

Hochschule Ulm



BACHELORARBEIT

FAKULTÄT INFORMATIK

STUDIENGANG TECHNISCHE INFORMATIK

Verlinkung und Verifikation von  
gedruckten Dokumenten auf Basis  
von kryptographischen Hashwerten  
aus OCR-Analysen

*vorgelegt von*

***Florian Schneider***

Matr.-Nr.: 3115559

1. Gutachter

**PROF. DR. TRAUB**

2. Gutachter

**PROF. DR. BAER**

# Eigenständigkeitserklärung

Diese Abschlussarbeit wurde von mir selbstständig verfasst. Es wurden nur die angegebenen Quellen und Hilfsmittel verwendet. Alle wörtlichen und sinngemäßen Zitate sind in dieser Arbeit als solche kenntlich gemacht.

6. Februar 2017

Florian Schneider

# Kurzzusammenfassung

In dieser Bachelorarbeit wird ein Konzept zur Verlinkung und Verifikation von gedruckten Dokumenten vorgestellt.

Um ein Dokument zu verlinken, wird welcher textuelle Inhalt des Dokuments mithilfe einer OCR-Analyse extrahiert. Aus dem Inhalt wird ein kryptographischer Hash erzeugt, der digital signiert wird. Der Hash wird zusammen mit der Signatur und dem dazugehörigen digitalen Zertifikat in der dezentralen IPFS-Blockchain verlinkt. Ein Link zu diesen Daten wird im Dokument platziert.

Bei der Verifizierung eines verlinkten Dokuments wird der Link aus dem Dokument extrahiert und die Verifikationsdaten aus dem IPFS-Netzwerk heruntergeladen. Außerdem wird erneut der Inhalt extrahiert und ein kryptographischer Hash erzeugt. Der heruntergeladene Hash wird mit dem neu berechneten Hash verglichen um die Integrität des Dokuments zu überprüfen. Des weiteren wird die Signatur sowie das Zertifikat auf Echtheit geprüft.

Die Funktionalität des entwickelten Konzepts wurde durch eine umfangreiche 'Proof-Of-Concept'-Implementierung bewiesen.

Außerdem wurde die Stabilität der OCR-Erkennung bezüglich der Invarianz der textuellen Ergebnisse getestet und evaluiert. Dabei wurde festgestellt, dass die OCR-Erkennung von komplexen Dokumenten keineswegs trivial ist und mit deutlich mehr Arbeitsaufwand untersucht und optimiert werden muss.

# Danksagung

Hiermit möchte ich mich bei Herrn Prof. Dr. Traub für das interessante Thema sowie die hilfreichen Diskussionen und die allgemeine Betreuung meiner Bachelorarbeit bedanken.

Außerdem möchte ich mich bei Frau K. Bosch bedanken, die dafür gesorgt hat, dass ich die, während der Arbeit benötigten, Geräte und Materialien schnellstmöglich an meinem Arbeitsplatz nutzen konnte.

Besonderer Dank gilt auch meinen Eltern, die mich in allen Belangen rund um das Studium stets sehr unterstützt haben. Ohne ihre Hilfe hätte ich den vollen Fokus nicht auf mein Studium legen können.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Problemstellung . . . . .	2
1.2	Ziel der Arbeit . . . . .	2
1.3	Aufbau der Arbeit . . . . .	3
<b>2</b>	<b>Stand der Technik</b>	<b>4</b>
2.1	Ansätze zur Verlinkung von gedruckten Dokumenten . . . . .	5
2.2	Ansätze zur Erzeugung fälschungssicherer Dokumente . . . . .	5
2.2.1	Wasserzeichenmethoden . . . . .	5
2.2.2	Einbetten der Daten zur Verifizierung in ein Dokument . . . . .	6
2.2.3	Speicherung der Daten zur Verifizierung eines Dokuments in einer Datenbank . . . . .	6
<b>3</b>	<b>Grundlagen</b>	<b>7</b>
3.1	Optical Character Recognition . . . . .	8
3.2	Kryptographische Hashwerte . . . . .	9
3.3	Digitale Signaturen . . . . .	9
3.4	Blockchain . . . . .	10
<b>4</b>	<b>Umsetzung</b>	<b>11</b>
4.1	Anforderungsanalyse . . . . .	12
4.1.1	Anwendungsfälle . . . . .	12
4.1.2	Anforderungen . . . . .	16
4.2	Beschreibung der Lösungsansätze . . . . .	19
4.2.1	Stabilitätstests von OCR-Ergebnissen . . . . .	19
4.2.2	Recherche über Möglichkeiten zur Verlinkung von Dokumenten . . . . .	19
4.2.3	Proof-Of-Concept Implementierung . . . . .	21
4.3	Tests zur Stabilität der OCR-Ergebnisse . . . . .	22
4.3.1	Beschreibung der allgemeinen Testumgebung . . . . .	22
4.3.2	OCR-Testphase 1 . . . . .	23
4.3.3	OCR-Testphase 2 . . . . .	26
4.3.4	OCR-Testphase 3 . . . . .	32
4.3.5	OCR-Testphase 4 . . . . .	39
4.3.6	Mögliche Ursachen der schlechten Testergebnisse . . . . .	43
4.4	Gesamtüberblick über das entwickelte Konzept . . . . .	44
4.4.1	Verlinkung von Dokumenten . . . . .	45
4.4.2	Verifikation von Dokumenten . . . . .	46
4.4.3	Abgrenzung zu den Arbeiten aus Kapitel 2 . . . . .	47
4.5	Proof-Of-Concept Software Design . . . . .	48
4.5.1	Verwendete Software-Bibliotheken . . . . .	48
4.5.2	Überblick über die Datentypen . . . . .	52

