Zur Benutzbarkeit und Verwendung von API-Dokumentationen

Rolf Huesmann
Hochschule Darmstadt
Darmstadt, Deutschland
rolf.huesmann@h-da.de

Alexander Zeier
Hochschule Darmstadt
Darmstadt, Deutschland
alexander.zeier@h-da.de

Andreas Heinemann
Hochschule Darmstadt
Darmstadt, Deutschland
andreas.heinemann@h-da.de

Alexander Wiesmaier
Hochschule Darmstadt
Darmstadt, Deutschland
alexander.wiesmaier@h-da.de

ZUSAMMENFASSUNG

Eine gute Dokumentation ist essenziell für eine gute Benutzbarkeit von (Sicherheits-)APIs, d.h. insbesondere für die korrekte Verwendung der APIs. Anforderungen an eine gute Dokumentation von APIs wurden in mehreren Arbeiten beschrieben, jedoch gibt es bislang keine technische Umsetzung (im folgenden Dokumentationssystem genannt), welche diese Anforderungen umsetzt. Die Anforderungen lassen sich unterteilen in Anforderungen an das Dokumentationssystem und Anforderungen an den Dokumentationsinhalt. Aus 13 identifizierten Anforderungen an ein Dokumentationssystem selbst wurden im Rahmen dieser Arbeit 9 in einen Prototypen umgesetzt und in einer Nutzerstudie mit 22 Probanden unter Verwendung einer kryptografischen API evaluiert. Es hat sich gezeigt, dass die Umsetzung der Anforderung *Schnelle Nutzung der API ermöglichen* zum einen wesentlich von der Qualität der eingepflegten Inhalte abhängt, zum anderen aber auch 5 weitere der betrachteten Anforderungen bzw. deren Umsetzungen subsumiert. Die zwei weiteren umgesetzten Anforderungen (*Klassische Referenz* und *Rückfragen und Kommentarfunktion*) wurden von den Probanden kaum oder nicht eingesetzt. Deren Nützlichkeit und Relevanz sollte in einer Langzeitstudie untersucht werden.

CCS CONCEPTS

• **Human-centered computing → Usability testing**; **User studies**; • **Security and privacy → Usability in security and privacy**.

KEYWORDS

API Dokumentation, Dokumentationssystem, Benutzbarkeit, Gebrauchstauglichkeit, Nutzerstudie, eUCRITE, Tink

1 EINLEITUNG UND MOTIVATION

Mehrere Studien [5, 10, 11, 14–16, 18, 19] haben die Bedeutung von Dokumentationen für die Benutzbarkeit einer API herausgestellt. Weitere Arbeiten [7–9, 12] haben gezeigt, was eine gute bzw. gut benutzbare Dokumentation ausmacht. Viele API-Dokumentationen erfüllen diese Anforderungen jedoch nicht oder nicht ausreichend [2], was zu einer fehlerhaften Verwendung der API führt und insbesondere bei Krypto-APIs die intendierten Sicherheitsziele einer Anwendung gefährdet. Bei einer unzureichenden Benutzbarkeit von API-Dokumentationen wird durch den Entwickler häufig auf andere Quellen wie beispielsweise die *Question and Answer* (Q&A) Plattform StackOverflow[1] zurückgegriffen. Jedoch sind hier die Codebeispiele oft nicht fehlerfrei [4].

Softwareentwickler sind Schlüsselpersonen, da sie Code produzieren, welcher möglicherweise auf Millionen von Geräten ausgeführt wird. Dadurch werden sie zu Multiplikatoren von Sicherheitsrisiken. Gerade im Bereich der kryptografischen APIs kann ein Programmierfehler eine große Tragweite haben [17]. Deswegen sollte besonders auf die Verständlichkeit und Benutzbarkeit von kryptografischen API-Dokumentationen für Softwareentwickler geachtet werden.

In dieser Arbeit wird ein Dokumentationssystem für kryptografische APIs anhand existierender Anforderungen entwickelt, prototypisch implementiert und in einer Nutzerstudie hinsichtlich der Benutzbarkeit evaluiert. Die Anforderungen wurden durch eine Literaturrecherche [6] ermittelt und anhand der Abhängigkeiten zueinander geordnet (siehe Abschnitt 2). Anschließend wird der entwickelte Prototyp (Abschnitt 3) und die durchgeführte Nutzerstudie (Abschnitt 4) sowie deren Ergebnisse (Abschnitt 5) beschrieben. In Abschnitt 6 werden Limitierungen dieser Arbeit aufgezeigt. Abschließend gibt es eine Zusammenfassung und einen Ausblick (Abschnitt 7). Folgender Abschnitt 2 geht auf die Anforderungsanalyse ein.

2 ANFORDERUNGSANALYSE ANHAND VERWANDTER ARBEITEN

In einer explorativen Studie haben die Autoren dieses Beitrags [7] 10 Kriterien identifiziert, die eine gut benutzbare API-Dokumentation erfüllen sollte. Dazu wurden in mehreren Fokusgruppen mit insgesamt 26 Teilnehmern die Vor- und Nachteile bestehender Dokumentationsarten und -systeme von APIs erarbeitet. Die Autoren Meng et al. [8] haben mit 17 Entwicklern jeweils ein Interview mit einem Fragenkatalog bestehend aus 45 Fragen durchgeführt. Aus den Antworten haben sie 13 Eigenschaften, welche im Bezug zu einer guten API-Dokumentation erwähnt wurden, extrahieren können. Ein Jahr später haben Meng et al. [9] die API der shipcloud GmbH[2] auf ihre

MuC'20 Workshops, Magdeburg, Deutschland

https://doi.org/10.18420/muc2020-ws119-002

[1] https://stackoverflow.com, Abgerufen am 7.10.2019
[2] https://www.shipcloud.io/ Abgerufen am: 07.10.2019

Benutzbarkeit untersucht. 11 Teilnehmer haben dazu eine Programmieraufgabe mit der API gelöst und wurden anschließend mit einem Fragebogen über die Qualität der API-Dokumentation befragt. Dabei wurden 11 Anforderungen diskutiert. Robillard [12] haben 2011 in ihrer Arbeit 6 Anforderungen an API-Dokumentationen beschrieben. Dazu haben sie mit 440 Microsoft-Entwicklern über eine explorative qualitative Studie und eine Umfrage die Benutzbarkeit einer API-Dokumentation untersucht.

