte Interviews werden als quantitative Befragungsmethoden bezeichnet, die offeneren und weniger strukturierten Interviewtechniken als qualitative Befragungsmethoden.

Laut Diekmann (2007) sind die qualitativen Methoden hauptsächlich für explorative Untersuchungen geeignet, um Typologien und Kategoriesysteme zu entwickeln und Forschungshypothesen zu generieren[36].

In dieser Arbeit sollen allgemeine Ansichten und Einstellungen der Software-Entwickler gesammelt werden, um Thesen zu erstellen, die zur Verbesserung von API-Dokumentationen dienen sollen. Dafür bietet sich die qualitative Forschung an, denn „der relativ offene Zugang qualitativer Forschung verhilft zu einer möglichst authentischen Erfassung der Lebenswelt der Betroffenen sowie deren Sichtweisen und liefert Informationen, die bei einer quantitativen Vorgehensweise auf Grund ihrer Standardisierung verloren gehen"[Mayer (2009), S. 25].

Daraus ergibt sich, dass es sinnvoll ist, die Befragung der Software-Entwickler in Form von Interviews durchzuführen. Doch was ist ein Interview?

[33] Vergleiche Mayer (2009), S. 9
[34] Vergleiche Diekmann (2007), S. 434
[35] Vergleiche Diekmann (2007), S. 437
[36] Vergleiche Diekmann (2007), S. 531

Laut Scheuch (1973) ist ein Interview wie folgt definiert:

> „Unter Interview als Forschungsinstrument sei hier verstanden ein planmäßiges Vorgehen mit wissenschaftlicher Zielsetzung, bei dem die Versuchsperson durch eine Reihe gezielter Fragen oder mitgeteilter Stimuli zu verbalen Informationen veranlasst werden soll."[Scheuch (1973), S. 70 f.]

Diekmann unterscheidet zwischen drei verschiedenen, häufig auftretenden Interviewformen[37]:

- Das fokussierte Interview - Diese Form des Interviews setzt voraus, dass die Befragten alle eine konkrete Situation erlebt haben, zum Beispiel ein bestimmtes Buch gelesen haben oder eine bestimmtes Gerät für eine konkrete Aufgabe genutzt haben. Das Ziel des Interviews ist es, diese Situation zu analysieren[38].

- Das narrative Interview - Diese Interviewtechnik nutzt die Erzählform und ist noch weniger strukturiert als das fokussierte Interview. Der Befragte wird nicht im eigentlichen Sinne befragt, sondern wird zur Erzählung ermuntert. So können „erfahrungsnahe, subjektive Aussagen über Ereignisse und biographische Abläufe"[Diekmann (2007), S. 540] gewonnen werden. Durch dieses Erzählschema „bringt der Informant Ereignisbestände zur Darstellung, über die er im konventionellen offenen Interview niemals Aussagen treffen würde"[Spöhring (1989), S. 169]. Daher wird diese Form des Interviews häufig angewendet, wenn es um besondere Erfahrungen im persönlichen Leben geht, wie zum Beispiel Arbeitslosigkeit, Drogenabhängigkeit etc.

- Das problemzentrierte Interview - Diese Interviewform ist wiederum etwas mehr strukturiert als das narrative Verfahren. Der Interviewer leitet das Gespräch durch problemorientierte Nachfragen auch in der Erzählphase. „Parallel zur Produktion von breitem und differenziertem Datenmaterial arbeitet der Interviewer schon an der Interpretation der subjektiven Sichtweise der befragten"[Witzel (2000), S. 2] Personen und konzentriert die Kommunikation dabei auf das Forschungsproblem.

Sowohl das fokussierte als auch das problemzentrierte Interview nutzen zur Durchführung einen Leitfaden, um das Interview zu steuern. Durch die Konzentration auf ein bestimmtes Problem und die Steuerung des Gesprächs mit gleichzeitiger Interpretation bietet sich das problemzentrierte Interview für die in dieser Arbeit durchgeführte Befragung an.

[37] Vergleiche Diekmann (2007), S. 536 ff.
[38] Vergleiche Merton u. Kendall (1979), S. 202 f.

4.1.2 Die Befragung im Rahmen dieser Arbeit

Um herauszufinden, was den Software-Entwicklern an API-Dokumentationen besonders gut gefällt und was sie eher am Arbeiten hindert, wurden im Rahmen dieser Arbeit Experteninterviews durchgeführt. Diese sind problemzentrierte Interviews, deren Ziel es ist, Aussagen über einen gewissen Gegenstand zu erhalten über den die Befragten sehr gut informiert sind.

Für diese Art der Datenerhebung ist die Nutzung eines Leitfadens der beste Weg[39]. Außerdem sichert der Leitfaden die Vergleichbarkeit der Interviews, da diese durch ihn einen einheitlichen Orientierungsrahmen erhalten[40].

Aufgabe des Interviewers in solchen Befragungen ist es, die Fragen des Leitfadens zu nutzen, um das Interview hinsichtlich der Problematik zu lenken.

> „Welche Frage des Leitfadens in welcher Formulierung der Interviewer wann stellt, soll der Situation angepasst sein."[Diekmann (2007), S. 542]

So sollte die Erzähllogik des Interviewten nicht durch das Vorbringen der Fragen beeinträchtigt werden.

Bei der Formulierung der Fragen ist es also notwendig zu beachten, dass „die Art der Frageformulierung die Antwortreaktionen erheblich beeinflussen kann"[Diekmann (2007), S. 458]. Beim Erstellen des Interviewleitfadens wird deshalb darauf geachtet, dass die Fragen möglichst offen formuliert sind und keine Wertung enthalten. So wird sichergestellt, dass der Befragte frei antworten kann und seine Meinung nicht beeinflusst wird.

Der Interviewleitfaden für die Befragung der Software-Entwickler enthält neben allgemeinen Informationen und zeitlichen Richtlinien die folgenden Fragen:

- Wie gehen Sie an die Programmierung mithilfe von neuen APIs heran?
- Welche Aufgabe hat die Dokumentation dabei zu erfüllen?

Diese beiden Fragen dienen dazu in das Thema einzusteigen und grundsätzlich herauszufinden, wie die Software-Entwickler an die Programmierung herangehen und wozu sie die Dokumentation hauptsächlich verwenden. So kann schon von vorneherein zwischen unterschiedlichen Lerntypen unterschieden werden und gleichzeitig die Frage beantwortet werden, welche Ziele und Aufgaben eine API-Dokumentation aus Sicht der Entwickler hat.

Daraufhin folgen direkt die Fragen zu der Dokumentation, die als besonders gutes Beispiel von dem jeweiligen Entwickler mitgebracht wurde:

- Nun bitte ich Sie mir ihre Lieblingsdokumentation vorzustellen. Welche ist das?

[39] Vergleiche Mayer (2009), S. 37f.
[40] Vergleiche Witzel (2000), S. 3

- Warum arbeiten Sie gerne mit dieser Dokumentation? Was ist besonders gut gelungen?
- Welche Elemente oder Abschnitte empfinden Sie als besonders hilfreich?
- Wenn Sie eine bestimmte Programmieraufgabe haben, wie suchen Sie nach bestimmten Informationen im Dokument?
- Was unterscheidet die ausgewählte Dokumentation von anderen?

Die ersten beiden dieser Fragen sind bewusst allgemein gehalten, um die Entwickler nicht in eine bestimmte Richtung zu drängen, sondern ihnen die Möglichkeit zu geben, sämtliche Themen anzusprechen, die für sie persönlich von besonderer Bedeutung sind.

Die dritte Frage zielt darauf ab, eine Liste zu erhalten, die diejenigen Inhalte der API-Dokumentation beinhaltet, die sich die Entwickler wirklich wünschen. Außerdem eröffnet diese Frage die Möglichkeit, zu bestimmten Elementen Nachfragen anzustellen.

Mit der vierten Frage wird der Frage nach der erwünschten Form der Navigation im Dokument nachgegangen, die in bisherigen Studien bisher noch nicht umfassend untersucht wurde. Es soll also herausgefunden werden, wie die Software-Entwickler sich durch das Dokument durcharbeiten und wie die Navigation durch Suchfunktion und passende Struktur erleichtert werden kann.

Die letzte Frage gibt dem Befragten die Möglichkeit, nochmals die wichtigsten Dinge hervorzuheben und Besonderheiten der vorgestellten Dokumentation zu erläutern.

Anschließend folgt die Befragung zu dem zweiten mitgebrachten Beispiel, das eine Dokumentation sein sollte, mit der der Entwickler ungern arbeitet und Probleme hat.

- Dann wechseln wir jetzt zu Ihrem schlechten Beispiel. Warum empfinden Sie diese Dokumentation als schlecht und welche Elemente der Dokumentation dafür verantwortlich?
- Was fehlt dieser Dokumentation?
- Was könnte verbessert werden?

Die erste Frage zielt darauf ab Hindernisse in API-Dokumentationen zu identifizieren und unnötige Informationen und Elemente aufzudecken. Die beiden anschließenden Fragen sind dann dazu da, zu ermitteln, was sich die Entwickler wünschen, um die Hindernisse zu überwinden. Außerdem gibt es ihnen nochmals die Möglichkeit wichtige Elemente aufzuzählen, die ihnen in den schlechten Dokumentationen fehlen. Durch diesen geänderten Blickwinkel können weitere für die Dokumentation notwendigen Informationen gefunden werden.

Alle diese Fragen sind im Interviewleitfaden enthalten, welcher die Grundlage für die Durchführung der Interviews bildete, die im folgenden Kapitel beschrieben sind.

4.2 Die Durchführung: Interviews mit Software-Entwicklern der 1&1 Internet SE.

An der Befragung beteiligt waren sechs Software-Entwickler der 1&1 Internet SE in Karlsruhe mit unterschiedlicher Berufs- und Programmiererfahrung, um eine möglichst repräsentative Gruppe zu erhalten. Alle waren männlich, zwischen 27 und 49 Jahre alt und hatten zweieinhalb bis 26 Jahre Berufserfahrung als Entwickler. Für die genaue Zusammensetzung der Gruppe siehe Tabelle 1.

Tabelle 1: Zusammensetzung der Gruppe der befragten Software-Entwickler (SE).

Kürzel	Alter	Berufserfahrung als Entwickler	Arbeitsbereich
SE1	48	22 Jahre	Access Development, VoIP
SE2	49	26 Jahre	Customer Self-Care Access (DSL & Mobile)
SE3	27	2,5 Jahre	Access Development, REST Services
SE4	35	7 Jahre	Access, DSL Geschäft
SE5	36	10 Jahre	Middleware-Entwicklung, Product Owner Architecture
SE6	36	10 Jahre	Home Access Processes, Konzeption und Implementierung

Sie wurden im Vorfeld dazu aufgefordert, sich ihre liebste API-Dokumentation auszusuchen und diese zum Interview mitzubringen, ebenso auch ein Negativbeispiel, um zu zeigen, wie eine API-Dokumentation nicht aussehen sollte.

Die Interviews wurden mit Einverständnis der Gesprächspartner für die spätere Auswertung aufgezeichnet. Zum einen durch eine Bildschirmaufnahmesoftware, die sowohl den Bildschirminhalt als auch den Umgebungston aufzeichnete und zum anderen durch eine Kamera, die zur Sicherheit den Interviewten und den Bildschirm filmte.

Die Entwickler stellten ihre Dokumentationsbeispiele vor. Anhand des Leitfadens wurde das Interview falls nötig gesteuert, doch viele der Interviewten hatten auch ohne die Leitfragen viel zu ihren mitgebrachten Dokumentationen zu erzählen. Die Interviews dauerten zwischen 20 und 40 Minuten.

Im Anschluss wurden die Interviews transkribiert. Um die Verständlichkeit zu erhöhen, wurden Füllwörter ausgelassen und umgangssprachliche Verkürzungen ausgeschrieben. Der nächste Schritt war die Auswertung der Interviews mithilfe des Analyseverfahrens nach Mühlfeld, welches im folgenden Kapitel beschrieben wird.

4.3 Das Auswertungsverfahren: Analyse nach Mühlfeld u. a.

Grundlage für die Auswertung sind die Transkripte der Interviews. Bei der Auswertung von Experteninterviews geht es vor allem darum, „Wissens- und Handlungsstrukturen, Einstellungen und Prinzipien theoretisch zu generalisieren"[Meuser u. Nagel (1991), S. 447] und „Aussagen über Eigenschaften, Konzepte und Kategorien zu treffen"[Meuser u. Nagel (1991), S. 447].

Im Falle der durchgeführten Befragungen ist besonders relevant, welche Aspekte der API-Dokumentationen für alle Software-Entwickler positiv oder negativ sind und welche die Meinungen spalten und in wie weit die Aussagen generalisiert werden können.

Das grundlegende Konzept für die Auswertung von Interviewmaterial ist die qualitative Inhaltsanalyse. In dieser wird das Datenmaterial durch die Bildung von Kategorien reduziert und den Ergebnissen ein Interpretationsrahmen gegeben [41].

Bei dem Auswertungsverfahren für Leitfadeninterviews nach Mühlfeld u. a. (1981) handelt es sich um ein sechsstufiges Verfahren, das es sich zur Aufgabe gemacht hat, möglichst effektiv im Interviewtranskript die „Problembereiche zu identifizieren, die den einzelnen Fragen des Leitfadens zugeordnet werden können"[Lamnek (1995), S. 206].

Die sechs Stufen des Verfahrens werden im Folgenden erläutert.

4.3.1 Markieren der Antworten

Im ersten Schritt werden alle Textstellen markiert, die offensichtliche Antworten auf die Fragen des Leitfadens sind. Dabei werden diese gleich der jeweiligen Frage zugeordnet. „Diese Stufe ist vor allem eine Orientierung an Fakten"[Mühlfeld u. a. (1981), S. 336], es können aber auch schon generelle Vermutungen formuliert werden, falls sich eine bestimmte Aussage durch das ganze Interview zieht.