4.5.3	'High-Level' Architektur . . . . .	54
4.5.4	'High-Level' Algorithmus . . . . .	56
4.5.5	Architektur und Algorithmen der einzelnen Module . . . . .	58
<b>5</b>	<b>Ergebnisse</b>	<b>76</b>
5.1	Überprüfung der Anforderungen . . . . .	77
5.1.1	Anforderungen des Typs 'Frage' . . . . .	77
5.1.2	Anforderung des Typs 'Funktionalität' . . . . .	80
5.1.3	Anforderung des Typs 'Design' . . . . .	85
5.1.4	Übersicht der Anforderungsergebnisse . . . . .	86
5.2	Beispielhafte Ausführungen des 'Printed Document Linking System' .	87
5.3	Schwachstellen der Implementierung . . . . .	91
5.3.1	OCR . . . . .	91
5.3.2	PDF-Dokumente . . . . .	91
5.4	Sicherheitskritische Schwachstellen des Systems . . . . .	91
<b>6</b>	<b>Zusammenfassung und Ausblick</b>	<b>92</b>
6.1	Zusammenfassung . . . . .	93
6.2	Ausblick . . . . .	94
6.2.1	Vorschläge zur Verbesserung der OCR-Ergebnisse . . . . .	94
6.2.2	Allgemeine Verbesserungs- und Erweiterungsvorschläge . . . . .	95

# Kapitel 1

## Einleitung

## 1.1 Problemstellung

Aussteller von wichtigen Dokumenten wie zum Beispiel Hochschulen oder Unternehmen stehen seit jeher vor dem Problem der Dokumentenfälschung, vor allem wenn die Dokumente verteilt werden sollen.

Dieses Problem kann, wenn die Dokumente nur digital verschickt werden, umgangen werden indem sie digital signiert werden. Durch die digitale Signatur kann die Integrität sowie Urheberschaft garantiert werden. Hierbei gibt es jedoch einen einfachen Weg, wie diese Signaturen umgangen werden können. Der Empfänger druckt das Dokument einfach aus und scannt es wieder ein. Der Scan ist nun nach Belieben fälschbar, und gefälschte Kopien können vom Empfänger zum Beispiel an zukünftige Arbeitgeber weiter geschickt werden. Der Arbeitgeber kann nun natürlich die Kopie beim Urheber des Dokuments verifizieren lassen, was allerdings umständlich ist und deshalb selten passiert.

Eine andere Möglichkeit besteht darin, dass der Urheber das Dokument ausdruckt und real signiert und/oder mit einem Stempel versieht. Da nur das echte, unterschriebene Dokument gültig ist, ist es für den Empfänger jetzt deutlich schwerer das Dokument zu fälschen. Allerdings kann der Empfänger jetzt keine einfachen Kopien mehr vom Dokument machen, sondern muss diese bei einer zentralen Stelle beglaubigen lassen.

Eine dritte Möglichkeit ist es, die Dokumente vor dem Versenden digital im Internet zu verlinken. Durch diese Verlinkung ist eine Verifizierung der Dokumente sowie deren Kopien möglich. Das Problem bei diesem Verfahren ist, dass die Dokumente üblicherweise durch URLs oder URNs referenziert sind. URLs sind bei Verschiebungen problematisch. URNs erzeugen die Eindeutigkeit der Referenz mithilfe eines zentralen Namensdienstes. Wenn dieser ausfällt, sind die Dokumente nicht mehr abrufbar.

## 1.2 Ziel der Arbeit

Ziel dieser Bachelorarbeit ist es eine Lösung zu finden, wie gedruckte Dokumente digital und dezentral ohne die Nutzung von URLs oder URNs in einer DHT und/oder Blockchain verlinkt werden können. Digitale sowie gedruckte Kopien dieser Dokumente müssen durch diese Verlinkung verifiziert werden können. Zusätzlich bedarf es einer Lösung, wie digitale, neu erstellte Dokumente in das Schema mit aufgenommen werden können.

Dafür soll ein prototypischer, sogenannter 'Proof-Of-Concept' implementiert werden, der zeigt, ob der in dieser Arbeit vorgeschlagene Lösungsansatz funktioniert und wo es zu Problemen kommt.

Wie bereits im Titel zu lesen ist, soll der Inhalt der gedruckten Dokumente über eine OCR-Erkennung extrahiert werden. Deshalb ist die Frage nach der Stabilität bezüglich der Invarianz der OCR-Ergebnisse von verschiedenen Dokumenten zu beantworten.

Des Weiteren muss ein Algorithmus zur Normalisierung des OCR-Ergebnisses entwickelt werden, um überflüssige Zeichen zu entfernen und somit eine normierte Basis für einen Hash-Algorithmus zu schaffen.

Außerdem soll beantwortet werden, ob mehrere Kopien eines Dokuments über dieselbe ID referenziert werden können.



## 1.3 Aufbau der Arbeit

In dieser schriftlichen Ausarbeitung wird zunächst im Stand der Technik in Kapitel 2 auf Arbeiten beziehungsweise Ansätze mit verwandter Ziel- und Problemstellung eingegangen. Im darauf folgenden Kapitel 3 sind die Grundlagen, welche zum Verständnis der Arbeit und speziell des vorgeschlagenen Ansatzes nötig sind, kurz und vereinfacht zusammengefasst. Die Beschreibung sowie Umsetzung des, in dieser Arbeit vorgeschlagenen, Konzepts sowie eine detaillierte Beschreibung der Ziele und Anforderungen der Arbeit wird im Kapitel 4 erläutert. Die dabei gewonnenen Erkenntnisse und Ergebnisse werden im Kapitel 5 diskutiert. Eine Zusammenfassung der gesamten Arbeit sowie Verbesserungs- und Erweiterungsvorschläge finden sich im Kapitel 6 wieder.

# Kapitel 2

## Stand der Technik

## 2.1 Ansätze zur Verlinkung von gedruckten Dokumenten

Arbeiten mit dem Ziel gedruckte Dokumente online zu verlinken, wurden bei der Recherchearbeit zu diesem Kapitel, trotz umfangreicher Suche nicht gefunden. Dies hat den Vorteil, dass dadurch der wissenschaftliche Neuheitsgehalt der Arbeit garantiert ist.