Die in diesen Arbeiten genannten Anforderungen bilden die Ausgangslange für diese Arbeit. Sie sind in Tabelle 1 zusammengefasst und nach Abhängigkeiten zueinander gruppiert. Beispielsweise trägt die Umsetzung der Anforderung 2 *Gute Struktur* zur Umsetzung der Anforderung 1 *Schnelle Nutzung der API ermöglichen* bei. Diese Abhängigkeiten sind als gerichteter Graph zu verstehen. Anforderungen, die nicht selbsterklärend sind, werden im Folgenden kurz erläutert:

1 Schnelle Nutzung der API ermöglichen: Dies sollte zum Beispiel durch einen Schnelleinstieg oder einer Tryout-Funktionalität[3] möglich sein.

5 Verschiedene Niveaus: Je nach Entwicklertyp und Wissensstand ist für ein Entwickler ein anderes Informationsangebot hilfreich. Ein Entwickler, der bspw. nur wenig Erfahrung mit kryptografischen APIs hat, braucht eine andere Detailtiefe und Herangehensweise als ein Entwickler, der täglich mit kryptografischen APIs arbeitet.

6 Inhalt gemäß der API-Funktionalität organisieren: Die API-Dokumentation sollte nach Kategorien strukturiert sein, welche die Funktionalität oder den Inhaltsbereich der API widerspiegeln. Stößt ein Entwickler auf ein Problem, sollte er ungefähr wissen, in welchem Bereich oder welcher Methode dieses Problem verortet ist. Die Dokumentation sollte so strukturiert sein, dass der Entwickler diesen Ort in der Dokumentation gut finden kann.

7 Selektiver Zugriff auf den Code aktivieren: Die Beispiele sollten über eine geeignete Designstrategie klar vom Text zu unterscheiden sein, wodurch es für Entwickler mit einer opportunistischen Arbeitsweise nach Clarke [3] einfacher wird, direkt zu relevanten Beispielen zu springen.

8 Konzeptionelle Informationen mit verwandten Aufgaben integrieren: Konzeptionelle Informationen verdeutlichen die Funktionsweise sowie Struktur der API und sollten zusammen mit Aufgaben, geeigneten Beispielen oder Nutzungsszenarien präsentiert werden.

14 Klassische Referenz: In der jede Klasse, Methode und Funktion mit allen Parametern beschrieben wird.

23 Wichtige Informationen redundant bereitstellen: Entwickler mit einer opportunistischen Arbeitsweise sind beim Erlernen einer neuen API stärker auf Beispiele fixiert. Das bringt das Risiko mit sich, dass sie Abschnitte in der API-Dokumentation überspringen, in denen kritische konzeptionelle Informationen präsentiert werden. Dies erfordert einen Ansatz, der diese Inhalte redundant darstellt, z.B. in dem beschreibenden Dokumentationstext und zusätzlich in dem Beispiel als Codekommentar.

24 Aufzeigen von Alternativen: Die API-Dokumentaion sollte alternative Methoden, Funktionen, Klassen und Parameter aufzeigen, welche beispielsweise ähnliche Funktionalitäten bieten und somit ein Problem ggfs. effizienter lösen.

Eine Dokumentation besteht aus zwei wesentlichen Faktoren: 1.) Aus dem Dokumentationssystem, der Plattform oder dem Framework, mit dem die Dokumentation erstellt wird. 2.) Aus den Informationen, die der Dokumentationsersteller in dem System pflegt. Alle oben genannten Anforderungen kann das Dokumentationssystem alleine nicht erfüllen. Ein Teil davon hängt von der redaktionellen Pflege durch den Dokumentationsersteller, also dem Inhalt, ab (in Tabelle 1 durch ein ○ markiert).

Bei genauerer Betrachtung der Abhängigkeiten von Anforderung 1 *Schnelle Nutzung der API ermöglichen* fällt auf, dass 8 der 11 Anforderungen von der inhaltlichen Qualität der eingepflegten Informationen abhängig sind. Je nach Anwendungsfall sind einige dieser Anforderungen ggfs. nicht relevant. So verfügt beispielsweise nicht jede API über verschiedene Versionen oder über ähnliche, alternative Funktionsaufrufe oder Klassen.

Die inhaltlichen Anforderungen, an die sich ein Dokumentationsersteller halten sollte, um eine optimierte Dokumentation zu erstellen, sind in Form einer Checkliste[4] aufbereitet worden. Mit Hilfe dieser Checkliste wurden die Inhalte der im Prototyp verwendeten eUCRITE-API-Dokumentation erstellt.

Aus den Anforderungen an das Dokumentationssystem (in Tabelle 1 durch ein • markiert) wurde ein Konzept[4] für eine webbasierte Realisierung erstellt. Anhand dessen wurde der im folgenden Abschnitt beschriebene Prototyp umgesetzt.