Bereits beim Markieren der Antworten in den Entwicklerinterviews sind einige Themengebiete aufgefallen, welche die Antworten der Entwickler prägten. Diese wurden schon in diesem Schritt gesammelt, um sie später im Kategorieschema zu verwenden.

4.3.2 Einordnen in ein Kategorieschema

Hier geht es um die „Extraktion von Einzelinformationen"[Mühlfeld u. a. (1981), S. 337]. Dazu werden die Informationen in ein Schema eingeordnet, das die Antworten kategorisiert. Dieses Kategorieschema entsteht aus vorhergehenden Studien und Erwartungen, wird aber auch durch die Interviewaussagen erweitert[42]. Der Interviewtext wird

[41] Vergleiche Flick (2007), S. 409
[42] Vergleiche Schmidt (2009), S. 448ff.

so interpretiert, dass die Aussagen in die Kategorien passen. Dabei soll die „Interpretation so regelgeleitet und so explizit wie möglich sein"[Mayring et al. (2008), S. 11], um Verfälschungen zu vermeiden.

Aus den bisherigen Studien und den Aussagen der Gesprächspartner ergaben sich die folgenden Kategorien:

- Typische Fragen an die API und die Dokumentation,
- beliebte Art der Dokumentation,
- Aufgaben und Ziele der Dokumentation,
- Erwartungshaltung der Entwickler,
- wichtige Informationen, die enthalten sein sollen,
- unnötige Informationen,
- hilfreiche Eigenschaften und Elemente,
- nicht hilfreiche Elemente,
- wodurch Dokumentation schlecht wird,
- Ansprüche an Formatierung und Struktur,
- erwünschter Umfang,
- Zielgruppenabhängigkeit,
- Aktualität,
- externe Dokumentation / Referenzdokumentation,
- Bedeutung von Codebeispielen,
- Code als Dokumentation,
- und ein paar wenige Aussagen, die für sich alleine eine Kategorie bilden und unter Sonstiges zusammengefasst wurden.

Diesen Kategorien wurden die zuvor markierten Antworten zugeordnet, wodurch ein guter Überblick über alle Aussagen der Befragten gewonnen werden konnte. Die erste Zuordnung erfolgte in ein tabellarisches Schema, da so eine gute Übersicht über alle Aussagen und Kategorien möglich und das Verschieben in andere Kategorien einfach war.

4.3.3 Herstellen innerer Logik

Nun folgt der nächste Schritt, in dem nach dem Zerlegen wieder eine Logik zwischen den einzelnen Informationen hergestellt wird. Dabei kommt es „darauf an, sowohl bedeutungsgleiche Passagen als auch sich widersprechende Informationen zu berücksichtigen"[Mayer (2009), S. 50]. Da es sich um die innere Logik handelt, geht es hier

nicht darum, die Informationen anhand eines Schemas abzuwägen, sondern darum, die Informationen innerhalb des Interviews abzuwägen[43].

Bei diesem Abwägungsprozess steht die „Identifizierung besonders prägnanter Passagen"[Mühlfeld u. a. (1981), S. 337] des Interviews im Vordergrund. Es geht nicht um die Einzelinformation, sondern um die Aussagekraft und Konsistenz im Gesamten. So können die Textstellen herausgefiltert werden, die im Rahmen der Fragestellung von besonderer Relevanz sind.

4.3.4 Innere Logik schriftlich verfassen

Die zuvor erfasste innere Logik wird nun schriftlich festgehalten, um die Gewichtung der einzelnen Passagen zueinander „noch weiter detailliert, differenziert und präzisiert"[Mühlfeld u. a. (1981), S. 337] darzustellen.

Dies ist die letzte Stufe der inhaltlichen und interpretativen Auswertung. Die folgenden Auswertungsschritte dienen der Darstellung der Ergebnisse.

4.3.5 Text mit Interviewausschnitten

An dieser Stelle wird „die Transkription manuell auseinandergenommen und die Textpassagen thematisch geordnet sowie die Auswertung erstellt"[Mühlfeld u. a. (1981), S. 338]. Am Ende wird ein Text erstellt, der die Ergebnisse beschreibt und mit Interviewausschnitten belegt [44].

Um diesen Text zu erstellen, wurde zunächst das in Schritt 2 [45] erstellte Kategorieschema bearbeitet. Während die erste Kategorisierung in tabellarischer Form erstellt wurde[46] wurde in diesem Schritt zu einer Listenform übergegangen, da so ein besserer Überblick über die einzelnen Aussagen innerhalb der jeweiligen Kategorie gewonnen werden konnte.

Die Aussagen innerhalb der Kategorien wurden verschiedenen Unterkategorien zugeordnet. Diese ergaben sich durch die Paraphrasierung der Aussagen der Entwickler. Alle Aussagen, die dieselbe Grundaussage hatten, konnten so unter einer Paraphrase zusammengefasst werden, die die Unterkategorie darstellt. So zeigte sich, welche Themen besonders häufig angesprochen wurden.

Außerdem konnte durch das wiederholte Sichten der Aussagen nochmals überprüft werden, ob sie wirklich der richtigen Kategorie zugeordnet sind oder ob sie doch besser in eine andere passen. Diese erneute Sortierung ermöglichte es Kategorien

[43] Vergleiche hierzu Lamnek (1995), S. 207
[44] Siehe Kapitel 4.4
[45] Siehe Abschnitt 4.3.2
[46] Siehe Abschnitt 4.3.2

zusammenzufassen und einige umzubenennen. Die in Abschnitt 4.3.2 aufgezählten Kategorien entsprechen den endgültigen Kategorien nach diesem Schritt.

Die so erhaltenen Kategorien und Unterkategorien dienten als Grundgerüst für die Ergebnisformulierung.

Die in den Texten zur inneren Logik verfassten Grundaussagen der einzelnen Entwickler wurden beim Zusammenstellen der Ergebnisse verwendet, um einen Kontext zwischen verschiedenen Aussagen herzustellen. Grundlage für den eigentlichen Text waren die im Kategorieschema gesammelten Antworten. Diese wurden innerhalb der jeweiligen Kategorien thematisch sortiert und unter passenden Überschriften gruppiert. Anhand dieser Überschriften wurden die Ergebnisse dann zusammengefasst und mit jeweils passenden Zitaten versehen.

4.3.6 Darstellung der Auswertung

In der letzten Stufe wird aus dem Auswertungstext eine Präsentation erstellt. Dabei werden keine weiteren Interpretationen durchgeführt. Dieser Schritt kann im Rahmen dieser Arbeit vernachlässigt werden, da die Beschreibung der Ergebnisse mithilfe von Interviewausschnitten ausreichend ist, um die Ergebnisse zu präsentieren.

Mühlfeld u. a. (1981) empfehlen die Stufen 4 („Innere Logik schriftlich verfassen”), 5 („Text mit Interviewausschnitten”) und 6 („Darstellung der Auswertung”) bei der konkreten Durchführung zu kombinieren, da transkribierter Text und die kategorisierten Informationen parallel gesichtet werden und sich die Schritte dadurch nicht wirklich trennen lassen. So reduziert sich die Auswertung auf vier Schritte: Markierung der Antworten, Kategorisieren der Informationen, Herstellen der inneren Logik und die Darstellung der Ergebnisse. Ebenjene Ergebnisse werden im nächsten Kapitel erläutert.

4.4 Ergebnisse

In diesem Kapitel sind die Ergebnisse der Experteninterviews dargestellt, die sich durch die in Kapitel 4.3 beschriebene Auswertung ergaben.

Die Aussagen werden durch die Angabe in Klammern den jeweiligen Software-Entwicklern (SE1 - SE6)[47] zugeordnet, um eine Verbindung zwischen Aussage und Software-Entwickler darstellen zu können. Zum Beispiel bedeutet (SE2, SE5), dass diese Aussage auf den Kommentaren der Software-Entwickler 2 und 5 basiert. Bei „Text"(SE1) handelt es sich um ein wörtliches Zitat von Software-Entwickler 1.

4.4.1 Erwartungshaltung der Entwickler

Zunächst ist auffällig, dass viele Entwickler bereits eine gewisse Erwartungshaltung haben, wenn sie an eine neue API herangehen und die Dokumentation das erste Mal öffnen. Zunächst wird eine „tabellarische Form, in der die ganzen Packages erklärt werden"(SE5) erwartet sowie auch eine Erläuterung der Methoden (SE1, SE4, SE5, SE6).

tabellarische Form

Da die befragten Entwickler momentan alle hauptsächlich mit Java arbeiten, entspricht diese erwartete tabellarische Form der eines Javadocs[48] (SE1, SE4, SE5). Außerdem wird erwartet, dass dieses Javadoc schon bereitgestellt wird, wenn der Entwickler sich in ein neues Projekt und eine neue API einarbeiten soll (SE1).

Des Weiteren wurde erwähnt, dass es wünschenswert ist, wenn es gleichzeitig eine weitere Dokumentation gibt, „die mir diesen Dienst oder dieses Modul grundsätzlich beschreibt, also die grundsätzliche Funktionsweise"(SE1) und somit einen guten Überblick über die API liefert.

Überblick

4.4.2 Aufgaben und Ziele der Dokumentation

Fragt man die Entwickler, was ihrer Meinung nach die wichtigen Aufgaben und Ziele einer Schnittstellen-Dokumentation sind, so ist eines der wichtigsten Ziele, dass die Dokumentation die Informationen so darstellt, dass spezielle Informationen für eine bestimmte Programmieraufgabe einfach aufzufinden sind und ohne den Kontext der restlichen Dokumentation verständlich sind (SE2, SE3, SE4, SE5). Denn gerade in Anbetracht dessen, dass Zeit beim Programmieren meist knapp ist, ist es von Vorteil, wenn „man nicht die ganze Doku von Anfang bis Ende durchlesen muss, um überhaupt zu verstehen, wie man an den einzelnen Fall herangeht"(SE4).

einfache Auffindbarkeit der Informationen

[47] Die Software-Entwickler wurden bereits in Kapitel 4.2 und Tabelle 1 beschrieben.

[48] Javadoc ist ein Werkzeug mithilfe dessen aus Java-Quellcode eine HTML-Dokumentation generiert werden kann. (Quelle: Javadoc Homepage, http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html, abgerufen am 20.01.2016)

Beschränkung auf relevante Informationen

Dabei geht es auch um die Beschränkung auf die relevanten Informationen. Das Ziel ist, eine an die Schwierigkeiten der Software angepasste Dokumentation zu erhalten, denn „bei dem einen Stück Software liegen die schwierigen Stellen woanders als bei dem anderen"(SE3). Es ist somit wichtig, „dass es die relevanten Dinge sind"(SE2), die in der Dokumentation beschrieben sind, denn die trivialen Dinge ergeben sich bereits aus dem Code (SE3, SE6).

verschiedene Abstraktionsebenen

Trotz der Beschränkung sollte die Dokumentation es dennoch ermöglichen, die Schnittstelle zu verstehen, ohne weitere Literatur verwenden zu müssen (SE5), und sollte Informationen auf verschiedenen Abstraktionsstufen enthalten (SE3). Diese unterschiedlich detaillierten Ebenen können dabei auch durch Verlinkungen auf gegebenenfalls notwendige, tiefer gehende Informationen erreicht werden (SE2).

Es ist außerdem besonders „wünschenswert, wenn man eben nicht in die Implementierung rein gucken muss"(SE4), denn dies ist schließlich genau die Aufgabe einer Schnittstelle[49].

Zusammenarbeit zwischen Code und Dokumentation

Essentiell ist auch, dass die Informationen da zu finden sein sollten, wo auch gearbeitet wird, sprich in der Entwicklungsumgebung (SE2, SE6), denn so wird die Zusammenarbeit zwischen Code und Dokumentation gestärkt.

Programmierung erleichtern

Weiter ist es für Entwickler bei der Dokumentation von Bedeutung, dass die Programmierung durch bereits angegebenen oder generierbaren Code erleichtert wird (SE1, SE2). Dabei geht es zum Beispiel darum, dass Dateien mitgeliefert werden, die der Entwickler in ein „Tool bei sich stecken [kann] und das erzeugt ihm für seine Programmiersprache passenden Programmcode, um diese Schnittstelle aufzurufen"(SE1). Aber auch Codebeispiele (SE1) und eine interaktive Dokumentation, die die Möglichkeit bietet die Schnittstelle bereits mit vorhandenen Beispielen zu testen (SE2), erleichtert die Programmierung.

4.4.3 Wichtige Informationen, die in der Dokumentation enthalten sein sollten

Um die im vorherigen Abschnitt erläuterten Aufgaben zu erfüllen, sind bestimmte Informationen fundamental und sollten daher in einer API-Dokumentation immer enthalten sein:

allgemeine Hinweise

Vier der sechs befragten Software-Entwickler empfinden einen allgemeinen Einstieg als sehr wichtig, denn ein „paar einleitende Worte sind eigentlich schon sinnvoll"(SE6). Dabei geht es vor allem um Hinweise zur Benutzung der API (SE4, SE6), um die Konzepte die dahinter stecken (SE3), eine allgemeine Beschreibung der Struktur der Schnittstelle (SE2) und eine Übersicht über die Arbeitsweise (SE5). Für die Entwickler sind dabei Fragen wie „Wie ist das aufgebaut? Welche Systeme ruft das Ding [die

[49] Aufgabe einer Schnittstelle ist es, dass diese genutzt werden kann, um etwas für ein System zu programmieren ohne die genaue Implementierung zu kennen (vergleiche Kapitel 2.1).

Schnittstelle] da auf? Also welche Abhängigkeit nach Außen gibt es?"(SE2) substantiell.