## 2.2 Ansätze zur Erzeugung fälschungssicherer Dokumente

In diesem Abschnitt werden Ansätze aus anderen wissenschaftlichen Arbeiten beschrieben um fälschungssichere Dokumente zu erzeugen. Die meisten dieser Ansätze passen nicht exakt in den Rahmen dieser Arbeit. Ziel ist es lediglich Ideen und Vorschläge zu sammeln, um im späteren Verlauf der Arbeit ein gutes Konzept zu entwickeln.

### 2.2.1 Wasserzeichenmethoden

Viele Ansätze um gedruckte Dokumente fälschungssicher zu machen, verwenden Wasserzeichen, die auf die Dokumente gedruckt werden. Am bekanntesten sind hier wohl Geldscheine. Da diese Wasserzeichen nur von speziellen Druckern erzeugt werden können, scheidet diese Methode aus.

Die Arbeiten [1] und [2] schlagen drei alternative Wasserzeichenmethoden vor, die von allen gängigen Druckern gedruckt werden können. Diese Wasserzeichen unterscheiden sich jedoch sehr stark von herkömmlichen Wasserzeichen und haben auch nichts mehr mit dem eigentlichen Begriff beziehungsweise dessen Ethymologie zu tun. Gemeint sind eher Muster, die man nur sehen kann, wenn man die Vorschriften zur Erzeugung der Muster kennt. Meistens sind diese Wasserzeichen sogar nur für Maschinen identifizierbar.

Eine Art der in [1] und [2] vorgeschlagenen alternativen Wasserzeichenmethoden basiert auf der Manipulation der Bildern von Dokumenten. Diese Art, die der digitalen Steganographie [3] ähnelt, wird auch in [4] umgesetzt. Hier wird das Bild nach bestimmten Rechenvorschriften verändert, um ein unsichtbares Muster im Dokument zu hinterlegen, was bei der Verifizierung überprüft wird. Einen ähnlichen Ansatz verwendet die Arbeit [5], bei der Zeichen und/oder Wörter nach einer bestimmten Vorschrift leicht verschoben werden. Das Muster besteht dann aus den Unterschieden zwischen den Abständen der Zeichen beziehungsweise Wörter.

Diese Ideen könnten übernommen werden. Die Vorschrift, mit der das Bild des Dokuments verändert wurde, könnte in einer Blockchain oder DHT gespeichert werden. Der Link zu dieser Vorschrift könnte dann in einem bestimmten, isolierten Bereich in das Dokument eingefügt werden. Bei der Verifizierung eines solchen Dokuments wird dann die Mustervorschrift heruntergeladen und das Dokument anhand der heruntergeladenen Vorschrift überprüft.

Andere alternative Wasserzeichenmethoden zur Erzeugung von fälschungssicheren Dokumenten basieren auf morphosyntaktischen oder morphosemantischen Veränderungen des Inhalts der Dokumente. Das heißt es wird der Inhalt nach einem bestimmten und geheimen Muster so verändert, dass der veränderte Inhalt zwar dasselbe ausdrückt, jedoch anders formuliert ist als das Original. Die Arbeiten [6],

[7], [8], [9], [10], [11], [12] und [13] setzen diese Vorgehensweise durch komplexe 'Natural Language Processing'-Algorithmen um, die eine geheime Veränderungsvorschrift umsetzen. Diese Verfahren fallen aufgrund der Komplexität der NLP-Algorithmen weg. Außerdem lassen sie sich nur auf Texte anwenden, bei denen der exakte Inhalt keine Rolle spielt, was bei Zeugnissen zum Beispiel nicht der Fall ist.

### **2.2.2 Einbetten der Daten zur Verifizierung in ein Dokument**

Das US-Patent US 5606609 A [14] aus dem Jahr 1997, beschreibt ein Verfahren, bei dem ein 'security object' in ein digitales Dokument eingebettet wird. Dieses Objekt enthält alle Daten, wie zum Beispiel einen kryptographischen Hash des Dokumenteninhalts, die digitale Signatur und/oder ein Bild der handschriftlichen Signatur des Verfassers, um das Dokument später zu verifizieren. Ähnliche Verfahren werden heute abgewandt, um zum Beispiel digitale PDFs oder Office-Dokumente zu unterschreiben.

Da sich diese Vorgehensweise nur auf digitale Dokumente anwenden lässt, hat sie für diese Arbeit wenig Nutzen. Was allerdings übernommen werden kann, ist die Idee, die Verifikationsdaten zu bündeln. Es muss dann allerdings eine Lösung gefunden werden, wie diese Verifikationsdaten mit einem gedruckten Dokument verbunden beziehungsweise verlinkt werden können, ohne auf einen zentralen Service angewiesen zu sein.

Die Ansätze aus den Arbeiten [15], [16], [17], [18] und [19] basieren auf 2D-Barcodes beziehungsweise QR-Codes. In diesen Barcodes sind die Daten zur Verifikation eines Dokuments kodiert und gebündelt. Bei der Verifikation werden diese Daten aus dem Barcode extrahiert und überprüft. Ein Vorteil hierbei ist, dass die Verifikation offline geschehen kann. Der große Nachteil ist die limitierte Speicherkapazität der Codes. Ein QR-Code kann maximal 2953 Bytes speichern, wobei jedoch beachtet werden muss, dass ein solcher (40-L)-QR-Code von einfachen QR-Code-Scannern aufgrund von Darstellungsproblemen nicht mehr erkannt wird. Die Idee einen Barcode zu nutzen, kann jedoch übernommen werden.

### **2.2.3 Speicherung der Daten zur Verifizierung eines Dokuments in einer Datenbank**

In dem US-Patent US 7051086 B2 [20] wird ein Ansatz beschrieben, ergänzende Daten zu einem Dokument in einer Datenbank zu speichern. Im Patent werden diese Daten nicht genau spezifiziert, theoretisch können aber sämtliche Arten von Daten in einer Datenbank gespeichert werden. Das heißt auch Daten, welche sich zur Verifikation eines Dokuments nutzen lassen können. Verlinkt werden die Daten durch eine ID, welche auf das Dokument gedruckt wird. Bei der Verifikation werden die Daten über die ID heruntergeladen und überprüft.