3 PROTOTYP

Die Anforderungen an das Dokumentationssystem aus Abschnitt 2 wurden weitgehend prototypisch umgesetzt. Als zu dokumentierende API wurde die von Zeier et al. [18] entwickelte eUCRITE-API gewählt, da diese bereits gute Ergebnisse zur Benutzbarkeit hervorgebracht hat. Dadurch sollte weitestgehend ausgeschlossen werden, dass negative Ergebnisse dieser Studie auf die API anstatt auf das Dokumentationssystem zurückzuführen sind. Die eUCRITE-API richtet sich an App-Entwickler, die keine oder nur wenige kryptografische Kenntnisse besitzen. Der aktuelle Stand der API unterstützt kryptografische Operatoren zur Verschlüsselung und Signierung von Daten. Über Templates können Entwickler die aktuellen Verschlüsselungsparameter oder Algorithmen wählen, welche in Zukunft regelmäßig durch automatische Updates aktualisiert werden sollen. Nachfolgender Abschnitt geht auf Anforderungen ein, die nicht oder nicht vollständig umgesetzt werden konnten.

Anforderung 19 der *IDE Integration* wurde nicht realisiert, da die Probanden bewusst eine separate Online-Dokumentation nutzen sollten. Die meisten modernen IDEs bieten eine Javadoc-Integration[5].

[3] Beispiel für eine Tryout-Funktionalität im Web: Der Besucher kann Code auf der Webseite *„compilieren"* lassen und sieht sofort ob der Code fehlerfrei oder fehlerhaft ist.

[4] Die Checkliste und das Konzept ist in einer Langversion der Arbeit zu finden: https://arxiv.org/abs/2007.04983

[5] Javadoc: http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html Abgerufen am 08.09.2019

#	**Anforderung**	Abhängig von	[7]	[8]	[9]	[12]
1	Schnelle Nutzung der API ermöglichen	2, 8			○●	
2	Gute Struktur	3, 4, 5, 6, 7	○●		○●	○●
3	Verlinkungen zur Referenz / Signalisierte Text-zu-Code-Verbindungen		●		●	
4	Erweiterte Suche			●	●	
5	Verschiedene Niveaus		○			
6	Inhalt gemäß der API-Funktionalität organisieren				○	
7	Selektiver Zugriff auf den Code aktivieren				●	
8	Konzeptionelle Informationen mit verwandten Aufgaben integrieren	9, 11			○	○
9	Viele Beispiele	10	○	○	○	○
10	Negativbeispiele		○			
11	Beinhaltet die Fehlerbehandlung			○		○
12	Vermittelt (technisches) Hintergrundwissen	13	○	○	○	○
13	Explizite Dokumentation über die Performance					○
14	Klassische Referenz		○●	○●		
15	Vollständigkeit	16, 17, 18		○●		
16	Aktualität			○		
17	Transparente Versionierung der API			○●		
18	Änderungshistorie der Dokumentation			●		
19	IDE Integration	20	○●	○●		
20	Sourcecode		○●			
21	Rückfragen und Kommentarfunktion		●	●		
22	Suchmaschinenfreundlich			●		
23	Wichtige Informationen redundant bereitstellen				○	
24	Aufzeigen von Alternativen			○		

Tabelle 1: Übersicht der Anforderungen an den Dokumentationsinhalt (○) und das Dokumentationssystem (●) einer API. Gruppiert nach Abhängigkeiten zueinander.

Durch das Einfügen von Links auf unserem Dokumentationssystem in das Javadoc, kann eine erste IDE-Integration unseres Dokumentationssystems erreicht werden. Anforderung 20 wurde nicht betrachtet, da der Source-Code der eUCRITE-API in der IDE des Entwicklers angezeigt wurde. Die *Suchmaschinenfreundlichkeit* (Anforderung 22) wurde in dieser Arbeit außen vorgelassen, da die Erreichbarkeit der eUCRITE-API-Dokumentation in Suchmaschinen für diese Studie nicht gegeben ist, da die API bisher nicht öffentlich verfügbar gemacht wurde. Da der Prototyp auf dem Wordpress[6] Content Management System basiert, wäre ein Auffinden durch Suchmaschinen jedoch leicht realisierbar, da hier Search Engine Optimization Plugins einfach integriert werden können. Anforderung 17 wurde durch einen Versions-Schalter im Footer der Seite angedeutet. Dieser wurde allerdings nicht implementiert, da die eUCRITE-API bisher nicht über unterschiedliche Versionen verfügt. Aus demselben Grund wurde die *Änderungshistorie der Dokumentation* (Anforderung 18) nicht realisiert. Trotz diesen Restriktionen haben wir uns für die eUCRITE-API entschieden, da zum Zeitpunkt des Studiendesings laut TIOBE Index[7] Java die meistbenutzte Programmiersprache war und uns bisher keine andere Java-Api bekannt ist, welche bereits auf Benutzbarkeit überprüft wurde.

Wenden wir uns nun den Anforderungen zu, die durch unsere Implementierung erfüllt werden.

[6]Wordpress: https://de.wordpress.org Abgerufen am 16.12.2019
[7]TIOBE Index: https://www.tiobe.com/tiobe-index/ Abgerufen am: 01.08.2019