Eine allgemeine Übersicht sollte dabei allerdings nicht zu lang geraten und sich auf die wesentlichen Dinge beschränken[50]. Es geht darum, dass dieser Teil „strukturiert aufgebaut und eben auch kurz genug, dass es einem nicht langweilig wird. Aber lang genug, dass man eben die Hintergründe versteht"(SE5).

allgemeine Konzepte

Bei den allgemeinen Hinweisen zur Benutzung sollte es vor allem um „Sachen, die nicht schon im Code drin stehen, sondern [um] Sachen, die man als Nutzer wissen muss, [also] die Bedeutung des Ganzen"(SE4) gehen. Um diese Bedeutung geht es auch bei den Konzepten, die erläutert werden sollen. Für die in der API aufrufbaren Funktionen sollte nicht nur angegeben werden, was am Ende heraus kommt, sondern auch warum, also aus welchem Grund das passiert (SE3).

Dies geht einher mit der Frage, die einer der Entwickler zunächst von der Dokumentation beantwortet haben möchte:

> „Wie will sie [die Library[51]] benutzt werden?" (SE4).

Übersicht über Methoden und Dienste

Andere Fragen der Entwickler sind „Wie mache ich genau diesen einen Aufruf, den ich hier machen soll?"(SE3) und ähnlich sind die Fragen „Welche Operationen gibt es? Was machen die?"(SE6). Daher ist es wichtig, dass die Dokumentation eine Übersicht über alle Methoden und Dienste, die die API anbietet, gibt (SE1, SE6):

„Sachen, die einzelne Methoden betreffen, das sollte natürlich in der API-Dokumentation [...] sein"(SE6), da sind sich die Entwickler einig. Dabei geht es hauptsächlich um „eine Auflistung: Welche Parameter gehen da rein? Welche Operationen gibt es? Was machen die?"(SE6).

Beschreibung der Parameter

Diese eben bereits erwähnten Parameter der API spielen für die Entwickler eine große Rolle. Vier der Befragten hielten die Erklärung der einzelnen Parameter für den elementarsten Teil einer API-Dokumentation (SE1, SE4, SE5, SE6). Beim Lernen einer neuen API haben die Entwickler folgende Fragen bezüglich der Parameter:

- Wie heißt der Parameter und was tut er? (SE1, SE6) Allerdings ist hierbei zu beachten, dass Funktionen, die der Name bereits aussagt, nicht weiter erläutert werden müssen[52].
- „Sind da vielleicht Parameter drinnen, die unerwartet sind?"(SE6)
- „Wie müssen die Parameter befüllt werden?"(SE6) und „Wie muss denn diese Information jetzt formatiert sein, damit der Service damit umgehen kann?"(SE6)
- „Darf er [der Parameter] null sein? Darf er nicht null sein?"(SE4)

[50] Mehr zum erwünschten Umfang für die Dokumentation im Allgemeinen wird in Abschnitt 4.4.9 erläutert.
[51] Eine Library ist eine Zusammenstellung der im Code verwendeten Klassen oder Templates.
[52] Siehe auch Abschnitt 4.4.4

- „Muss ich diesen Parameter angeben? Ist der jetzt optional?"(SE6)

Es reicht also nicht aus nur die reine Funktion eines Parameters zu erklären, es sollten auch die Vorgaben für diesen beschrieben werden.

Ein Beispiel hierfür sind zum Beispiel die Formatierung der Eingaben:

> Es „wird gerne mal vergessen zu schreiben, dass der Name maximal 40 Zeichen haben darf, weil er sonst im dümmsten Fall einfach abgeschnitten wird. Das ist natürlich eine Information, die interessiert [...] natürlich vorher."(SE6)

Typisierung

Wenn ein Interface stark typisiert[53] ist, so sollten auch die Definitionen der verschiedenen Typen in der API-Dokumentation vorhanden sein (SE5).

Laufzeitverhalten

Da Parameter und Typen häufig auch mit dem Laufzeitverhalten der Software zusammenhängen, sollte deren Auswirkungen auf die Laufzeit in der Dokumentation beschrieben werden (SE4, SE5).

was selbsterklärend ist, muss nicht dokumentiert werden

Bei allen diesen Erläuterungen von Methoden, Klassen, Parametern usw. sollte allerdings eines beachtet werden: Beschreibungen, die genau das sagen, was auch schon die Benennung selbst aussagt, sind unnötig (SE1, SE2, SE3, SE4):

> „Wenn etwas selbsterklärend ist, dann ist es selbsterklärend. Man nutzt auch so gut es geht Namen von Klassen, Methoden, Parametern, Exceptions, damit sie auch selbsterklärend werden."(SE4)

Ein gutes Beispiel hierfür wurde von einem der Befragten beschrieben:

> Es gab „Tools in den Entwicklungsumgebungen, die haben automatisch so eine Code-Dokumentation geschrieben für einen. Das heißt, wenn man da eine Methode hatte, die hieß „GetFirstName", dann hat der automatisch daraus generiert: gets the first name of the person object. Das heißt da steht nichts drin außer dem, was der Methodenname schon sagt."(SE2)

Zurück zu den Informationen, die in der Dokumentation vorhanden sein sollten. Fünf der interviewten Software-Entwickler haben einen weiteren Aspekt angesprochen, der von einem der Gesprächspartner auf den Punkt gebracht wurde:

Fehlermeldungen

> „Fehlerfälle - die will ich sehen."(SE3)

Es geht also um die Beschreibung der möglichen Fehlermeldungen, die bei der Programmierung mit der Schnittstelle auftreten können und deren Bedeutung erklärt sein sollte (SE1, SE3-SE6), am Besten auch mit Beispielen (SE3). Und zwar „auch die Fehlermeldungen, die nicht offensichtlich fest in der Schnittstelle spezifiziert sind, sondern die immer kommen können"(SE6).

[53] Als Typisierung bezeichnet man den Vorgang, in dem Daten in ein Typsystem eingeordnet werden, das es ermöglicht, Variablen in ihrer Art und ihrem Wertebereich einzuschränken. (Quelle: http://www.itwissen.info/definition/lexikon/Typisierung.html, abgerufen am 20.01.2016)

Neben der Erklärung was der Fehler bedeutet, ist eine wichtige Information „auch ein Hinweis falls notwendig – Was mache ich jetzt? Kann ich es ignorieren, kann ich es weiterreichen, soll ich sie abfangen?"(SE4) und „was passiert mit den Daten, die man nicht zurück bekommt?"(SE5).

Außerdem wurden auch Beispiele und Beispielcode als essentielle Inhalte für die Dokumentation genannt. Dies wird in Abschnitt 4.4.6 näher erläutert.

4.4.4 Unnötige Informationen

Im Gegensatz zu den im vorherigen Abschnitt genannten Informationen, gibt es auch einige, die von den Software-Entwicklern eher als störend oder zumindest überflüssig empfunden werden, denn sie vergrößern den Umfang der Dokumentation unnötig und erschweren so das Arbeiten[54] (SE3).

Informationen für Betreiber der Schnittstelle

Am häufigsten wurden in diesem Zusammenhang Informationen genannt, die für den API Benutzer nicht relevant sind, sondern nur für die Betreiber der Schnittstelle (SE1, SE3, SE4, SE6). Dies betrifft zum Beispiel Erläuterungen über „Caching. Das ist für die relevant, die den Service betreiben. Das gehört hier eigentlich gar nicht hin"(SE3).

Oft stammen diese für Schnittstellenbenutzer überflüssigen Information daher, „dass Dokumentation im Rahmen eines Projektes in der Projektdokumentationen quasi mitgeliefert wird und verteilt wird. Und danach ist das Projekt vorbei und man müsste quasi in diesem Berg von Projektinformationen sich wieder die relevante Info raus ziehen"(SE3). Wird dies nicht vom Anbieter der Schnittstelle gemacht, so wird es für den Benutzer unangenehm.

Dazu zählen auch Hinweise, welche die Implementierung betreffen. Dies ist „vor allen Dingen problematisch, [...] weil Implementation sollte sich ja ändern dürfen"(SE4). Auch welche weiteren Dienste und Technologien von der Schnittstelle aufgerufen und verwendet werden, die aber vom Benutzer nicht selbst angesprochen werden müssen, ist in der API-Dokumentation nicht relevant (SE6).

Duplizierung des Codes

Ein weiteres „Problem an Dokumentation ist, dass es zum Teil den Code dupliziert"(SE3). Daher ist es unnötig, nochmals extra zu dokumentieren, was schon im Code steht, wie eben die bereits in Abschnitt 4.4.3 angesprochenen Benennungen der Parameter, Methoden etc.

Auch wurde erwähnt, dass es unnötig sei, jede Zeile des Codes zu erklären (SE4). Es kann meist davon ausgegangen werden, dass ein Entwickler, der mit der Schnittstelle arbeitet, sich zumindest grundsätzlich in dem Themengebiet auskennt[55].

[54] siehe auch Abschnitt 4.4.9

[55] Natürlich muss dabei die Zielgruppe, an die sich die Dokumentation richtet, beachtet werden. Siehe dazu Abschnitt 4.4.12.

4.4.5 Hilfreiche Eigenschaften und Elemente

Interaktivität

Für einige der Entwickler ist Interaktivität eine besonders attraktive Eigenschaft, da die Möglichkeit die Funktionen der API ausprobieren zu können, den Entwicklern ein Gefühl für die Schnittstelle gibt und so das Programmieren erleichtert (SE2, SE4, SE6). Unter dieser Interaktivität einer API-Dokumentation versteht einer der Entwickler das Folgende:

> „Man kann im Prinzip auch die API ausprobieren, [...] ich kann die auch tatsächlich mal mit Parametern befüllen, kann mir Ergebnisse angucken, [...] kann auch Beispielcode eingeben, wie die Ergebnisse aussehen, was mir einfach als Entwickler hilft."(SE6)

Möglichkeit reale Daten zu testen

Besonders die Möglichkeit eigene reale Daten testen zu können (SE6) und „die tatsächlichen Objekte von dem echten Server zurück"(SE2) zu bekommen, ist besonders hilfreich.

Ein anderer Entwickler beschreibt seine ideale Dokumentation so:

> „Da sieht man aufgelistet, was es für Widgets gibt. Dann klickst du das an, dann siehst du dieses Widget in Aktion - also im Prinzip du siehst direkt wie die Tabelle aussieht, was du damit machen kannst und untendrunter siehst du gleich den Code, den du brauchst, um diese Tabelle bei dir direkt einzubinden. Und dazu noch ein Button zur Referenzdokumentation, wo alle Variablen noch erklärt sind. Und das ist eigentlich ideal."(SE4)

interaktive Elemente Links

Sprich eine gute Dokumentation ist geprägt von interaktiven Elementen, wie Buttons, Dropdown-Boxen und Links, denn durch diese können Informationen zusammengefasst werden. Die Details zu gewissen Themengebieten erhält der Benutzer nur bei Bedarf angezeigt, so wird die Dokumentation übersichtlicher (SE1, SE3, SE5, SE6).

Gerade diese Verweise auf weitere Informationen geben dem Autor der Dokumentation die Möglichkeit, den verschiedenen Ansprüchen bezüglich des Umfangs[56] gerecht zu werden.

Doch nicht nur Verweise auf tiefer gehende Informationen sind gerngesehen, sondern auch Verweise auf den Quellcode sind hilfreich (SE1, SE6):

> „Zum Beispiel ist hier auch beschrieben, wo in welchem File finde ich was in dieser Quellcode-Datei. Und dann kann ich da hin springen und ich habe den originalen Quellcode und kann da schnell mal was nachgucken."(SE1)

Suchfunktion

Bei all den Informationen an unterschiedlichen Orten ist eine Suchfunktion praktisch und nahezu unumgänglich (SE1). Auch die Hervorhebung der verschiedenen Parameter, die in der Dokumentation beschrieben werden, vereinfachen die Suche nach den passenden Erklärungen (SE6).

[56] Siehe Abschnitt 4.4.9

Eine weitere Möglichkeit Informationen zu bestimmten Themen leichter auffindbar zu machen, ist die automatische Generierung einer Dokumentation, die nur die für eine bestimmte Aufgabe interessanten Inhalte enthält (SE1).

Grafiken

Darstellungen in grafischer Form sind für Beziehungen und komplexe Strukturen gut geeignet (SE1, SE2, SE5). Einer der Befragten beschrieb sehr gut, wann eine grafische Darstellung sinnvoll ist:

> „Wenn es aber darum geht, dass ich ein System weiterentwickle, dann brauch ich noch eine andere Darstellung. Da muss ich wissen – wie ist das aufgebaut? Welche Systeme ruft das Ding da auf? Also welche Abhängigkeit nach Außen gibt es? Also da brauch ich dann auch noch mehr Dokumentation der Systemumgebung und der Komponenten aus denen die Software besteht. Und das finde ich immer ganz angenehm, wenn das grafisch ist."(SE2)

How Tos

Als weitere hilfreiche Elemente, die eine gute API-Dokumentation ausmachen, wurden auch Schritt für Schritt Anleitungen, sogenannte How Tos, genannt. Diese sind besonders von Vorteil, wenn man bereits ein gewisses Grundwissen hat, „dann findet man sich da schnell zurecht"(SE5).

Des Weiteren wurde die Möglichkeit direkt den Code zu generieren, der dann in den eigenen Code kopiert werden kann[57], als eine gute Eigenschaft beschrieben (SE6).

4.4.6 Bedeutung von Codebeispielen

Der hohe Stellenwert von Codebeispielen wurde bereits in der Studie von Robillard und DeLine Robillard u. DeLine (2011) erkannt. Daher wurden die Entwickler im Interview explizit darauf angesprochen, falls sie nicht schon von selbst auf das Thema zu sprechen kamen.

Codebeispiele generell hilfreich

Alle sechs bezeichneten Codebeispiele als generell hilfreich, lediglich einer sah diese etwas kritisch.

Die Befragten waren sich einig, dass es die beste Hilfe ist, zu sehen wie ein anderer die Module bereits verwendet hat (SE1-SE6):

> „Wenn man solche Quellcode Module benutzen soll, möchte man dann eigentlich auch sehen, wie hat das jemand anders schon mal benutzt."(SE1)

Die Beispiele sind gut geeignet, um einen ersten Eindruck zu vermitteln, wie die Schnittstelle verwendet werden muss (SE6). Und meist können Beispiele auch komplexere Informationen besser erklären als simple Erläuterungen (SE3, SE6).