Die Arbeit [21] geht nach dem gleichen Prinzip vor, wobei dort ausschließlich Verifikationsdaten in der Datenbank gespeichert werden.

Das Problem bei diesen Ansätzen ist die Nutzung einer zentralen Datenbank. Wenn die Verbindung zu dieser Datenbank nicht aufgebaut werden kann, ist auch keine Verifizierung mehr möglich. Die Idee die Verifizierungsdaten eines Dokuments und nicht das Dokument selbst zu speichern, kann jedoch übernommen werden.

# Kapitel 3

## Grundlagen

In diesem Kapitel werden die absoluten Grundlagen, die zum Verständnis des in Kapitel 4 vorgeschlagenen Konzepts unbedingt benötigt werden, kurz und vereinfacht erläutert. Eigentlich wird davon ausgegangen, dass der Leser bereits mit diesen Begriffen vertraut ist, weshalb die folgenden Abschnitte nur als Erinnerungsstütze dienen sollen. Für genauere Informationen können die Literaturhinweise hilfreich sein.

## 3.1 Optical Character Recognition

‘Optical Character Recognition’ wird mit OCR abgekürzt und heißt in die deutsche Sprache übersetzt, optische Zeichenerkennung oder auch Texterkennung. Wie der Name des Begriffs schon andeutet, geht es bei der Texterkennung darum, Zeichen beziehungsweise Text auf digitalen Bildern zu erkennen. Die Eingabe eines OCR-Algorithmus ist also ein Bild, zum Beispiel ein Scan eines Dokuments. Die Ausgabe ist, der im Idealfall fehlerfrei erkannte textuelle Inhalt des Scans. Unterschieden wird in Algorithmen, die digitale Schriftarten erkennen können und Algorithmen, welche menschliche Handschrift erkennen können. In dieser Arbeit geht es ausschließlich um Verfahren zur Erkennung von digitalen Schriftarten.

Algorithmen dieser Art stammen aus dem Bereich der digitalen Bildverarbeitung und erkennen Zeichen durch Mustervergleiche. Die Muster der einzelnen Zeichen werden in einer Datenbank oder einer ähnlichen Struktur gespeichert und müssen anhand von Schriftarten oder Vektorgrafiken generiert werden. Dieser Prozess wird oft auch als ‘Trainieren’ bezeichnet. Je größer der Datensatz der Muster, auch ‘Trainingsdaten’ genannt, ist, desto akkurater ist die OCR-Analyse. OCR-Algorithmen arbeiten mehrstufig, wobei die erste Stufe die Erkennung und Isolierung von Textblöcken im Eingangsbild ist. Hier wird auch oft eine Vorverarbeitung der Eingangsbilder, wie zum Beispiel die Erstellung eines Binärbilds und/oder Rauschunterdrückung, gemacht. Die nächste Stufe beinhaltet die eigentliche Mustererkennung, welche oft auf Basis von ‘Template-Matching’-Algorithmen basiert, und auf den Textblöcken der vorherigen Stufe arbeitet. Neuartige Ansätze arbeiten auf Basis von Neuronalen Netzwerken und Deep Learning, die ähnlich wie Standardverfahren mit gelabelten Datensätzen trainiert werden. Diese Ansätze sind deutlich akkurater als herkömmliche Methoden, benötigen allerdings deutlich mehr Rechenleistung. Oft werden in dieser Stufe auch Fehlerkorrekturen auf Zeichen- und/oder Wörterbasis gemacht. Diese Fehlerkorrekturen basieren auf Wörterbüchern, die man den meisten Algorithmen mitgeben kann. Die letzte Stufe konvertiert die erkannten Zeichen in das gewünschte Ausgabeformat. (vgl. [22], [23] & [24])

Da diese Verfahren sehr komplex sind, werden im Normalfall bereits vorhandene Bibliotheken hergenommen, um die OCR-Funktionalität in einem Projekt zu nutzen. Die folgende Liste beinhaltet eine Auswahl an Open-Source OCR-Bibliotheken.

- OCRopus [25]
- GOCR [26]
- CuneiForm [27]
- Ocrad [28]
- Tesseract [29]

## 3.2 Kryptographische Hashwerte

Kryptographische Hashwerte sind das Ergebnis einer kryptographischen Hashfunktion. Dies ist eine spezialisierte Version einer Hashfunktion, bei der es praktisch unmöglich ist, eine Eingabe E1 zu finden, welche denselben Hashwert wie eine andere Eingabe E2 als Resultat der Funktion liefert. [30]

Kryptographische Hashwerte werden daher genutzt, um die Integrität von Nachrichten zu überprüfen. Dabei wird eine Originalnachricht (in diesem Fall der Inhalt eines Originaldokuments) als Eingabe einer kryptographischen Hashfunktion bereitgestellt und der resultierende Hashwert notiert. Soll nun eine Kopie des Dokuments überprüft werden, wird der Inhalt der Kopie als Eingabe derselben Hashfunktion verwendet und der Ausgabe-Hashwert mit dem Original-Hashwert verglichen. Sind die beiden Hashes gleich, gilt die Integrität der Nachricht als bestätigt.

## 3.3 Digitale Signaturen

Digitale Signaturen werden verwendet, um die Integrität eines Dokuments zu sichern und die Identität des Unterzeichners und damit auch Verfassers des Dokuments eindeutig festzulegen. Dieses Verfahren wird auf Basis eines asymmetrischen Kryptosystems, wie zum Beispiel RSA [31], und einer Public-Key-Infrastructure (PKI) [32] durchgeführt. (vgl. [33] & [34])