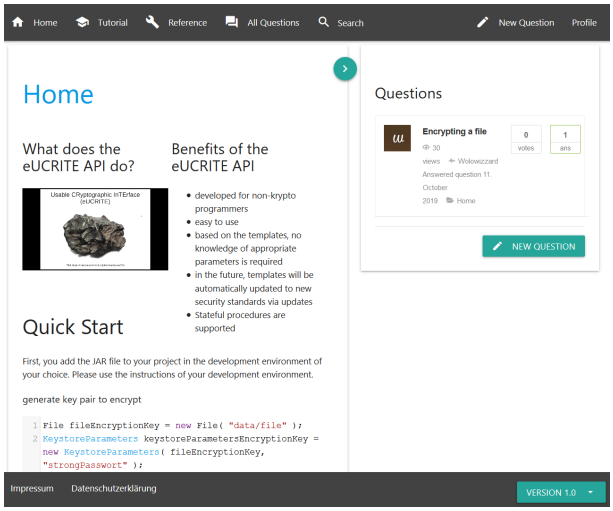

Abbildung 1: Startseite der eUCRITE-API-Dokumentation

Abbildung 1 zeigt die Startseite der prototypischen Dokumentation. Der Fragen-Bereich (Anforderung 21) ist auf der rechten Seite zu sehen. Dieser zeigt die Fragen an, die für die aktuell angezeigte Seite relevant sind. Der Besucher kann so direkt thematisch passend zum Inhalt der Dokumentation eine Frage stellen, Antworten

lesen und verfassen. Anforderung 2 *Gute Struktur* wird durch ein auf das Wesentliche reduziertes und mit Icons optisch aufbereitetes Header-Menü realisiert. In ihm ist die Suche integriert, welche auf der Trefferseite verfeinert werden kann und so Anforderung 4 entspricht. Unter dem Menü-Punkt „Reference" ist eine *klassische Referenz* (Anforderung 14) angegliedert. Diese könnte zukünftig automatisiert von dem Dokumentationssystem aus dem Javadoc der API generiert und durch den Dokumentationsersteller mit wichtigen Informationen verfeinert werden. Anforderung 3 wurde durch die Verlinkung von Klassen-Namen und -Aufrufen in Texten und Code-Beispielen realisiert. Die Besucher gelangen durch das Klicken auf einen Klassen-Aufruf in Code-Beispielen in die jeweilige Klassen-Beschreibung der API-Referenz. Code-Beispiele werden mit Syntaxhervorhebung, Zeilennummern und Einrückungen optisch von Beschreibungen hervorgehoben. Damit wurde Anforderung 7 umgesetzt.

Die inhaltlichen Anforderungen (in Tabelle 1 durch ein ∘ markiert) wurden vom Autor der eUCRITE-Dokumentation anhand der Checkliste umgesetzt.

4 STUDIENDESIGN UND DURCHFÜHRUNG

Ziel dieser Studie war es, die Benutzbarkeit anhand der in der Literatur gefundenen Anforderungen (siehe Abschnitt 2) der prototypischen Dokumentation in Kombination mit der eUCRITE-API zu evaluieren. Zur Vergleichbarkeit des Usability Scores nach Acar et al. [1] wurde zusätzlich die Tink-API[8] von Google untersucht. Die Tink-API ist eine plattformübergreifende Open-Source-Bibliothek, welche kryptografische APIs zur Verfügung stellt. Sie wurde unter dem Aspekt ausgewählt, dass sie gleichermaßen den Anspruch hat gut benutzbar zu sein und sich so für den Usability Score Vergleich eignet. In dieser Studie spielt die Tink-API eine untergeordnete Rolle, da ihre Dokumentation auf Github realisiert und ein expliziter Vergleich mit unserem Dokumentationssysten Unschärfen beinhaltet (siehe Abschnitt 6). Für diese Studie sollten Probanden (Entwickler) einen fiktiven Chat-Client programmieren. Dazu bekamen sie zunächst ein Code-Grundgerüst zur Verfügung gestellt, in dem sie die Funktionen zur Generierung des Schlüsselmaterials, der Ver- und Entschlüsselung sowie der Erzeugung und Verifizierung von digitalen Signaturen implementieren sollten. Bei Unklarheiten wurden sie aufgefordert, sich in der entsprechenden Online-Dokumentation zu informieren. Dabei wurden sie über ein Eye-Tracking-System[9] beobachtet. Zum Vergleich sollte die gleiche Aufgabe von den Teilnehmern ebenfalls mit der Tink-API durchgeführt werden. Die Reihenfolge der APIs wurde anhand der Teilnehmernummer alternierend festgelegt. In Pre-Tests wurde eine Zeit von ca. 45 Minuten pro API ermittelt. Sollte der Teilnehmer für die Lösung der ersten Aufgabe länger als eine Stunde benötigt haben, so ist die zweite Aufgabe entfallen.

Die Aufgabenstellung und Fragen wurden in englischer Sprache gestellt. In einem Fragebogen wurden zu Beginn einige Informationen über die Teilnehmer abgefragt und später eine Bewertung der jeweiligen APIs erfasst. Hierzu wurden neben den von uns formulierten Fragen der von Acar et al. [1] beschriebene API Usability

[8]Tink-API: https://github.com/google/tink Abgerufen am 16.12.2019
[9]Tobii Pro Studio https://www.tobiipro.com/de/produkte/tobii-pro-studio/ Abgerufen am 16.12.2019

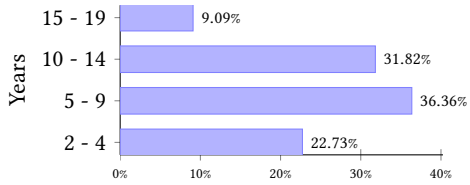

Abbildung 2: Programmiererfahrung der Probanden

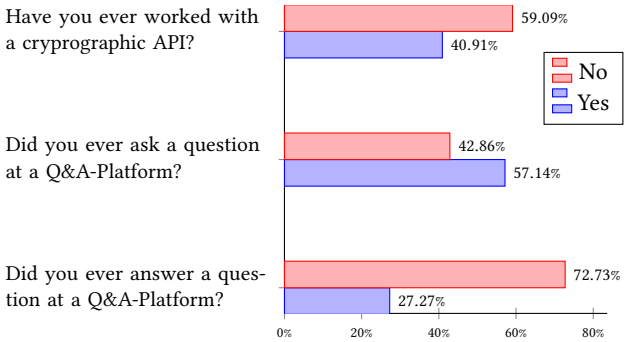

Abbildung 3: Ergebnisse der Ja/Nein-Fragen.