Weiter eignen sich Beispiele, um relevante Teile der API hervorzuheben und den korrekten Umgang mit den Parametern zu zeigen (SE1, SE6).

[57] Siehe auch Abschnitt 4.4.6

Ein Beispielcode kann auch verwendet werden, um die eigene Implementierung zu testen:

> „Beispielcode hilft halt oft zu verstehen, wie man das befüllt und auch leichter Tests für seine eigene Implementierung zu schreiben. Wenn man mal ein realistisches Beispiel hat, dass man mal in seinen Test kopieren kann, sagt der halt auch mehr aus, als wenn ich jeden Text nur mit fünf Buchstaben "Lorem Ipsum" fülle."(SE6)

Codebeispiele gehören in die externe Dokumentation, die eigentliche API-Dokumentation würden sie überladen. Hier kommt die kritische Sicht des einen Befragten ins Spiel:

> „Beispiele sind schön und gut, aber wenn sie falsch sind, sind sie noch viel schlimmer."(SE3)

Testcode

Daher sollte auch immer zusätzlich zu den Beispielen ein Testcode existieren, um überprüfen zu können, ob diese auch funktionieren (SE3).

Das Problem ist, dass Beispiele keinesfalls änderungsresistent sind und somit fehleranfällig. Wenn sich der Code ändert, muss daran gedacht werden, dass auch die Beispiele verändert werden müssen (SE3).

4.4.7 Wodurch Dokumentation schlecht wird

Es gibt auch viele Dinge, die dazu führen, dass eine Dokumentation schlecht wird und manchmal sogar mehr hinderlich als hilfreich ist.

Neben dem sehr häufig auftretenden Problem, dass „die Entwickler [...] meist keine Zeit für eine ordentliche Dokumentation"(SE5) haben und schon allein darunter die Qualität leidet, gibt es einige weitere Dinge, durch welche die Dokumentation als eher schlecht empfunden wird.

fehlende und veraltete Informationen

Das größte Problem ist leider generell eine fehlende oder stark veraltete[58] Dokumentation (SE5, SE6). Aber auch die Unterschlagung von Informationen in der Dokumentation ist problematisch und kommt häufig im Bereich der Fehlerbehandlung vor. „Die [Informationen] werden dann einfach unterschlagen, weil man einfach immer vom Positivfall ausgeht"(SE5).

Weitere Informationen, die häufig fehlen, sind die über die Befüllung der Parameter (SE5,SE6):

> „Das wird einfach auch vergessen und unterschlagen. Das sind so Kleinigkeiten eigentlich, die einfach wahnsinnig wichtig sind."(SE5)

[58] Weitere Aspekte zur Aktualität der Dokumentation werden in Abschnitt 4.4.11 näher erläutert.

Als „nicht so toll" bezeichnete daher einer der Entwickler Dokumentationen, die eigentlich nur aus Beispielen bestehen und keinerlei weitere Erklärungen enthalten (SE5).

fehlende Grundlagen

Außerdem führen zum Beispiel fehlende Grundlagen dazu, dass zu viel selbst erarbeitet werden muss und das kostet viel Zeit (SE5).

Manchmal werden die Entwickler, die mit der Schnittstelle arbeiten, durch fehlende Grundlagen und fehlende Dokumentation dazu gezwungen „in den eigenen Code relativ viel zu schreiben, warum man diese Schnittstelle so befüllt oder sich eine Wiki-Seite oder Dokumentation anzulegen wie man mit der Dokumentation umgeht. Und das hat natürlich immer den Nachteil, wenn sich irgendwann die eigentliche Schnittstelle ändert, dann wird diese Dokumentation, die man als sozusagen Klient entwickelt hat"(SE6), fehlerhaft.

Auch mangelnde Konsistenz in Benennungen ist verwirrend (SE3), genauso wie unbenannte, beziehungsweise ungenügend benannte Methoden, Variablen usw. Einer der Interviewten beschrieb dieses Problem so:

> „Wenn ich dann merke, dass Methoden zum Beispiel viel zu kurz gemacht werden oder Variablen einfach A, B oder so was sind oder meine IDE dann im schlimmsten Fall, also die Entwicklungsumgebung, nicht mehr imstande ist die Namen der Parameter wiederherzustellen, dass da also nur noch StringA, StringB, StringC steht, [...] dann ist es halt wirklich endgültig vorbei."(SE5)

nicht ausreichende Verlinkung

Was ansonsten häufig stört ist, wenn die Verlinkung nicht ausreichend ist und dadurch der Zugriff auf weitere Informationen nicht möglich oder umständlich ist (SE1). Umständlich ist zum Beispiel die Navigation zwischen vielen verschiedenen Dokumenten und Programmen, denn diese lenkt zu sehr ab (SE3).

zu viel Dokumentation im Code

Beim Lesen der Dokumentation in der Entwicklungsumgebung ergibt sich manchmal das Problem, dass zu viele HTML-Inhalte störend sind, da sie von den eigentlichen Informationen ablenken (SE3):

> „Das heißt ich schreibe nicht zu viel Dokumentation in den Code. Das ist nämlich ein Problem."(SE1)

Allerdings sind gerade HTML-Inhalte für eine übersichtliche Dokumentation mit Verlinkungen etc. häufig notwendig, was das Weglassen schwierig gestaltet und daher sorgfältig abgewogen werden muss.

Methodenbeschreibung in codeferner Dokumentation

Wie bereits in Abschnitt 4.4.3 erwähnt, ist es wichtig die einzelnen Methoden zu beschreiben. Allerdings macht es keinen Sinn in codeferner Dokumentation, wie zum Beispiel How Tos auf einer Webseite, Methodenaufrufe zu beschreiben. Diese sollten direkt in der Entwicklungsumgebung abrufbar sein (SE3).

Ein weiterer interessanter Effekt wurde wie folgt beschrieben: „wenn etwas markiert wird, hat es manchmal den Effekt, dass man es gerade deshalb überliest"(SE3). Das

zu viele Markierungen

heißt zu viele Markierungen führen manchmal zum Gegenteil, nämlich zum Ignorieren der markierten Information.

Diese Markierungen beschreiben bereits einen Teil der Formatierung, welche im folgenden Abschnitt näher betrachtet wird.

4.4.8 Formatierung und Struktur

einheitliche Formatierung

Auch an die Formatierung werden von den Entwicklern einige Ansprüche gestellt:

> „Die Formatierung muss zum Beispiel ja einheitlich sein, auch das ist schon ein Teil der Dokumentation."(SE1)

Noch besser ist es, wenn sich diese einheitliche Formatierung auf mehrere APIs anwenden lässt, wie es mit Frameworks möglich ist. Ein Befragter beschrieb den Vorteil solcher Frameworks folgendermaßen:

einheitliches Format durch Frameworks

> „Was auch ganz gut ist, dadurch dass hier ein einheitliches Format verwendet wird: Wenn ich viele APIs habe, die in der gleichen Form dokumentiert sind, finde ich relativ schnell die Information, die ich brauche. Also ich habe eine Struktur, ich kann die Methoden erkennen. Das ist natürlich auch von Vorteil gegenüber jetzt einem Freitext, wo ich einfach nur mir zusammensuchen muss, was ich letztendlich brauche."(SE6)

Ein Beispiel für ein solches Framework ist Swagger, welches für REST-APIs verwendet werden kann. Zwei der befragten Software-Entwickler nannten dieses Interface als gutes Beispiel (SE2, SE6). Die Funktionsweise kann am Besten anhand der Demoseite erklärt werden (siehe Abbildungen 4.1 - 4.3).

> „Also was ich ganz positiv finde, ich hab auch ein paar Notizen mitgebracht dazu, ist einfach es gibt eine übersichtliche Seite, die mir erst mal zeigt: Was bietet die Schnittstelle überhaupt an Methoden an?
> Ich hab dann je nachdem für die einzelnen Methoden [...] eben auch eine Auflistung: Welche Parameter gehen da rein? Gegebenenfalls dann noch eine Beschreibung.
> Und dann eben auch die Menge der Fehlermeldungen."(SE2)

Abbildung 4.1 zeigt die Startseite der Demonstrationsseite, welche die API für einen imaginären Haustierverkauf („Petstore") zeigt. Es werden alle Methoden aufgezählt und kurz erläutert, die in der API verfügbar sind. Die Methoden sind für eine bessere Auffindbarkeit nach Objekten sortiert: Zuerst findet man alles, was die Tiere („Pets") betrifft, dann die Methoden, die Bestellungen („Petstore orders") betreffen und zum Schluss die Methoden, die den Benutzern („user") zugehörig sind. Die Listen sind nach Bedarf ein- und ausklappbar, um die Seite noch übersichtlicher zu gestalten und es dem Entwickler zu ermöglichen, nur für ihn gerade relevante Methoden anzuzeigen. Die farblichen Markierungen zeigen die Art der Methoden und ermöglichen eine

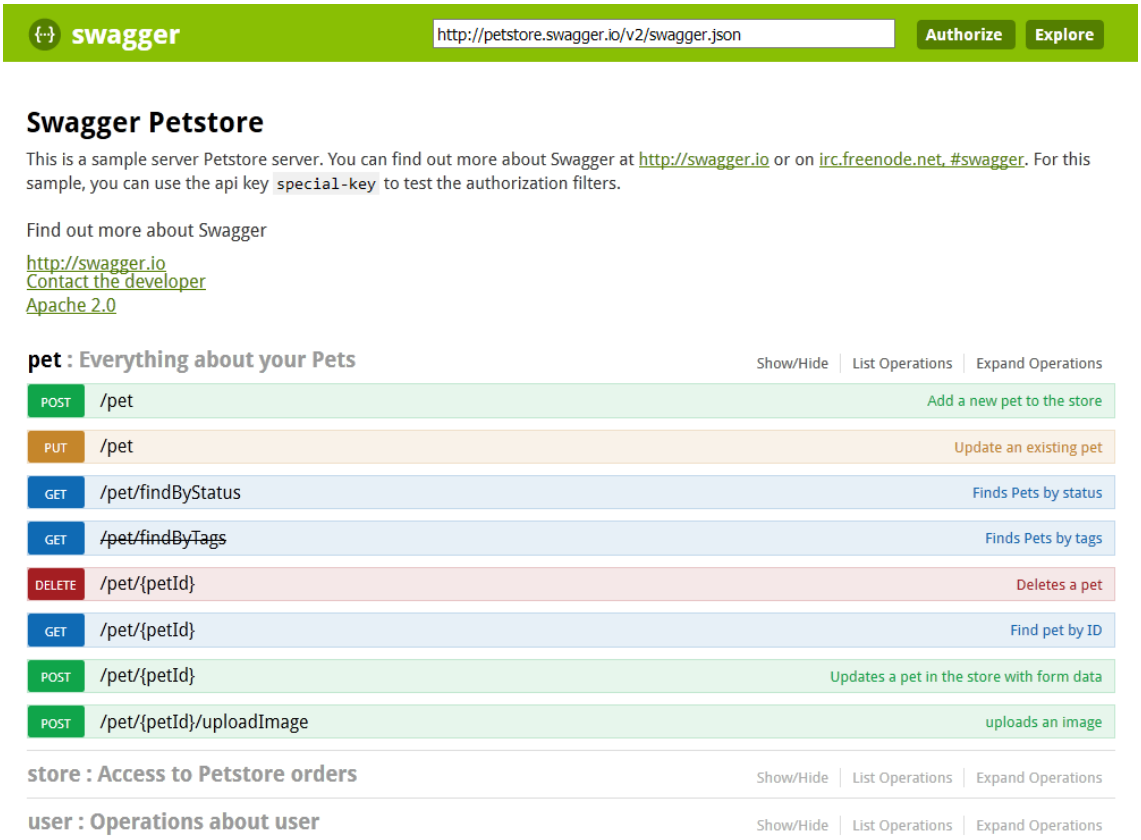


Abbildung 4.1: Startseite der Petstore API auf der Swagger Demonstrationsseite. Sie zeigt eine übersichtliche Auflistung der Methoden und eine kurze Beschreibung derer. (Quelle: Swagger Petstore API Dokumentation, http://petstore.swagger.io/#/pet, abgerufen am 04.04.2016)

schnellere Auffindbarkeit der gesuchten. Suche ich zum Beispiel nach einer Methode, um etwas zu löschen, sind nur rot markierte Einträge relevant.

Möchte man nähere Informationen zu einer der Methoden haben, so lassen sich diese ebenfalls wieder aufklappen und man erhält eine ausführlichere Beschreibung mit Beispielen. In Abbildung 4.2 wird die nähere Beschreibung der Methode „/pet/findByStatus" angezeigt, die es ermöglicht, Tiere in Abhängigkeit von ihrem Status zu suchen. Die näheren Informationen zu der Methode enthalten Hinweise zur Implementierung („Implementation Notes") und die Antwort der API im Normalfall („Response Class") und die Antwort, falls ein Fehler auftritt („Response Messages").

Von beiden Software-Entwicklern, die Swagger als gutes Beispiel vorstellten, wurde vor allem auch der „Try it out!" Button als positiv empfunden, denn die Interaktivität unterstützt den Lernprozess (SE2, SE6). Dieser ermöglicht es, die API auszuprobieren und Daten von einem echten Server zurückzubekommen (SE2). Beispielsweise wurde in Abbildung 4.3 nach Tieren gesucht, die verfügbar („available") sind. Durch das Klicken des „Try it out!" Buttons erhält man die Antwort der Schnittstelle auf die Anfrage.