Zu einer digitalen Signatur gehört immer ein Public-Key-Private-Key Schlüsselpaar. Um die Signatur zu erzeugen, wird die zu signierende Nachricht mit dem Private-Key des Unterzeichners verschlüsselt. Das Ergebnis dieser Verschlüsselung wird als Signatur der Nachricht beziehungsweise des Dokuments bezeichnet. Bei der Überprüfung der Signatur wird der Public-Key des Schlüsselpaars verwendet, um die Signatur zu entschlüsseln. Die entschlüsselte Nachricht wird daraufhin mit der Originalnachricht verglichen. Sind beide Nachrichten gleich, gilt die Signatur als verifiziert. Hierbei ist anzumerken, dass aufgrund von Beschränkungen in asymmetrischen Kryptosystemen, nicht die Nachricht beziehungsweise das Dokument selbst signiert wird, sondern ein kryptographische Hashwert, welcher aus der Nachricht generiert wurde. Um die Identität des Verfassers beziehungsweise Unterzeichners zu bestätigen, wird nicht der 'rohe' öffentliche Schlüssel verwendet, sondern ein digitales Zertifikat, in das der öffentliche Schlüssel eingebettet ist. Das Zertifikat ist Bestandteil einer PKI und enthält neben dem öffentlichen Schlüssel noch den Namen und die Organisation des Besitzers, für den das Zertifikat ausgestellt wurde, sowie weitere (teilweise optionale) Informationen zum Besitzer. Zusätzlich enthält das Zertifikat Informationen zu dessen Aussteller (engl. Certificate Authority oder CA). Um ein digitales Zertifikat zu erhalten, muss ein Antrag bei einem offiziellen Zertifikatsaussteller beantragt werden, der dann die Identität des Antragstellers überprüft. Ein solcher Aussteller ist in Deutschland zum Beispiel die Deutsche Telekom.

## 3.4 Blockchain

In diesem Abschnitt ist zu beachten, dass es sich um eine sehr vereinfachte und unvollständige Erklärung einer Blockchain handelt. Ziel ist es, nur das Grundprinzip einer Blockchain zu erläutern. Für eine genaue Erklärung der technischen Umsetzung muss die Spezifikation des spezifischen Blockchain-Betreibers eingesehen werden, da jede Blockchain Implementierung unterschiedlich ist.

Eine Blockchain ist im Prinzip eine dezentrale, öffentliche Datenbank, die, wie der Name schon sagt, aus einer Verkettung einzelner Blöcke besteht. Diese Blöcke können Daten jeglicher Art enthalten. Bei der Bitcoin-Blockchain [35] sind die Daten (unter anderem) Transaktionen der Bitcoins; bei der Ethereum-Blockchain [36] sind es (unter anderem) Smart-Contracts [37]; bei der IPFS-Blockchain sind es (unter anderem) Merkle-Tree-Hashes [38]. Zusätzlich zu den Daten beinhaltet jeder Block, einen kryptographischen Hash, der aus den Daten des vorherigen Blocks generiert wurde (siehe Abbildung 3.1). Durch diesen Hash wird die Integrität der Daten garantiert. (vgl. [39])

Wie bereits erwähnt, ist eine Blockchain-Datenbank dezentral und öffentlich. Das heißt, alle Knoten im Netzwerk hosten eine Kopie der Blockchain. Wird nun ein neuer Datensatz beziehungsweise Block zur Blockchain hinzugefügt, wird der Block im gesamten Netzwerk verteilt. Bei der Verkettung des neuen Blocks wird der Hash der Daten des Vorgängerblocks gebildet und in den neuen Block eingefügt.

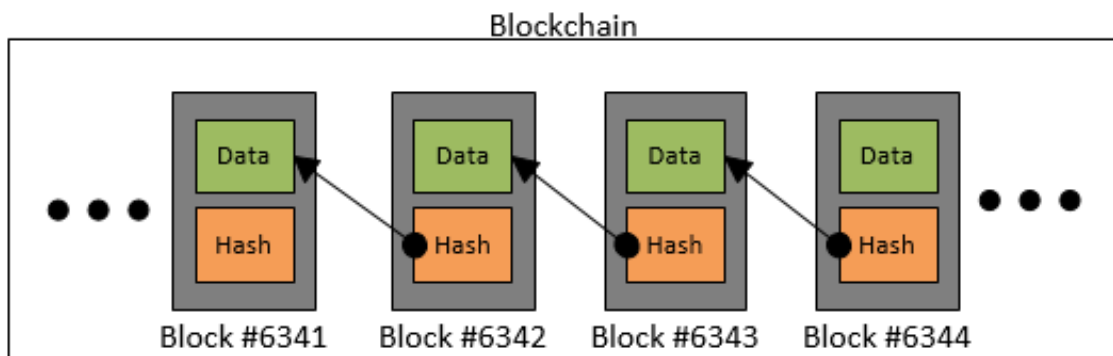


Abbildung 3.1: Vereinfachte schematische Darstellung des Blockchain Prinzips



# Kapitel 4

## Umsetzung

## 4.1 Anforderungsanalyse

In diesem Abschnitt wird die, im Rahmen dieser Bachelorarbeit, betriebene Anforderungsanalyse beschrieben. Ziel dieser Analyse ist es, die Anforderungen und Ziele der Arbeit zu ermitteln und zu strukturieren, sodass diese im späteren Verlauf der Arbeit einfach überprüft und bewertet werden können.

### 4.1.1 Anwendungsfälle

Aus der Zielstellung der Arbeit wurden drei Anwendungsfälle generiert, auf die im diesem Abschnitt eingegangen wird (siehe Abbildung 4.1).

Diese Anwendungsfälle beziehen sich alle auf Notenspiegel Testdokumente (siehe Abschnitt 4.2.1). Dies kommt daher, dass ein zukünftiges Anwendungsgebiet des, in dieser Arbeit vorgeschlagenen, Ansatzes, die Zeugnis- beziehungsweise Notenspiegelausgabe der Hochschule Ulm ist. Da nur untersucht werden soll, ob der Ansatz generell durchführbar ist, werden von der Struktur her einfachere Testdokumente und keine echten Notenspiegel Dokumente der Hochschule genutzt.