Score verwendet. Die genau gestellten Fragen und ein detaillierter Ablauf sind in einer Langversion[10] der Arbeit zu finden. Tabelle 2 zeigt die Zuordnung der zu untersuchenden Anforderungen zu den im Rahmen der Studie eingesetzten Methoden (die konkreten Q&A Fragen sowie Aussagen 1–5 sind zusammen mit den Ergebnissen in den Abbildungen 3 und 4 zu finden). Die Aussagen wurden von den Probanden in der Rohrmann-Skala [13] bewertet. Die Teilnahme erfolgte freiwillig und unter den geltenden datenschutzrechtlichen Gegebenheiten. Als Aufwandsentschädigung bekamen die Teilnehmer im Anschluss einen 20 Euro Amazon-Gutschein ausgehändigt.

5 ERGEBNISSE

Insgesamt haben 22 Probanden an der Studie teilgenommen. 6 davon aus der Industrie. Bei den ersten acht Probanden wurde das Eye-Tracking-System nicht richtig konfiguriert, sodass deren Aufzeichnungen keine effiziente Auswertung zuließ. Mit dem Eye-Tracking-System haben 13 Probanden die Tink- und 12 die eUCRITE-Programmieraufgabe bearbeitet. Die Probanden haben durchschnittlich 30 Minuten für das Lösen der eUCRITE- und 43 Minuten für das Lösen der Tink-Programmieraufgabe benötigt. Die Studiendauer lag im Durchschnitt bei 93 Minuten. Die durchschnittliche Programmiererfahrung der Probanden liegt bei 8 Jahren (Median: 7 Jahre). Die Verteilung ist in Abbildung 2 zu sehen. Für 13 Probanden (59, 09%) stellt diese Studie zum ersten Mal ein Berührungspunkt mit einer Kryptographie-API dar. Die Frage, ob sie schon mal eine Frage auf einer Q&A-Plattform gestellt haben, beantworteten 12 Probanden (57, 14%) mit ja. 6 Probanden (27, 27%) haben schon ein Mal auf eine Frage in einer Q&A-Plattform geantwortet. Abbildung 3 zeigt die Ergebnisse als Balkendiagramm.

[10]Langversion: https://arxiv.org/abs/2007.04983

#	Anforderung \ Methode	Q&A Fragen	Aussage 1	Aussage 2	Aussage 3	Aussage 4	Aussage 5	Eye-Tracking	Benötigte Zeit
1	Schnelle Nutzung der API ermöglichen				⋆				⋆⋆
2	Gute Struktur		⋆	⋆⋆	⋆				
3	Verlinkungen zur Referenz / Signalisierte Text-zu-Code-Verbindungen							⋆⋆	
4	Erweiterte Suche							⋆⋆	
5	Verschiedene Niveaus		⋆⋆		⋆⋆	⋆⋆			
7	Selektiver Zugriff auf den Code aktivieren						⋆	⋆⋆	
9	Viele Beispiele					⋆	⋆	⋆⋆	
14	Klassische Referenz							⋆⋆	
21	Rückfragen und Kommentarfunktion	⋆						⋆⋆	

Tabelle 2: Zuordnung der zur Abfrage genutzten Methoden und der damit abgefragten Anforderungen. Ein ⋆⋆ zeigt eine starke Verbindung der Methode mit der Anforderung. Ein ⋆ zeigt eine schwache Verbindung mit der Anforderung. Je mehr Sterne in einer Zeile sind, desto gründlicher wurde die Anforderung abgefragt. Aussagen aus Abbildung 4.

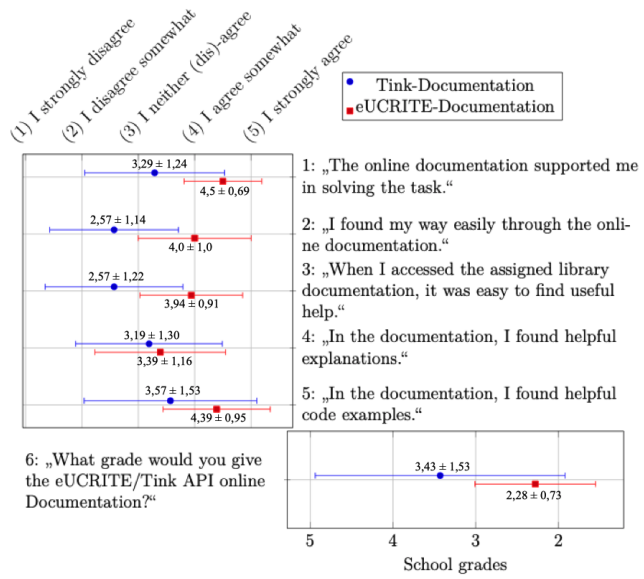

Abbildung 4: Der arithmetische Mittelwert und die Standardabweichung (als Fehlerbalken) der Aussagen 1 – 5 und Frage 6 in Schulnoten.