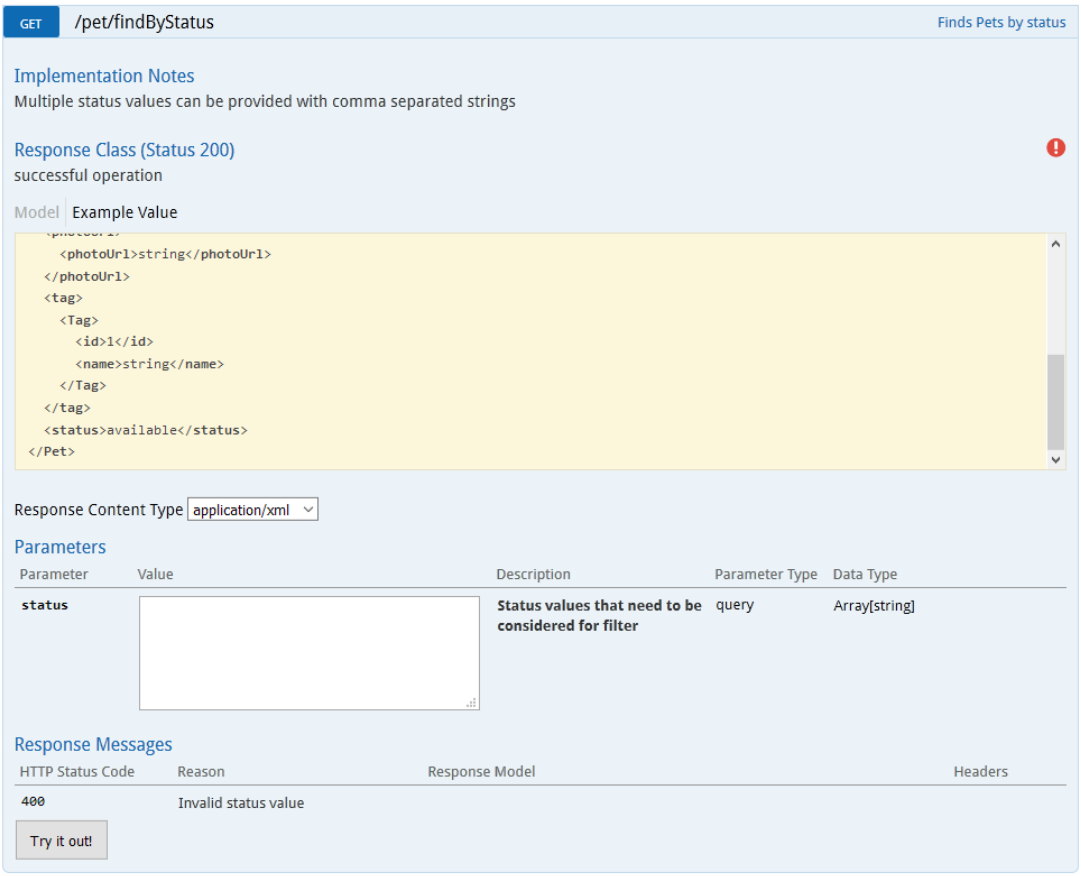


Abbildung 4.2: Beschreibung der Methode „/pet/findByStatus". Die Informationen enthalten Hinweise zur Implementierung („Implementation Notes"), die Antwort der API im Normalfall („Response Class") und die Antwort, falls ein Fehler auftritt („Response Messages"). (Quelle: Swagger Petstore API Dokumentation, http://petstore.swagger.io/#!/pet/findPetsByStatus, abgerufen am 04.04.2016)

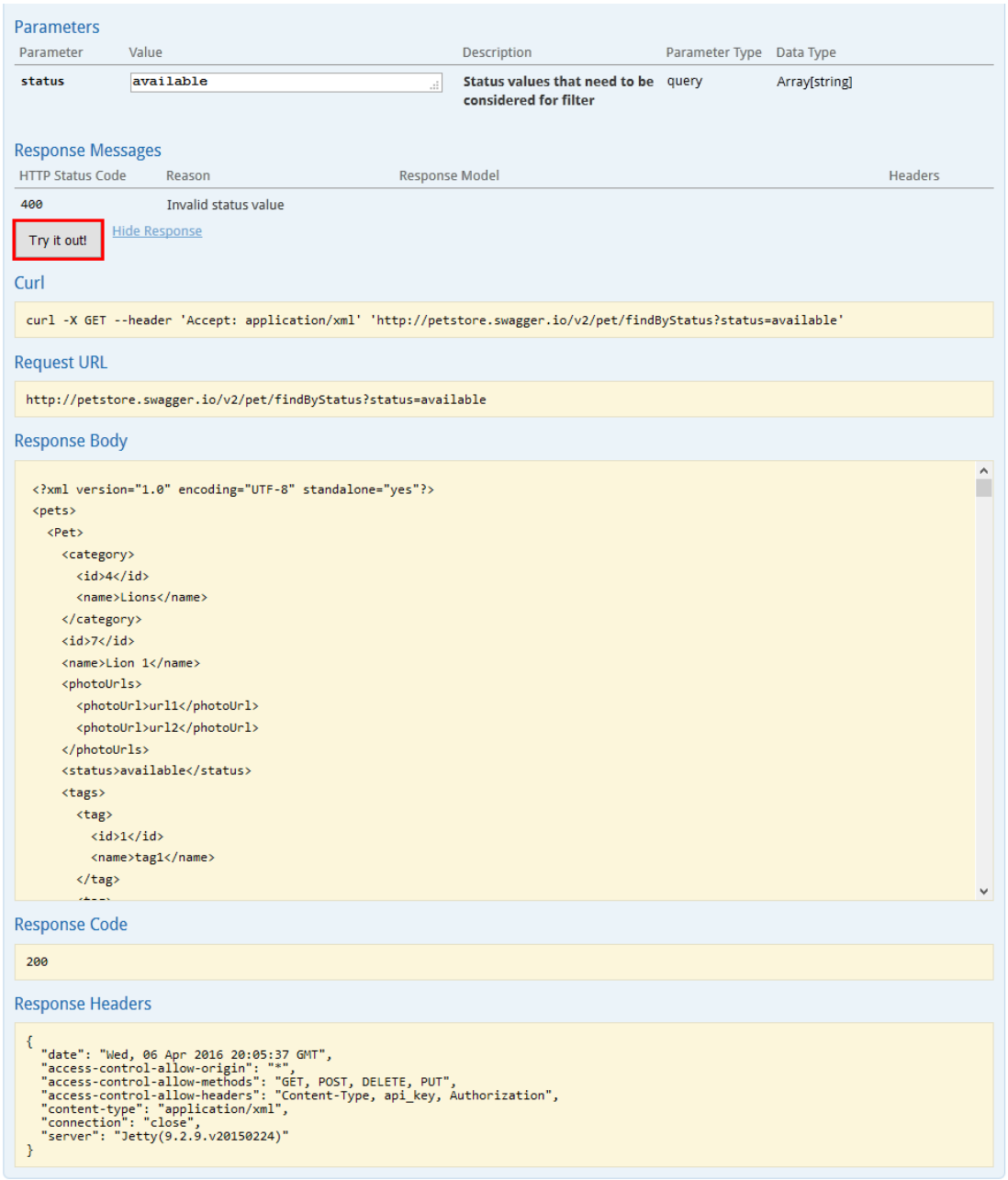


Abbildung 4.3: Antwort der Schnittstelle auf die Suche nach verfügbaren („available") Tieren mithilfe des „Try it out!" Buttons (rot markiert). (Quelle: Swagger Petstore API Dokumentation, http://petstore.swagger.io/#!/pet/findPetsByStatus, abgerufen am 04.04.2016)

Alle Dokumentationen von Schnittstellen, die mithilfe von Swagger erstellt werden, sehen grundsätzlich so aus wie das Beispiel auf der Demonstrationsseite. Swagger überzeugt die Entwickler daher zum einen durch die Interaktivität und zum anderen durch die einheitliche Formatierung und Struktur.

Doch nicht nur die einheitliche Struktur, sondern auch die Übereinstimmung der Struktur von Software und Dokumentation ermöglicht ein besseres Zurechtfinden in der Dokumentation (SE2).

gute Übersicht

Zudem sollte die Dokumentation so aufgebaut sein, dass die jeweils vorhandenen Informationen direkt ersichtlich sind (SE3, SE6). Dabei geht es darum eine übersichtliche Dokumentation zu haben, bei der man nicht erst nach langem Suchen feststellen muss, dass die gesuchte Information nicht vorhanden ist (SE3). Warum das so wichtig ist, erklärt einer der Entwickler so:

> „Weil ich durch so eine gewisse Struktur auch erkennen kann: Okay, oben steht sozusagen das was macht der Dienst. Dann kommen vielleicht in weiteren Absätzen auch mehr Details, die ich mir dann einfach auch beim Querlesen ersparen kann, wenn das jemand machen will."(SE6)

Allerdings sollte dabei darauf geachtet werden, es mit strukturellen Elementen nicht zu übertreiben, denn diese blähen die Dokumentation unnötig auf:

> „Gleichbleibende Struktur ist gut, wenn ich zu viel hab, hab ich aber mehr Heuhaufen als Nadel."(SE3)

einheitliche Terminologie

Auch einheitliche Terminologie ist wichtig:

> „Nimm immer dieselben Begriffe, dass es keine Verwirrung gibt."(SE3)

Man sollte also sicherstellen, „dass die Dokumentation eine gewisse Struktur hat, also jetzt auch nicht 30 Seiten Fließtext ohne Absatz und ohne Komma, sondern knapp und das Wesentliche beschreibt"(SE6).

4.4.9 Erwünschter Umfang

Wie schon zuvor erwähnt, haben die befragten Software-Entwickler eine bestimmte Vorstellung, welchen Umfang eine gut gelungene API-Dokumentation haben sollte.

hohe Präzession

Die Dokumentation sollte möglichst kurz und präzise sein, denn ein zu großer Umfang ist störend, wie es die folgenden Aussagen belegen:

> „Womit ich gar nichts anfangen kann ist, wenn ich so riesige Dokumente habe, in denen im Endeffekt nichts drin steht."(SE2)

> „Je weniger Dokumentation da ist, desto besser ist es eigentlich. Nur die relevante Information, die muss es eigentlich nur sein."(SE3)

> „Es ist auch wichtig, dass man schnell zum Punkt kommt."(SE4)

Ausschließlich große Dokumente, die den gesamten Code dokumentieren, sind zumindest bei vier der befragten Entwickler (SE1, SE2, SE5, SE6) weniger beliebt:

> „Große Dokumente sind immer schwierig und je größer sie sind, desto schneller veralten sie ja auch"(SE2).

Ist der gesamte Quellcode eines Projektes in einem einzelnen Dokument erklärt, so kann das zu viel werden (SE1), wie es auch dieser Entwickler beschreibt:

> „Deswegen ist so ein grober Überblick vielleicht manchmal hilfreich, aber die Fachlichkeit ist manchmal ziemlich komplex und es dauert ziemlich lange, um das wirklich zu verstehen und manchmal oder meistens brauche ich ja nur einen Teil davon."(SE2)

Mittelweg finden

In manchen Aspekten widersprechen sich die Ansprüche also. Der eine sagt „Je weniger Dokumentation da ist, desto besser ist es eigentlich"(SE3) und der andere sagt „im Zweifel natürlich lieber zu viel wie zu wenig"(SE4). Das Schwierige ist: „Man muss jeweils einen Mittelweg finden"(SE3).

Dazu bietet sich eine Kombination aus einer präzisen API-Dokumentation beim Code[59] und einer ausführlichen Referenzdokumentation an (SE3, SE4, SE5, SE6).

4.4.10 Aufsplitten in mehrere Dokumentationen

Um ein möglichst effizientes Arbeiten mit der Schnittstelle und deren Dokumentation zu ermöglichen, sollten lieber Verweise auf weitere Informationen verwendet werden, anstatt alles in einem Dokument zusammenzufassen (SE3, SE5, SE6). Sinnvoll ist hier eine Aufteilung in eine codenahe klassische API-Dokumentation und eine externe Dokumentation.

Aufteilung in codenahe und externe Dokumentation

Die codenahe API-Dokumentation[60] sollte Informationen enthalten, die den Code direkt betreffen, also zum Beispiel die einzelnen Methodenaufrufe (SE3). Informationen, die die gesamte Library betreffen, können zum Beispiel gut in eine externe Dokumentation ausgelagert werden (SE4, SE6):

> „Also die wichtigen und zentralen Informationen direkt oder nahe an der Schnittstelle und eben weitere Informationen über vielleicht auch Konzepte was dahinter steckt, wie man das im größeren Kontext verwendet, das dann auch eher auf einer externen Dokumentation. Einfach damit die eigentliche Dokumentation auch noch überschaubar bleibt."(SE6)

Besonders angenehm ist es, wenn die verschiedenen Dokumentationen parallel laufen und ein hin und her Wechseln dadurch einfach ist. Ein gutes Beispiel hierfür ist die Android API-Dokumentation, die von einem der Entwickler vorgestellt wurde.

[59] Dies wäre zum Beispiel ein Javadoc, wenn es um Javaentwicklung geht.
[60] Auch hier sind wieder Dokumentationen im Stil von Javadocs gemeint.

Google bietet Android-Entwicklern sowohl einen ausführlichen „API-Guide" als auch eine „Reference" Dokumentation, welche eine eher klassische API-Dokumentation darstellt.

Das von einem der Befragten genannte Beispiel bezieht sich auf Intents, die ein wichtiger Bestandteil der Android-Programmierung sind. Das besondere an seinem Beispiel ist für ihn das Folgende:

> „Man hat Grundlagen, aber gleichzeitig ist man auch schon in der Materie drin. [...] Weil man eben dadurch die perfekte Mischung hat. Eben schnell einzusteigen, aber auch umfassend informiert zu werden."(SE5)

Die Zusammenarbeit der beiden Dokumentationen ist für ihn das Wichtigste. Unter „Reference" findet der Entwickler kurz und knapp Erläuterungen zu Klassen, Daten und Aktionen wie in einem typischen Javadoc (siehe Abbildung 4.4). In dem zugehörigen „API-Guide" finden sich ähnlich wie in einem Lehrbuch ausführlichere Erklärungen zur Funktionsweise der Intents, deren Verwendung und Beispiele (siehe Abbildung 4.5).