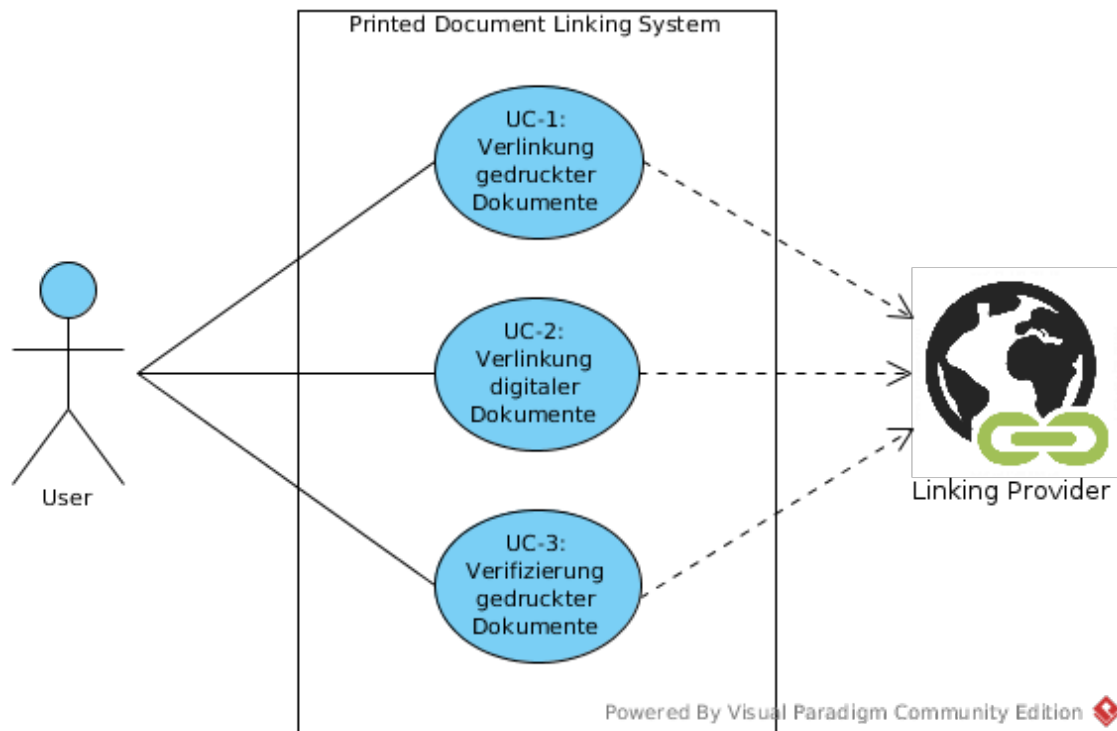


Abbildung 4.1: Use-Case Diagramm des Proof-Of-Concept

Im Folgenden wird pro Anwendungsfall erst die, als Best-Practice angesehene, Use-Case Tabelle in Englischer Sprache und danach eine kurze deutsche Zusammenfassung gezeigt. In den Tabellen sind bewusst alternative Ausführungen im Fehlerfall der einzelnen Schritte weg gelassen worden. Der Grund hier für ist, dass das System bei jeder Art von Fehler eine entsprechende Fehlermeldung ausgibt und der Vorgang abgebrochen wird.

Alle Anwendungsfälle haben die gleiche Priorität, weshalb die Reihenfolge ignoriert werden kann.

#### 4.1.1.1 Use-Case #1: Verlinkung eines gescannten Testdokuments

<b>ID</b>	UC-1
<b>Title</b>	Linking of a scanned test document at a linking provider
<b>Description</b>	A scan of a test document linked at a linking provider. The User obtains an output document containing the original document and the injected Link.
<b>Actors</b>	User, System
<b>Precondition</b>	<ul style="list-style-type: none"><li>• Document is in Image or PDF format.</li><li>• RSA Private-Key is available in PEM format file.</li><li>• X.509 Certificate is available in PEM format file.</li><li>• Password for the RSA Private-Key PEM file is known.</li><li>• System with Modules is available</li></ul>
<b>Postcondition</b>	<ul style="list-style-type: none"><li>• Output-Document consists of Input-Document and Link</li><li>• Linking-Data can be downloaded via Link</li><li>• Linking-Data contains the generated Content-Hash, Digital Signature and X.509 Certificate</li></ul>
<b>Basic Flow</b>	<ol style="list-style-type: none"><li>1. User provides Input-Document, RSA Private-Key and X.509 Certificate as input to the System</li><li>2. System generates Output-Document with injected Link</li><li>3. User obtains Output-Document</li></ol>
<b>Success Scenario</b>	User can open, read and print Output-Document containing the injected Link

*Tabelle 4.1: UseCase: Verlinkung eines gedruckten Testdokument*

In diesem Anwendungsfall geht es um die Verlinkung gescannter Testdokumente (siehe Abbildung 4.12). Ein solches Dokument soll bei einem Provider verlinkt werden (dieser Vorgang ist schematisch in Abbildung 4.14 beschrieben). Der User stellt dem System einen Privaten Schlüssel, das dazu passende Zertifikat und einen Scann des Testdokuments in PDF- oder Bildformat als Eingabe zur Verfügung. Das System liefert bei erfolgreicher Verlinkung ein Ausgabedokument, welches aus dem Original Dokument und dem eingefügten Link besteht.

#### 4.1.1.2 Use-Case #2: Verlinkung eines neu erstellten Testdokuments

<b>ID</b>	UC-2
<b>Title</b>	Linking of a pure digital test document at a linking provider
<b>Description</b>	An test document that was newly created and is not printed gets linked at a linking provider. The User obtains an output document containing the original document and the injected Link.
<b>Actors</b>	User, System
<b>Precondition</b>	<ul style="list-style-type: none"> <li>• Document is in Text, Image, PDF or XSL.FO format.</li> <li>• RSA Private-Key is available in PEM format file.</li> <li>• X.509 Certificate is available in PEM format file.</li> <li>• Password for the RSA Private-Key PEM file is known.</li> <li>• System with Modules is available</li> </ul>
<b>Postcondition</b>	<ul style="list-style-type: none"> <li>• Output-Document consists of Input-Document and Link</li> <li>• Linking-Data can be downloaded via Link</li> <li>• Linking-Data contains the generated Content-Hash, Digital Signature and X.509 Certificate</li> </ul>
<b>Basic Flow</b>	<ol style="list-style-type: none"> <li>1. User provides Input-Document, RSA Private-Key and X.509 Certificate as input to the System</li> <li>2. System generates Output-Document with injected Link</li> <li>3. User obtains Output-Document</li> </ol>
<b>Success Scenario</b>	User can open, read and print Output-Document containing the injected Link