In dieser Auswertung werden ausschließlich die Fragen zur prototypischen Dokumentation mit der eUCRITE-API betrachtet. Die von den Probanden bewerteten Aussagen sind in Abbildung 4 mit den arithmetischen Mittelwerten und den Standardabweichungen abgebildet. Bei Betrachtung der Abbildung 4 kann für die eUCRITE-API Dokumentation gefolgert werden, dass die Beispiele (Aussage 5) in der Dokumentation geholfen haben, die gestellte Aufgabe zu lösen (Aussage 1). Das Ergebnis der Aussage 4 kann daraus resultieren, dass viele Probanden eventuell eine opportunistische Arbeitsweise (siehe Clark [3]) gewählt haben und dadurch die Textinhalte nicht gelesen haben. Diese These wird in der vom Eye-Tracking-System generierten negativ Heatmap[11] der Startseite unterstützt. Dort ist zu erkennen, dass die Probanden oft das Code-Beispiel zum Generieren der Schlüssel zur Verschlüsselung betrachtet haben. Um dies

[11]Die negativ Heatmap ist in der Langversion der Arbeit zu finden: https://arxiv.org/abs/2007.04983

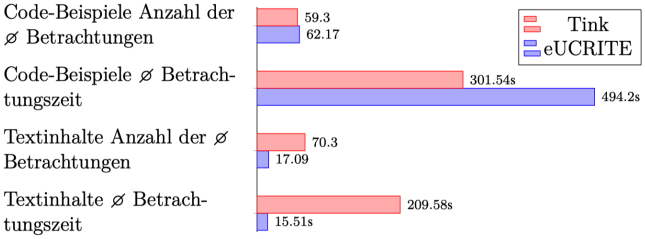

Abbildung 5: AOI-Gruppen der Tink- und eUCRITE-Dokumentation im Vergleich.

mit dem Eye-Tracking-System zu evaluieren, wurden alle Code-Beispiele und Textinhalte in Area of Interests (AOI) eingeteilt und gruppiert. Allgemein lässt sich zu den AOI-Gruppen feststellen, dass die Code-Beispiele ⌀62, 17 Mal und ⌀494, 2 Sekunden lang von Probanden studiert wurden. Textinhalte und Beschreibungen wurden pro Proband mit ⌀17, 09 Ansichten und ⌀15, 51 Sekunden Aufmerksamkeit benutzt. Damit wird den Textinhalten gerade einmal 3% der Betrachtungszeit von Code-Beispielen gewidmet. Dies bestätigt die Vermutung, dass viele Probanden eine opportunistische Arbeitsweise (siehe Clarke [3]) gewählt haben. Zum Vergleich wurde bei der Tink-Dokumentation festgestellt, dass die Code-Beispiele ⌀59, 3 Mal und ⌀301, 54 Sekunden lang studiert wurden. Die Textinhalte wurden ⌀70, 3 Mal und ⌀209, 58 Sekunden betrachtet. Diese Zahlen lassen darauf schließen, dass die Probanden eine pragmatische Arbeitsweise angenommen haben, da sie vergleichsweise kürzer auf die Code-Beispiele aber öfter und länger auf die Textinhalte schauten. Der Vergleich lässt sich in Abbildung 5 ablesen. Viele Probanden haben in der eUCRITE-API-Dokumentation ausschließlich die Startseite genutzt. Proband Nr. 5 hat auf die Frage, was bei der Online-Dokumentation weiter verbessert werden kann folgendes angegeben: *„Bessere Struktur. Das Tutorial habe ich erst ganz zum Schluss gefunden“*. Proband Nr. 2 schrieb: *„Auf [der] Startseite [sind] wenig erklärende Elemente zum Code“*. Er hat dem Tutorial, in dem mehr erklärende Elemente zum Code vorhanden sind, beziehungsweise dem Header-Menü, keine Aufmerksamkeit geschenkt. Die Klick-Zahlen der Eye-Tracking-Aufzeichnungen unterstützen die These, dass viele Probanden ausschließlich die Startseite genutzt haben. Von den 12 Probanden haben ausschließlich 5 die Seite „Tutorial“ aufgerufen. Wir vermuten, dass die Probanden das

Menü-Punkt	Anzahl Probanden	⌀ Fokus	⌀ Betrachtungszeit
Tutorial	4	1	0, 24 Sek.
Referenz	5	2, 4	0, 7 Sek.
Suchfeld	3	1	0, 24 Sek.

Tabelle 3: AOIs des Header-Menüs der Startseite

Header-Menü nicht als Menü erkannt haben oder die Informationen auf der Startseite völlig ausreichend waren. Dies untermauert das in unserer vorherigen Arbeit [7] beschriebene Interaktionsmuster, nach dem sich Probanden bevorzugt über den Schnelleinstieg (was dem Tutorial in [7] entspricht) in das für sie neue Themengebiet der eUCRITE-API eingearbeitet haben.

Der Absprungrate in die klassische Referenz zufolge haben drei Probanden bemerkt, dass die blauen Klassen-Namen in den Code-Beispielen Querverweise (Hyper-Links) in die Referenz darstellen (Anforderung 3). Die anderen 12 Teilnehmer haben die blaue Schrift vermutlich als Syntax-Highlighting interpretiert.

Das Eye-Tracking-System hatte Schwierigkeiten den Bereich des Header-Menüs zu verarbeiten. Weil das Header-Menü über CSS am Browserfenster fixiert ist, folgt es dem Browserfenster beim Scrollen. Dies vollzieht das Eye-Tracking-System nicht nach und ordnet die Blicke auf das Header-Menü und dem Footer Stellen auf der Webseite zu. Daher sind die Eye-Tracking-Zahlen im Header-Menü mit Vorsicht zu interpretieren und es wurden ausschließlich die Blicke von Probanden im Header-Menü gezählt, wenn die Webseite sich im Urzustand befand und nicht gescrollt wurde. Dementsprechend ist davon auszugehen, dass die tatsächlichen Zahlen höher liegen als in Tabelle 3 aufgelistet.