> Der Guide „ist recht gut strukturiert, er ist nämlich thematisch strukturiert. Und da hat man klassisch Text, den man lesen kann. Man kriegt – das ist auch schön gemacht – hier durchaus mal ein paar Grundlagen beigebracht, ohne dass es zu sehr ins Detail geht."(SE5)

Durch entsprechende Verlinkungen in beiden Teilen wird das hin und her Wechseln erleichtert, wodurch die beiden Dokumentationsteile gut zusammenarbeiten und für den Entwickler das Abrufen der für ihn notwendigen Information erleichtert wird. Kennt er sich mit dem Thema noch nicht aus, kann er die Grundlagen in dem „API-Guide" lernen. Kennt er die Grundlagen bereits, reicht ihm die „Reference" Dokumentation meist aus.

Beim Erstellen der externen, codeferneren Dokumentation muss darauf geachtet werden, dass die Informationen allgemeiner und stabiler sein sollten, um die Fehleranfälligkeit zu reduzieren (SE3). Gerade diese Anfälligkeit für Fehler ist durch sich schnell ändernde Software sehr hoch, weshalb es wichtig ist, dass die Dokumentation immer aktuell ist.

4.4.11 Aktualität der Dokumentation

Ein Befragter sagte über die Übereinstimmung von Quellcode und Dokumentation

> „Man sollte sich bemühen es gleich zu halten, aber es gelingt in der Praxis nicht wirklich. [...] Man hat sehr oft die Situation, dass der Quellcode und die Dokumentation nicht mehr übereinstimmen. [...] Die Methode heißt dann inzwischen anders und erfüllt einen anderen Zweck. Das ist so ein großes Dilemma."(SE1)

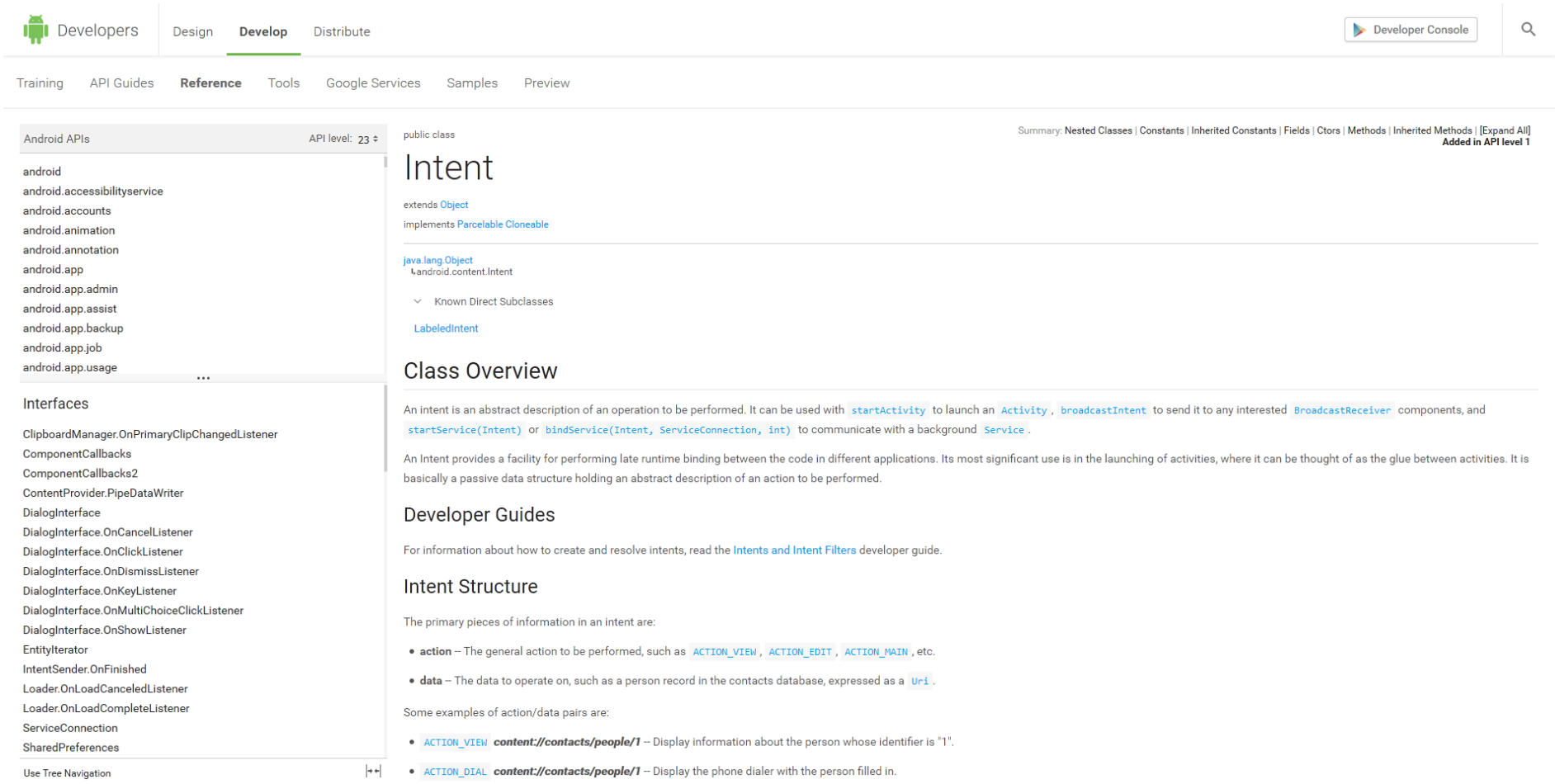


Abbildung 4.4: „Reference” Teil der API-Dokumentation von Android zum Thema Intents. Dieser Teil der Dokumentation zeigt eine klassische API-Dokumentation, in der alle APIs, Interfaces, Klassen etc. aufgelistet sind. Durch Anklicken erhält man die zugehörige kurze Beschreibung. (Quelle: Android Developers Website , http://developer.android.com/reference/android/content/Intent.html, abgerufen am 30.03.2016)

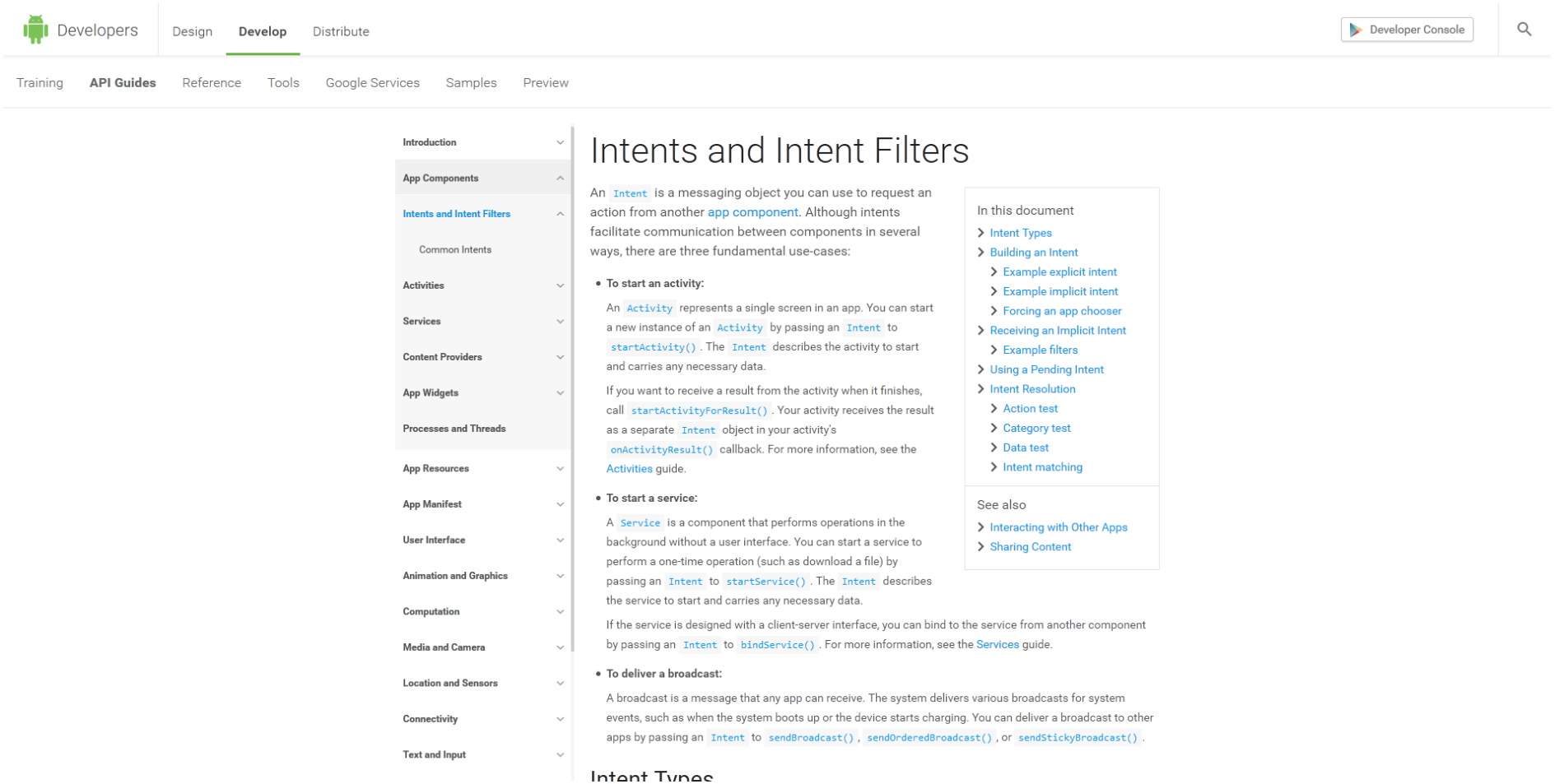


Abbildung 4.5: „API-Guide” Teil der API-Dokumentation von Android zum Thema Intents. Dies ist der ausführliche Teil der Dokumentation. Hier werden Grundlagen genauer erläutert und ausführliche Beispiele gegeben. Die blau unterlegten Links führen zum „Reference” Teil der Dokumentation. So wird eine gute Zusammenarbeit zwischen den beiden Teilen gewährt. (Quelle: Android Developers Website , http://developer.android.com/guide/components/intents-filters.html, abgerufen am 30.03.2016)

und ein anderer sagte sogar

> „Lieber keine Dokumentation als eine Lügendokumentation."(SE2)

Änderungsresistenz

Das zeigt, dass die Richtigkeit und Aktualität der Dokumentation das A und O sind. Daher sollte eine möglichst hohe Änderungsresistenz, gerade bei externer Dokumentation, das Ziel sein (SE1, SE2, SE3). Die Problematik ist dabei:

> „Man tendiert dazu, diese Dokumentation als Second Citizen zu betrachten. Doch der Code ändert sich so schnell und dann vergisst man die Dokumentation nachzuziehen."(SE1)

Hält man die Dokumentation möglichst nah am Code, so kann ein Veralten erschwert werden, da sie den Entwicklern der Schnittstelle beim Programmieren immer vor Augen ist. Je weiter weg die Dokumentation vom Code ist, desto eher wird sie vergessen (SE3).

Vorsicht bei Verweisen

Was gefährlich ist, sind Verweise von externer Doku auf einzelne Codestellen, da diese besonders schnell veralten. Aus dem Code heraus auf externe Dokumentation zu verweisen, ist dagegen nahezu problemlos (SE3).

automatisch generierte Dokumentation

Um Fehlern durch Code-Änderungen vorzubeugen, ist eine automatisch generierte Dokumentation das Beste (SE1, SE3). Dies zeigt sich allein schon daran, dass fünf der sechs interviewten Entwickler Frameworks vorstellten, die zur Generierung von Dokumentationen dienen. Der Vorteil der automatisch erzeugten Dokumentation ist dieser:

> „Die Idee ist natürlich, dass wenn ich jetzt an meinem Code etwas ändere, ändert sich die Doku automatisch mit, denn es wird ja aus dem Code generiert und aus dem, was ich dann habe. Und deswegen ist die dann immer auf dem neuesten Stand, weil sie aus dem Code direkt generiert ist."(SE1)

Man erhält also eine stets an den aktuellen Code angepasste Dokumentation. Diese automatisch generierte Dokumentation entspricht der bereits mehrmals erwähnten klassischen, codenahen API-Dokumentation, die erfahrenen Entwicklern meist ausreicht. Die manuell erstellte externe Dokumentation mit allgemeineren Informationen ist eher für unerfahrenere Benutzer relevant. Darüber wird im folgenden Abschnitt berichtet.

4.4.12 Zielgruppenabhängigkeit

Eine Dokumentation sollte stets die Zielgruppe im Auge behalten und so beschreiben das auch die befragten Software-Entwickler im Hinblick auf API-Dokumentation:

Blickwinkel des Benutzers

> „Die Dokumentation, vor allem in der API-Dokumentation, sollte immer so gehalten sein, dass sie semantisch ist, also immer aus dem Blickwinkel des Benutzers."(SE4)

Informationen, die nicht für den Benutzer, sondern nur für die Entwickler der API relevant sind, können und müssen daher weggelassen werden[61].

Doch auch innerhalb der Benutzer einer API gibt es unterschiedliche Gruppen mit verschiedenen Ansprüchen.

Eine ausführliche Dokumentation ist insbesondere notwendig, wenn die API für einen großen Benutzerkreis gedacht ist (SE3).

Unterschied zwischen erfahrenen und unerfahrenen Entwicklern

Erfahrene Entwickler benötigen weniger Dokumentation als Einsteiger, denn sie sind eher genervt von zu vielen selbstverständlichen Informationen. Unerfahrene Programmierer hingegen benötigen eben jenes Hintergrundwissen (SE1, SE4, SE5, SE6).

Dies ist ein Grund, warum es sinnvoll ist, die Dokumentation aufzusplitten[62], „weil oftmals, wenn man dann das kennt, braucht man ja den Kontext nicht mehr so sehr. Wenn man jetzt neu einsteigt, interessiert einen der natürlich auch"(SE6).