*Tabelle 4.2: UseCase: Verlinkung eines neu erstellten Testdokument*

In diesem Anwendungsfall geht es um die Verlinkung eines neu erstellten Testdokumente (siehe Abbildung 4.11). Ein solches Dokument soll bei einem Provider verlinkt werden (dieser Vorgang ist schematisch in Abbildung 4.14 beschrieben). Der User stellt dem System einen Privaten Schlüssel, das dazu passende Zertifikat und ein Testdokument in Text-, XSL.FO-, PDF- oder Bildformat als Eingabe zur Verfügung. Das System liefert bei erfolgreicher Verlinkung ein Ausgabedokument, welches aus dem Original Dokument und dem eingefügten Link besteht.

#### 4.1.1.3 Use-Case #3: Verifizierung eines gedruckten Testdokument

<b>ID</b>	UC-3
<b>Title</b>	Verification of a previously linked scan of a test document .
<b>Description</b>	A scan of a test document that was previously linked at a provider gets verified. The User receives the verification status in the terminal.
<b>Actors</b>	User, System
<b>Precondition</b>	<ul style="list-style-type: none"><li>• Document is in Image or PDF format.</li><li>• Document contains a valid Link to it's linked data</li><li>• System with Modules is available</li></ul>
<b>Postcondition</b>	<ul style="list-style-type: none"><li>• If the Document is valid, the system prints VALID</li><li>• If the Document, Signature or Certificate is invalid, the system prints INVALID</li></ul>
<b>Basic Flow</b>	<ol style="list-style-type: none"><li>1. User provides scanned document as file to the System</li><li>2. System evaluates document verification state by checking the linked data via the Link in the document</li><li>3. User can read document verification state in terminal</li></ol>
<b>Alternative Flow</b>	<ol style="list-style-type: none"><li>1.1. User provides Link to linked data of the document</li></ol>
<b>Success Scenario</b>	The user reads VALID or INVALID in the terminal depending if the document is valid or not.

*Tabelle 4.3: UseCase: Verifizierung eines gedruckten Testdokument*

In diesem Anwendungsfall geht es um die Verifizierung gedruckter und bereits verlinkter Testdokumente (dieser Vorgang ist schematisch in Abbildung 4.16 beschrieben). Der User stellt dem System einen Privaten Schlüssel, das dazu passende Zertifikat und einen Scan des Testdokuments in PDF- oder Bildformat als Eingabe zur Verfügung. Das System ermittelt den Verifikationsstatus des Dokuments und zeigt diesen im Terminal an.

### 4.1.2 Anforderungen

Die in Tabelle 4.6 dargestellten Anforderungen wurden, wie in der Spalte 'Ursprung' ausgedrückt, aus den ursprünglich zu lösenden Fragestellungen beziehungsweise Zielen der Bachelorarbeit oder den Anwendungsfällen aus Abschnitt 4.1.1 generiert. In der Spalte 'Typ' wird die Art der Anforderung gekennzeichnet, was in Tabelle 4.4 gezeigt wird. Die Bedeutung sowie der Wert der Priorität einer Anforderung ist in Tabelle 4.5 erläutert. Der Wert einer Anforderung dient zum Vergleich der Anforderungsprioritäten.

Typ	Bedeutung
Frage	Anforderungen dieses Typs sind Fragestellungen die im Verlauf der Arbeit beantwortet werden sollen.
Funktionalität	Anforderungen dieses Typs sind funktionale Anforderungen an die Proof-Of-Concept Implementierung.
Design	Anforderungen dieses Typs sind nicht funktionale Anforderungen und beziehen sich auf das Software Design der Proof-Of-Concept Implementierung

*Tabelle 4.4: Bedeutung der Anforderungstypen*

Priorität	Wert	Bedeutung
Hoch	7	Anforderungen mit dieser Priorität gilt es in jedem Fall zu erfüllen beziehungsweise zu beantworten.
Mittel	3	Anforderungen mit dieser Priorität sollten erfüllt beziehungsweise beantwortet werden nach dem alle Anforderungen der Priorität 'Hoch' erfüllt worden sind.
Optional	1	Anforderungen mit dieser Priorität müssen nicht erfüllt werden, es wäre allerdings praktisch und sollte in eventuell weiterführenden Arbeiten erfüllt werden.

*Tabelle 4.5: Bedeutungen der Anforderungsprioritäten*

<b>Titel</b>	<b>ID</b>	<b>Typ</b>	<b>Ursprung</b>	<b>Priorität</b>
Wie stabil ist die OCR Erkennung?	R1	Frage	Zielstellung	Hoch
Wie kann ein Dokument in einer DHT oder Blockchain referenziert werden?	R2	Frage	Zielstellung	Hoch
Können Kopien von Dokumenten über die gleiche ID referenziert werden?	R3	Frage	Zielstellung	Mittel
Kann ein Dokument über seinen Content-Hash referenziert werden?	R4	Frage	Zielstellung	Mittel
Wie kann eine Abbildung vom Content-Hash auf die Binär-Hashes verwaltet werden?	R5	Frage	Zielstellung	Optional
Extraktion des Inhaltes eines Dokuments	R6	Funktionalität	Use Cases	Hoch
Normalisierung des Inhaltes eines Dokuments	R7	Funktionalität	Use Cases	Hoch
Erzeugung des Content-Hash eines Dokuments	R8	Funktionalität	Use Cases	Hoch
Erzeugen einer digitalen Signatur eines Dokuments	R9	Funktionalität	Use Cases	Optional
Verlinkung eines Dokuments in einer 'Distributed-Hash-Table' oder Blockchain	R10	Funktionalität	Use Cases	Hoch
Injektion eines Links in ein Dokument	R11	Funktionalität	Use Cases	Mittel
Extraktion eines Links aus dem Inhalt eines Dokuments	R12	Funktionalität	Use Cases	Hoch
Auflösen eines Links und Laden der Verlinkungsdaten eines Dokuments	R13	Funktionalität	Use Cases	Hoch
Überprüfung des Inhalts eines Dokuments	R14	Funktionalität	Use Cases	Hoch
Überprüfung der Signatur eines Dokuments	R15	Funktionalität	Use Cases	Optional
Unterstützung von Dokumenten im PDF Format	R16	Funktionalität	Use Cases	Hoch
Unterstützung von Dokumenten in gängigen Bild Formaten	R17	Funktionalität	Use Cases	Optional
Unterstützung von Dokumenten im XSL.FO Format	R18	Funktionalität	Use Cases	Mittel
Modulare und übersichtliche Software	R19	Design	Zielstellung	Hoch
Erweiterbarkeit der Software	R20	Design	Zielstellung	Hoch
Einfache Möglichkeit zur Auswechslung der verwendeten Bibliotheken	R21	Design	Zielstellung	Hoch
Implementierung des Proof-Of-Concept als Webdienst oder Handy-App	R22	Design	Zielstellung	Optional