Die Aussage von Proband Nr. 13: *„Besonders gefallen hat mir die übersichtliche Doku und die einfachen Codebeispiele"* unterstützt die Bewertungen der Aussagen 1 und 5. Da zu jeder Methode, Klasse und Funktion ein Beispiel vorhanden ist, kann daraus mit der Aussage des Probanden gefolgert werden, dass Anforderung 9 *Viele Beispiele* erfüllt wurde. Anforderung 21 *Rückfragen und Kommentarfunktion* wurde umgesetzt. Laut der Antworten in Abbildung 3 ist die Hälfte der Probanden mit der Funktionsweise vertraut. 11 Probanden haben die Q&A-Fragen ⌀28 Mal und ⌀19, 01 Sekunden betrachten. 2 Probanden haben aktiv mit ihnen interagiert. Da viele Probanden anscheinend genügend Informationen zum Erfüllen der Programmieraufgabe mit der für sie unbekannten eUCRITE-API auf der Startseite gefunden und ⌀30 Minuten zum Lösen der Programmieraufgabe benötigt haben, kann daraus geschlossen werden, dass Anforderung 1 *Schnelle Nutzung der API ermöglichen* erfüllt ist. Anforderung 7 *Selektiver Zugriff auf den Code aktivieren* unterstützt die opportunistische Arbeitsweise. Da die Probanden dies bevorzugt in der eUCRITE-API-Dokumentation angewendet haben (siehe Schlussfolgerung weiter oben), lässt dies den Schluss zu, dass Anforderung 7 erfüllt ist.

6 LIMITIERUNG

Neben den bereits benannten Limitierungen, weist unsere Studie und die Ergebnisse die folgenden weiteren Limitierungen auf: Eine Aussage, ob das Dokumentationssystem verhindert, dass Programmiere unsicheren Code schreiben, kann nicht getroffen werden, da die Tink- und eUCRITE-API über ihr Design bereits versuchen dies zu verhindern. Aus Zeit- und Ressourcengründen haben wir mit unserer Nutzerstudie zwei unterschiedliche Ziele verfolgt. Neben der Evaluierung des in dieser Arbeit vorgestellten Dokumentationssytems wurde zusätzlich die Benutzbarkeit der eUCRITE-API selbst im Vergleich zur Tink-API untersucht. (Die Benutzbarkeit der eUCRITE-API selbst wurde mit einem Usability Score nach Acar et al. [1] von 70, 5 bewertet. Im Vergleich erreichte die Tink-API einen Score von 48, 23.)

Da die Tink-API in der Github-Umgebung dokumentiert ist, bringt ein direkter Vergleich mit dem prototypischen Dokumentationssystem eine Unschärfe mit sich. Allerdings stehen die Erkenntnisse der Evaluation der prototypischen Umsetzung des Dokumentationssystem für sich. Die Teilnehmer wurden aus dem lokalen Umfeld der Autoren gewonnen, somit könnten diese wohlwollend gehandelt haben und spiegeln keine repräsentative Gruppe wider.

7 ZUSAMMENFASSUNG UND AUSBLICK

Uns ist kein Dokumentationssystem bekannt, welches alle in der Literatur aufgeführten Anforderungen in seiner Gesamtheit umsetzt. In dieser Arbeit wurden diese Anforderungen aus der Literatur extrahiert und in Teilen prototypisch umgesetzt. Hierbei wurden die Anforderungen zweigeteilt. Zum einen die Anforderungen, welche von den bereitgestellten Informationen erfüllt werden sollten. Aus diesen wurde eine Checkliste zur Orientierung für Dokumentationsersteller erarbeitet. Zum anderen in Anforderungen, welche ein optimales Dokumentationssystem erfüllen sollte. Aus diesen Anforderungen wurde ein Konzept eines optimierten Dokumentationssystems entworfen. Auf der Grundlage dieses Konzepts wurde eine prototypische Implementierung eines optimierten Dokumentationssystems erstellt. Im Anschluss wurde in diesem Dokumentationssystem die eUCRITE-API-Dokumentation exemplarisch umgesetzt.

Diese Arbeit kommt zu dem Ergebnis, dass einige der gestellten Anforderungen im Prototypen erfüllt werden konnten. Zum Beispiel die Anforderung 9 *Viele Beispiele*, 21 *Rückfragen und Kommentarfunktion*, 1 *Schnelle Nutzung der API ermöglichen* und 7 *Selektiver Zugriff auf den Code aktivieren*. Bei weiteren besteht jedoch noch Verbesserungsbedarf. Beispielsweise könnte durch eine Umgestaltung des Header-Menüs vermutlich eine Verbesserung der Anforderungen 2 *Gute Struktur*, 14 *Klassische Referenz*, 4 *Erweiterte Suche* und 5 *Verschiedene Niveaus* erzielt werden. Dies soll durch weitere Studien untersucht werden. Es wurde festgestellt, dass die prototypische Implementierung der optimierten Dokumentation die opportunistische Arbeitsweise von Entwicklern nach Clarke [3] unterstützt. In weiteren Studien soll untersucht werden, ob die Dokumentation für pragmatisch und systematisch arbeitende Entwickler ebenso geeignet ist. Insgesamt wurde die prototypische Implementierung von den Teilnehmern der Studie mit der Schulnote „2−" bewertet. Zum Vergleich wurde die via Github gehostete Tink-API-Dokumentation mit einer „3−" bewertet.

In einem nächsten Schritt sollen die nicht betrachteten Anforderungen aus der Literatur in den Prototypen integriert und erneut in Gänze evaluiert werden.

Die Erkenntnisse aus dieser Arbeit sind nicht auf kryptografische APIs beschränkt. Allerdings sind die Auswirkungen von

Programmierfehlern bei kryptografischen APIs aus unserer Sicht vergleichsweise gravierender [17]. Deswegen wurde im Kern dieser Arbeit eine kryptografische API betrachtet.