Ein anderer Software-Entwickler drückte die Abhängigkeit der Dokumentation vom Kenntnisstand der Benutzer so aus:

> „Wenn man sich mit dem Thema sowieso schon so ein bisschen auskennt, dann ist auch vor allen Dingen der Code die Dokumentation, das stimmt wirklich. Das darf man nie pauschalisieren, weil gerade Leute, die sich mit dem Thema nicht auskennen, das nutzt denen fast gar nichts. Aber wenn man sich damit auskennt, dann ist der Code auch Dokumentation."(SE5)

4.4.13 Code als Dokumentation

klarer Code ist die beste Dokumentation

Wie bereits im vorigen Abschnitt erwähnt, sind einige Entwickler der Ansicht, dass ein klarer Code die beste Dokumentation ist (SE1, SE2, SE5). Einer ging sogar so weit zu sagen

> „Strukturierte Software ist wichtiger als gute Dokumentation."(SE2)

Doch die Warnung kommt von einem anderen direkt hinterher:

> „Das darf man nie pauschalisieren, weil gerade Leute, die sich mit dem Thema nicht auskennen, das nutzt denen fast gar nichts."(SE5)

Der Code allein dient nur als Dokumentation für erfahrenere Entwickler, die sich mit dem Thema bereits auskennen.

Es mag also nicht für alle Entwickler stimmen, dass der Code die beste Dokumentation ist, doch strukturierter und klarer Code ist auf jeden Fall eine gute Grundlage für eine gute Dokumentation, die ja häufig aus dem Code generiert wird. Auf die Frage, welche

[61] Vergleiche Abschnitt 4.4.4
[62] Vergleiche Abschnitt 4.4.10

Elemente eine gute Dokumentation ausmachen[63], kam von vielen der Entwickler die Antwort „sprechende/selbsterklärende Namen" (SE1, SE3, SE4, SE5).

Denn „wenn man die Methoden gut schreibt, die Parameter richtig sauber macht – versteht man's einfach"(SE5) und die Dokumentation kann sich darauf beschränken, die wirklich relevanten Dinge zu erklären (SE1, SE3).

Zusätzlich zu den Benennungen ist auch die Typisierung der Schnittstelle eine Möglichkeit, den Code selbsterklärender zu gestalten. Dies ersetzt Dokumentation zwar nicht vollständig, aber erhöht die Verständlichkeit (SE6).

Der Vorteil des Codes als Dokumentation ist, dass der Code automatisch überprüft und korrigiert werden kann:

> „Das Schöne an Code ist, dass das alles der Compiler prüft und die IDE bietet einem schöne ‚refactoring features', die einem ermöglichen quasi über verschiedene Dateien hinweg, mit einer Operation Umbenennungen und so was konsistent zu machen."(SE3)

Wird nun aus diesem Code die Dokumentation generiert, so ist die gesamte Dokumentation weniger fehleranfällig (SE3).

verringerte Fehleranfälligkeit

4.4.14 Sonstige Aussagen

Ein weiterer Punkt, der den Umfang der Dokumentation verringern kann, ist ein gutes Interface. Ist dieses übersichtlich gestaltet, so wird weniger Dokumentation benötigt (SE5).

gutes Interface ist wichtig

Dies und die Punkte zu aussagekräftigem Code[64] ermöglichen, „dass es möglichst wenige Möglichkeiten gibt, dass das Zeug veraltet oder falsche Informationen liefert" und „dann ist das oft deutlich besser als wenn es einfach mehr Möglichkeiten gibt, dass etwas schief gehen kann"(SE3).

Einer der Entwickler erwähnte außerdem, dass zusätzliche Hilfen wie zum Beispiel Foren und Community Seiten, besonders bei Open Source Software, sehr beliebt und hilfreich sind (SE5).

[63] siehe hierzu Abschnitt 4.4.5
[64] Siehe Abschnitt 4.4.13

5 Diskussion

Im abschließenden Kapitel werden die Ergebnisse (Kapitel 5.1) und die Methode (Kapitel 5.2) diskutiert und Empfehlungen zur Erstellung hilfreicher API-Dokumentation abgeleitet (Kapitel 5.3).

Zum Schluss wird in Kapitel 5.4 ein Ausblick auf weitere Forschungsmöglichkeiten gegeben.

5.1 Diskussion der Ergebnisse

Es zeigt sich, dass die im Rahmen dieser Untersuchung befragten Entwickler bestimmte Erwartungen und Meinungen zu Software-Dokumentation haben, die Konsequenzen für die Gestaltung von API-Dokumentationen nach sich ziehen.

Schon deren Erwartungshaltung zeigt eine Tendenz. Da alle die Programmiersprache Java verwenden, erwarten sie die tabellarische Form eines Javadocs und einen kurzen Überblick über die API[65]. Daraus lässt sich schließen, dass es wichtig ist, die Zielgruppe der Dokumentation genau zu bestimmen und deren Bedürfnisse zu analysieren, denn von der Zielgruppe hängt die Erwartungshaltung ab. Ebenso auch einige andere Dinge, die im Folgenden beschrieben werden.

Ein Großteil der Entwickler betont, dass der Unterschied zwischen unerfahrenen und erfahrenen Entwicklern in der Regel groß sei, da die einen wesentlich mehr Informationen benötigen als die anderen. Wie auch von Jeong u. a. (2009) empfohlen, ist es deshalb wichtig, in Dokumentationen für die verschiedenen Nutzerzielgruppen jeweils angepasste Informationen und Navigationsmöglichkeiten anzubieten. Die Notwendigkeit einer Anpassung der Informationen an den Wissensstand der Benutzer beschreiben auch Novick u. Ward (2006).

Einher mit dieser an die Zielgruppe angepassten Information geht die Frage nach dem idealen Umfang einer API-Dokumentation. Alle interviewten Software-Entwickler haben dazu ihre eigenen Meinungen. Diese driften allerdings auseinander - manche

[65] Vergleiche Kapitel 4.4.1

haben sie gerne möglichst kurz, andere lieber so ausführlich wie nur möglich[66]. Die unterschiedlichen Ansichten lassen sich nicht auf den Hintergrund der Entwickler zurückführen, auch unter Entwicklern mit ähnlicher Berufserfahrung gibt es verschiedene Ansichten. Es scheint also allein eine Frage des persönlichen Geschmacks zu sein.

Daher ist es notwendig, einen Mittelweg zu finden. Daher kann es sinnvoll sein, den Umfang durch das Aufsplitten der Dokumentation in eine eigentliche API-Dokumentation und eine Referenzdokumentation variabel zu halten[67].

So können essentielle Informationen, die für alle Benutzer wichtig sind, wie Methoden- und Klassenbeschreibungen, in einer codenahen API-Dokumentation präsentiert werden und weiterführende, auf unterschiedliche Zielgruppen angepasste Informationen in einer Referenzdokumentation. Durch diese ausführlichere Referenzdokumentation kann vermieden werden, dass die Dokumentation durch zu starke Kürzungen unzureichend wird, was laut Robillard u. DeLine (2011) eines der größten Probleme der API-Dokumentationen ist. Außerdem bestätigen Ko u. Riche (2011) die Relevanz des konzeptionellen Wissens, welches nur in einer solchen Referenzdokumentation dargestellt werden kann. Denn in dieser sind Ausschweifungen in Themengebiete möglich, die nicht unbedingt für alle Benutzer der API relevant sind, da ihre Notwendigkeit unter anderem vom bisherigen Wissensstand abhängt.

Ein weiterer wichtiger Aspekt, den man im Auge haben sollte, um unzureichende Dokumentation zu vermeiden, ist die Aktualität der Dokumentation[68]. Veraltete Dokumentation führt zu Fehlinformationen, die das Vertrauen der Benutzer in die Dokumentation schwächen. Dieser Wunsch nach Fehlerfreiheit wurde von Novick u. Ward (2006) bestätigt.

Der Wunsch nach einer möglichst aktuellen Dokumentation ist möglicherweise auch einer der Gründe, weshalb die meisten Entwickler automatisch generierte Dokumentationen lieber nutzen als statische Dokumente.

Jeong u. a. (2009), Robillard u. DeLine (2011) und Novick u. Ward (2006) führen an, dass die Benutzer einer Dokumentation sich wünschen, dass diese klar und einheitlich strukturiert, gut zu navigieren sowie leicht zu durchsuchen sein sollte. Dies kann durch diese Untersuchung bestätigt werden. Vier der befragten Software-Entwickler gaben eine einheitliche Formatierung und Terminologie sowie eine gute Übersicht als wichtige Eigenschaften einer guten Dokumentation an[69]. Auch die Suchfunktion wurde von zwei Entwicklern als ein hilfreiches Element einer guten Dokumentation erwähnt[70].

Generell werden in dieser Studie einige hilfreiche Eigenschaften und Elemente angesprochen, die bereits in der Literatur genannt wurden, wie zum Beispiel die How-To-

[66] Vergleiche Kapitel 4.4.9
[67] Vergleiche hierzu Kapitel 4.4.10
[68] Vergleiche dazu Kapitel 4.4.11
[69] Vergleiche Kapitel 4.4.8
[70] Vergleiche Kapitel 4.4.5

Anleitungen oder die Möglichkeit, die API mit realen Daten zu testen. Beides Aspekte, die auch in Jeong u. a. (2009) behandelt werden.

Die Hauptaufgabe bei der Untersuchung im Rahmen dieser Arbeit bestand darin, die Frage nach den für die Software-Entwickler wichtigen Informationen, die in der API-Dokumentation und der zugehörigen Referenzdokumentation enthalten sein sollten, zu klären[71].

Die Ergebnisse dazu, welche in Kapitel 4.4.3 beschrieben sind, decken sich dabei mit den von Maalej u. Robillard (2013) gefundenen Wissensarten, was dafür spricht, dass in vielen Referenzdokumentationen bereits ein Großteil der Informationen zumindest angesprochen werden.

Dass einige Entwickler trotzdem fehlende Informationen bemängeln[72], könnte also nicht nur darin begründet sein, dass diese wirklich nicht vorhanden sind, sondern auch darin, dass sie nicht gut genug ausgewiesen sind und somit nicht gefunden werden können.

Das Gegenteil von fehlender Information ist überflüssige. Maalej u. Robillard (2013) fanden in ihrer Untersuchung heraus, dass Nullinformationen am zweithäufigsten API-Dokumentationen zu finden sind.

Dass dem so ist, wurde auch von vier der im Rahmen dieser Arbeit befragten Entwickler erwähnt[73].

Vollständigkeit, Aktualität und inhaltliche Korrektheit sind wichtige Qualitätskriterien wie es zum Beispiel auch Steinhardt u. a. (2015) herausgefunden haben und diese Studie es ebenfalls bestätigt. Es stellte sich aber außerdem heraus, dass es überaus wichtig ist, dass die Dokumentation sich auf die wirklich relevanten Dinge beschränkt. Denn häufig scheint das Streben nach Vollständigkeit dazu zu führen, dass die Dokumentationen von Nullinformationen überflutet werden[74].

Die Studien von Robillard u. DeLine (2011) und Wang u. a. (2012) beschreiben die hohe Relevanz von Code-Beispielen in API-Dokumentation. Dies belegen auch die Ergebnisse dieser Arbeit. Für alle Teilnehmer der Studie waren Code-Beispiele von großer Bedeutung und sie gaben an, dass es die Teile der Dokumentation sind, die sie mit am Häufigsten zurate ziehen.

„Ich schreibe Code, daher suche ich Code”[Steinhardt u. a. (2015), S. 12] ist eine weitere der von Steinhardt u. a. (2015) postulierten Strategien von Software-Entwicklern beim Erlernen neuer APIs. Diese konnte duch die Befragung der Entwickler der 1&1 Internet SE bestätigt werden. Das Vorhandensein von Beispielen[75] machen für sie

[71] Vergleiche hierzu Kapitel 4.4.3
[72] Vergleiche Kapitel 4.4.7
[73] Vergleiche Kapitel 4.4.4
[74] Vergleiche Kapitel 4.4.2 Absatz 2 und Kapitel 3.1.6 zu Maalej u. Robillard (2013)
[75] siehe Kapitel 4.4.6

eine gute Dokumentation aus. Zudem zeigte die Untersuchung im Rahmen dieser Arbeit, dass die Software-Entwickler sich zusätzlich zu den Code-Beispielen eine interaktive Dokumentation wünschen, durch die sie eine weitere Möglichkeit erhalten, die API und ihre Funktion zu testen [76].

5.2 Diskussion der Methode

Dafür, dass die Ergebnisse dieser Studie möglicherweise nicht repräsentativ sein könnten, gibt es einige Gründe.

Zum einen hatte sie mit sechs befragten Software-Entwicklern nur wenige Teilnehmer. Alle stammen aus einem einzigen Betrieb und haben einen ähnlichen Hintergrund: Alle programmieren mithilfe von Java und alle sind Software-Entwickler.

Zum anderen erhält man durch Experteninterviews, wie die in dieser Arbeit, viele subjektive Einzelmeinungen, die nur bedingt generalisiert werden können. Es wird nur ein Ausschnitt der Realität abgebildet.

Das Auswertungsverfahren nach Mühlfeld ist ein weiterer Punkt, der die Ergebnisse beeinflusst. Denn in der ihm zugrunde liegenden qualitativen Inhaltsanalyse werden die Daten durch die auswertende Person interpretiert. Diese Person sucht die Themen für die Analyse aus und entscheidet, welche Aussagen der Befragten dafür relevant sind und interpretiert diese, um Hypothesen aufzustellen.

Allerdings wird durch die Befragung der Software-Entwickler ein weiterer Einblick in das Forschungsfeld der API-Dokumentation erlangt, durch den die Ergebnisse der vorhergehenden Untersuchung bekräftigt, weiter detailliert und ergänzt werden[77].

Die Experteninterviews sind zwar nicht im statistischen Sinne repräsentativ, doch die gewonnenen Aussagen der Experten können als Basis für Empfehlungen dienen, da sie eine Tendenz anzeigen.