Eine Dokumentation ist nur so gut, wie sie ihr Ersteller einrichtet und pflegt. Ein Dokumentationssystem sollte nicht ausschließlich die Benutzbarkeit aus der Sicht des Softwareentwicklers betrachten, sondern zusätzlich dem Dokumentationsersteller gute Werkzeuge bieten, um die Dokumentation leicht und teilautomatisiert zu erstellen und zu pflegen. Dieser Aspekt wurde in dieser Arbeit nicht betrachtet, da das Augenmerk dieser Studie darauf gelegt wurde, welches Dokumentationssystem Softwareentwickler am besten unterstützt. In zukünftigen Arbeiten sollte untersucht werden, wie ein API-Dokumentationsersteller unterstützt werden kann, Inhalte einfach und benutzerfreundlich zu integrieren.

8 FÖRDERHINWEIS

Diese Arbeit wurde im Rahmen der Innovationsförderung Hessen aus Mitteln der LOEWE – Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz, Förderlinie 3: KMU-Verbundvorhaben (Projekt HA-Projekt-Nr.: 633/18-5) gefördert sowie vom Bundesministerium für Bildung und Forschung (BMBF) und vom Hessischen Ministerium für Wissenschaft und Kunst (HMWK) im Rahmen ihrer gemeinsamen Förderung für das Nationale Forschungszentrum für angewandte Cybersicherheit ATHENE unterstützt.

LITERATUR

[1] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. 2017. Comparing the Usability of Cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*. 154–171. https://doi.org/10.1109/SP.2017.52

[2] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl. 2017. Developers Need Support, Too: A Survey of Security Advice for Software Developers. In *2017 IEEE Cybersecurity Development (SecDev)*. 22–26. https://doi.org/10.1109/SecDev.2017.17

[3] Steven Clarke. 2007. What is an End User Software Engineerl. In *End-User Software Engineering (Dagstuhl Seminar Proceedings)*, Margaret H. Burnett, Gregor Engels, Brad A. Myers, and Gregg Rothermel (Eds.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany.

[4] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl. 2017. Stack Overflow Considered Harmful? The Impact of Copy amp;Paste on Android Application Security. In *2017 IEEE Symposium on Security and Privacy (SP)*. 121–136. https://doi.org/10.1109/SP.2017.31

[5] A. Heydarnoori, K. Czarnecki, W. Binder, and T. T. Bartolomei. 2012. Two Studies of Framework-Usage Templates Extracted from Dynamic Traces. *IEEE Transactions on Software Engineering* 38, 6 (Nov. 2012), 1464–1487. https://doi.org/10.1109/TSE.2011.77

[6] Rolf Huesmann. 2020. *Konzeption, Erstellung und Evaluierung einer neuen Dokumentationsart für APIs.* Mc Thises. Hochschule Darmstadt, Darmstadt.

[7] Rolf Huesmann, Alexander Zeier, and Andreas Heinemann. 2019. Eigenschaften optimierter API-Dokumentationen im Entwicklungsprozess sicherer Software. In *Mensch und Computer 2019 - Workshopband*. Gesellschaft für Informatik e.V., Bonn. https://doi.org/10.18420/muc2019-ws-302-03

[8] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2018. Application Programming Interface Documentation: What Do Software Developers Want? *Journal of Technical Writing and Communication* 48, 3 (July 2018), 295–330. https://doi.org/10.1177/0047281617721853

[9] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2019. How Developers Use API Documentation: An Observation Study. (2019), 11.

[10] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente. 2013. Documenting APIs with examples: Lessons learned with the APIMiner platform. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 401–408. https://doi.org/10.1109/WCRE.2013.6671315

[11] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. 2016. Jumping Through Hoops: Why Do Java Developers Struggle with Cryptography APIs?. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 935–946. https://doi.org/10.1145/2884781.2884790 event-place: Austin, Texas.

[12] Martin P. Robillard and Robert DeLine. 2011. A field study of API learning obstacles. *Empirical Software Engineering* 16, 6 (Dec. 2011), 703–732. https://doi.org/10.1007/s10664-010-9150-8

[13] Bernd Rohrmann. 1978. Empirische Studien zur Entwicklung von Antwortskalen für die sozialwissenschaftliche Forschung. *Zeitschrift für Sozialpsychologie* 9, 3 (1978), 222–245.

[14] J. Stylos, B. Graf, D. K. Busse, C. Ziegler, R. Ehret, and J. Karstens. 2008. A case study of API redesign for improved usability. In *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. 189–192. https://doi.org/10.1109/VLHCC.2008.4639083

[15] Jeffrey Stylos and Brad A. Myers. 2008. The implications of method placement on API learnability. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering - SIGSOFT '08/FSE-16*. ACM Press, Atlanta, Georgia, 105. https://doi.org/10.1145/1453101.1453117

[16] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API Documentation. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 643–652. https://doi.org/10.1145/2568225.2568313 event-place: Hyderabad, India.

[17] Chamila Wijayarathna, Nalin A. G. Arachchilage, and Jill Slay. 2017. A Generic Cognitive Dimensions Questionnaire to Evaluate the Usability of Security APIs. In *Human Aspects of Information Security, Privacy and Trust (Lecture Notes in Computer Science)*, Theo Tryfonas (Ed.). Springer International Publishing, 160–173.

[18] Alexander Zeier, Alexander Wiesmaier, and Andreas Heinemann. 2019. API Usability of Stateful Signature Schemes. Springer International Publishing.

[19] M. F. Zibran, F. Z. Eishita, and C. K. Roy. 2011. Useful, But Usable? Factors Affecting the Usability of APIs. In *2011 18th Working Conference on Reverse Engineering*. 151–155. https://doi.org/10.1109/WCRE.2011.26