[76] siehe Seite 38 und Seite 43
[77] Vergleiche Kapitel 4.4, Kapitel 5.1 und Kapitel 5.3.

5.3 Resultierende Empfehlungen zur Erstellung hilfreicher API-Dokumentation

Aus den in dieser Arbeit zusammengetragenen Ergebnissen lassen sich folgende Empfehlungen zur Erstellung hilfreicher API-Dokumentation ableiten:

- Es ist unerlässlich, Dokumentation aktuell zu halten. Schon eine veraltete Information weckt das Misstrauen der Entwickler in die gesamte Dokumentation.
- Die Dokumentation sollte an die Erwartungshaltung der Entwickler angepasst sein, sprich eine Kenntnis der Zielgruppe beim Erstellen der Dokumentation ist unerlässlich.
- Die API-Dokumentation muss gut strukturiert sein und sollte den Entwicklern durch Übersichten und eine gute Suche immer die für ihr aktuelles Problem relevanten Informationen anbieten. Es ist daher notwendig, den Blickwinkel des Benutzers einzunehmen, wenn man an der Dokumentation arbeitet.
- Auf keinen Fall fehlen sollte eine Übersicht über die API und deren Funktionen, um den Einstieg in die Arbeit mit der API und der zugehörigen Dokumentation zu erleichtern und Zusammenhänge aufzuzeigen.
- Die wichtigsten Informationen sind für die Software-Entwickler diejenigen zu Klassen, Methoden und Parametern. Wichtig ist nicht nur die Frage „Was machen sie?”, sondern vor allem die Frage „Warum machen sie das?” und „Wie darf ich sie verwenden?”.
- Codebeispiele haben ebenfalls einen hohen Stellenwert und sollten deshalb in einer guten Dokumentation nicht fehlen. Dabei muss darauf geachtet werden, dass diese auch funktionieren, auf dem aktuellsten Stand sind und problemlos in den eigenen Code übernommen werden können.
- Unnötige Information ist einer der größten Störfaktoren. Beschreibungen von Dingen, die selbsterklärend sind, wie zum Beispiel die simple Paraphrasierung von Methodennamen, sind zu vermeiden.
- Was als unnötige Information deklariert wird, hängt vom Kenntnisstand des Benutzers ab. Es ist darum empfehlenswert, allgemeinere Informationen in Referenzdokumentation auszulagern, die von den Benutzern nur im Bedarfsfall aufgerufen werden können.

Nun gilt es, diese Resultate durch weitere Nachforschungen zu bestätigen und gegebenenfalls zu erweitern.

5.4 Ausblick auf weiteren Forschungsbedarf

Die Ergebnisse und die sich daraus ergebenden Empfehlungen zeigen auch Ansätze für weitere Nachforschungen auf.

Zum einen gilt es durch statistische Folgeuntersuchungen (zum Beispiel eine Befragung mittels Fragebögen), herauszufinden inwiefern die Untersuchung repräsentativ und generalisierbar ist, denn wie in Kapitel 5.2 beschrieben, kann dies nicht garantiert werden.

Zum anderen sollte überprüft werden, ob die Entwickler auch wirklich so handeln, wie sie es beschreiben, was durch Beobachtungsstudien möglich ist.

Außerdem empfehlenswert wäre eine Untersuchung von bereits vorhandener API-Dokumentation, um zu erfahren, welche der angesprochenen wichtigen Inhalte und Eigenschaften bereits häufig umgesetzt sind und welche nicht. So könnte die Forschung die häufig genannten Schwachstellen noch spezifischer identifizieren und weitere Maßnahmen zur Beseitigung dieser erarbeiten.

Literatur

[Bloch 2006] BLOCH, Joshua: How to Design a Good API and Why It Matters. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*. New York, NY, USA : ACM, 2006 (OOPSLA '06), 506–507

[Clements 2003] CLEMENTS, P.: *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2003 (SEI series in software engineering). – ISBN 9780201703726

[Dagenais u. Robillard 2010] DAGENAIS, Barthélémy ; ROBILLARD, Martin P.: Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors. In: *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA : ACM, 2010 (FSE '10). – ISBN 978–1–60558–791–2, S. 127–136

[DATACOM Buchverlag 2015] DATACOM BUCHVERLAG: *ITWissen - Das große Online-Lexikon für Informationstechnologie*. itwissen. Version: 2015. – http://www.itwissen.info/definition/lexikon/application-programming-interface-API-Programmierschnittstelle.html [Abgerufen am 07.12.2015]

[Diekmann 2007] DIEKMANN, A.: *Empirische Sozialforschung: Grundlagen, Methoden, Anwendungen*. Rowohlt-Taschenbuch-Verlag, 2007 (Rowohlts Enzyklopädie). – ISBN 9783499556784

[Fielding 2000] FIELDING, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Diss., 2000

[Flick 2007] FLICK, Uwe: *Qualitative Sozialforschung: eine Einführung*. Rowohlt-Taschenbuch-Verlag, 2007 (Rowohlts Enzyklopädie). – ISBN 9783499556944

[Gründerszene Magazin 2009] GRÜNDERSZENE MAGAZIN: *Web APIs - Ein nicht-technischer Erklärungsversuch*. November 2009. – http://www.gruenderszene.de/allgemein/web-apis-ein-nicht-technischer-erklarungsversuch [Abgerufen am 10.03.2016]

[Gruenbaum 2010] GRUENBAUM, Peter: A Coder's Guide to Writing API Documentation. In: *MSDN Magazine* 25 (2010), S. 70–76

[Hery-Moßmann 2015] HERY-MOSSMANN, Nicole: *Was ist API? Einfach erklärt*. Juni 2015. – http://praxistipps.chip.de/was-ist-api-einfach-erklaert_41370 [Abgerufen am 10.03.2016]

[Hoffmann 2016] HOFFMANN, Mark: *Gründerszene Lexikon - Application-Programming-Interface*. 2016. – http://www.gruenderszene.de/lexikon/begriffe/application-programming-interface-api [Abgerufen am 07.01.2016]

[Jeong u. a. 2009] JEONG, Sae Y. ; XIE, Yingyu ; BEATON, Jack ; MYERS, Brad A. ; STYLOS, Jeff ; EHRET, Ralf ; KARSTENS, Jan ; EFEOGLU, Arkin ; BUSSE, Daniela K.: Improving Documentation for eSOA APIs through User Studies. In: *Lecture Notes in Computer Science* 5435 (2009), S. 86–105

[Ko u. Riche 2011] KO, Andrew J. ; RICHE, Yann: The role of conceptual knowledge in API usability. In: COSTAGLIOLA, Gennaro (Hrsg.) ; KO, Andrew J. (Hrsg.) ; CYPHER, Allen (Hrsg.) ; NICHOLS, Jeffrey (Hrsg.) ; SCAFFIDI, Christopher (Hrsg.) ; KELLEHER, Caitlin (Hrsg.) ; MYERS, Brad A. (Hrsg.): *VL/HCC*, IEEE, 2011. – ISBN 978–1–4577–1246–3, 173-176

[Konrad 2010] *Kapitel* Lautes Denken. In: KONRAD, Klaus: *Handbuch Qualitative Forschung in der Psychologie*. Wiesbaden : VS Verlag für Sozialwissenschaften, 2010. – ISBN 978–3–531–92052–8, 476–490

[Lamnek 1995] LAMNEK, Siegfried: *Qualitative Sozialforschung*. Beltz, 1995 (Qualitative Sozialforschung Bd. 2). – ISBN 9783621271769

[Maalej u. Robillard 2013] MAALEJ, Walid ; ROBILLARD, Martin P.: Patterns of Knowledge in API Reference Documentation. In: *IEEE Transactions on Software Engineering* 39 (2013), Nr. 9, S. 1264–1282

[Mayer 2009] MAYER, Horst O.: *Interview und schriftliche Befragung: Entwicklung, Durchführung und Auswertung*. Oldenbourg, 2009. – ISBN 9783486590708

[Mayring et al. 2008] MAYRING ET AL. ; MAYRING, Philipp [. (Hrsg.): *Die Praxis der qualitativen Inhaltsanalyse*. 2., neu ausgestattete Aufl. Weinheim [u.a.] : Beltz, 2008 (Beltz Pädagogik). – ISBN 978–3–407–25502–0

[Merton u. Kendall 1979] In: MERTON, Robert K. ; KENDALL, Patricia L.: *Das fokussierte Interview*. Klett-Cotta, 1979. – ISBN 3–12–923591–4, S. 171–204

[Meuser u. Nagel 1991] MEUSER, Michael ; NAGEL, Ulrike: ExpertInneninterviews - vielfach erprobt, wenig bedacht: ein Beitrag zur qualitativen Methodendiskussion. In: DETLEF GARZ, Klaus K. (Hrsg.): *Qualitativ-empirische Sozialforschung: Konzepte, Methoden, Analysen.*, Opladen: Westdt. Verl., 1991, S. 441–471

[Mühlfeld u. a. 1981] MÜHLFELD, Claus ; WINDOLF, Paul ; LAMPERT, Norbert ; KRÜGER, Heidi: Auswertungsprobleme offener Interviews. In: *Soziale Welt* 32 (1981), Nr. 3, 325-352. http://www.jstor.org/stable/40877322. – ISSN 00386073

[Nijim u. Pagano 2014] NIJIM, Sharif ; PAGANO, Brian: *APIs For Dummies*. John Wiley & Sons, Inc., 2014

[Novick u. Ward 2006] NOVICK, David G. ; WARD, Karen: What Users Say They Want in Documentation. In: *Proceedings of the 24th Annual ACM International Conference on Design of Communication*. New York, NY, USA : ACM, 2006 (SIGDOC '06). – ISBN 1–59593–523–1, 84–91

[Oey 2007] OEY, K.J.: *Moderne Softwaredokumentation: der Weg zu effizienter Entwicklerdokumentation*. VDM-Verlag Dr. Müller, 2007. – ISBN 9783836421768

[parson 2013] PARSON: *Entwicklerdokumentation: das notwendige Übel?* Juni 2013. – https://www.parson-europe.com/de/wissensartikel/202-entwicklerdokumentation-wissen.html [Abgerufen am 29.02.2016]

[Robillard u. DeLine 2011] ROBILLARD, Martin P. ; DELINE, Robert: A field study of API learning obstacles. In: *Empirical Software Engineering* 16 (2011), Nr. 6, S. 703–732

[Savage 2013] SAVAGE, Neil: *Open your data to the world*. April 2013. – http://www.computerworld.com/article/2496465/enterprise-applications/open-your-data-to-the-world.html [Abgerufen am 10.03.2016]

[Scheuch 1973] In: SCHEUCH, Erwin K.: *Das Interview in der Sozialforschung*. Dt. Taschenbuch Verl., 1973. – ISBN 3–423–04236–2, S. 66–190

[Schmidt 2009] SCHMIDT, Christiane: Analyse von Leitfadeninterviews. In: UWE FLICK, Ines S. Ernst von Kardorff K. Ernst von Kardorff (Hrsg.): *Qualitative Forschung - Ein Handbuch*, rowohlts enzyklopädie im Rowohlt Taschenbuch Verlag, 2009, S. 447–456

[Siedersleben 2004] SIEDERSLEBEN, Johannes: *Moderne Softwarearchitektur: umsichtig planen, robust bauen mit Quasar*. dpunkt-Verlag, 2004. – ISBN 9783898642927

[Spöhring 1989] SPÖHRING, Walter: *Qualitative Sozialforschung*. Stuttgart : Teubner, 1989 (Teubner-Studienskripten : 133 : Studienskripten zur Soziologie : 133 : Studienskripten zur Soziologie). – ISBN 3–519–00133–0

[Steinhardt u. a. 2015] STEINHARDT, Stephanie ; SCHUBERT, Andreas ; MENG, Michael: Software-Entwicklerdokumentation: Was wollen Entwickler wirklich? In: *Vortragsfolien zur tekom Jahrestagung 2015*, 2015. – http://tagungen.tekom.de/fileadmin/tx_doccon/slides/1119_Software_Entwicklerdokumentation_Was_wollen_Entwickler_wirklich_.pdf [Abgerufen am 18.12.2015]

[TechTarget 2015] TECHTARGET: *Definition: Programmierschnittstelle (API)*. April 2015. – http://www.searchenterprisesoftware.de/definition/Programmierschnittstelle-API [Abgerufen am 10.03.2016]

[Wang u. a. 2012] WANG, Lijie ; ZOU, Yanzhen ; FANG, Lu ; XIE, Bing ; YANG, Fuqing: An Exploratory Study of API Usage Examples on the Web. In: LEUNG, Karl R. P. H. (Hrsg.) ; MUENCHAISRI, Pornsiri (Hrsg.): *APSEC*, IEEE, 2012. – ISBN 978–0–7695–4922–4, 396-405

[Watson u. a. 2013] WATSON, Robert ; STAMNES, Mark ; JEANNOT-SCHROEDER, Jacob ; SPYRIDAKIS, Jan H.: API Documentation and Software Community Values:

A Survey of Open-source API Documentation. In: *Proceedings of the 31st ACM International Conference on Design of Communication*. New York, NY, USA : ACM, 2013 (SIGDOC '13). – ISBN 978–1–4503–2131–0, 165–174

[Witzel 2000] WITZEL, Andreas: The Problem-centered Interview. In: *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 1 (2000), Nr. 1. http://www.qualitative-research.net/index.php/fqs/article/view/1132. – ISSN 1438–5627

[Zörner 2015] ZÖRNER, Stefan: *Softwarearchitekturen dokumentieren und kommunizieren: Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten*. Carl Hanser Verlag GmbH & Company KG, 2015. – ISBN 9783446444423

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig angefertigt, alle dem Wortlaut oder Sinn nach entnommenen Inhalte anderer Werke an den entsprechenden Stellen unter Angabe der Quellen kenntlich gemacht und keine weiteren Hilfsmittel verwendet zu haben.

Julia Dusold
Karlsruhe, den 15. April 2016