*Tabelle 4.6: Tabelle der Anforderungen*

Insgesamt gibt es 13 Anforderungen mit der Priorität 'Hoch', 4 mit der Priorität 'Mittel' und 4 mit der Priorität 'Optional', was einen akkumulierten Gesamtwert von  $13 * 7 + 4 * 3 + 5 * 1 = 106$  ergibt (siehe Abbildung 4.2).

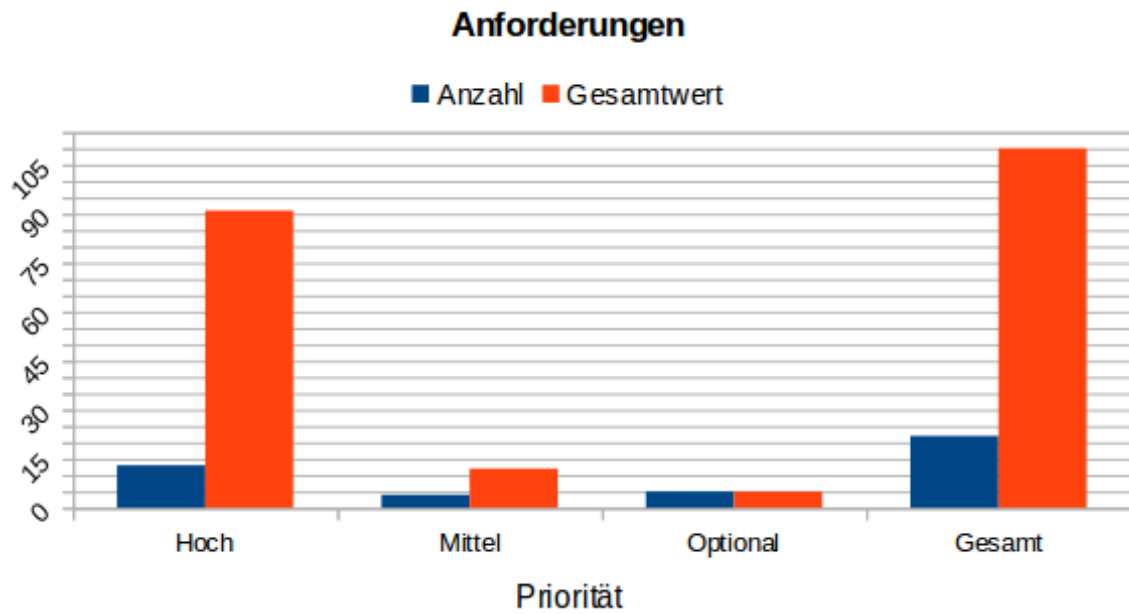


Abbildung 4.2: Säulendiagramm der Anzahl und Gesamtwerte der Anforderungen



## 4.2 Beschreibung der Lösungsansätze

In diesem Abschnitt werden die Ansätze zur Erfüllung beziehungsweise Lösung der Anforderungen aus Abschnitt 4.1 beschrieben.

### 4.2.1 Stabilitätstests von OCR-Ergebnissen

Um die Anforderung R1 (siehe Tabelle 4.6) 'Wie stabil ist die OCR-Erkennung?' zu beantworten, wurden mehrere Tests durchgeführt und evaluiert. Da es sich bei diesem Lösungsansatz um die Lösung einer wichtigen Anforderung und um ein umfangreiches Unterfangen handelt, wird die Durchführung des Ansatzes in Abschnitt 4.3 genauer erläutert.

### 4.2.2 Recherche über Möglichkeiten zur Verlinkung von Dokumenten

Um die Frage von Anforderung R2 (siehe Tabelle 4.6) 'Wie kann ein Dokument in einer DHT oder Blockchain referenziert werden?' zu beantworten, wurde im Internet recherchiert. Gesucht wurde nach DHT- und Blockchain-Anbietern, welche globalen und öffentlichen Zugang sowie die Möglichkeit zur Speicherung von (Text-)Dateien bieten. Ein wichtiges Kriterium ist außerdem, ob die Nutzung des Anbieters einen Account benötigt oder ob Daten ohne Account hoch- und runtergeladen werden können.

Die Fragen aus Anforderung R3 'Können Kopien von Dokumenten über die gleiche ID referenziert werden?' und R4 'Kann ein Dokument über seinen Content-Hash referenziert werden?' werden ebenfalls im Rahmen dieser Recherche beantwortet, indem die Funktionsweise der gefundenen Anbieter näher untersucht wird.

#### 4.2.2.1 Ermittelte Anbieter

Die in den folgenden Tabellen dargestellten Anbieter wurden bei der Recherche ermittelt. Jeder dieser Anbieter bietet die Möglichkeit Dateien zu speichern und durch einen globalen und öffentlichen Zugriff wieder herunterzuladen.

<b>Name</b>	<b>Dat Data [40]</b>
<b>Basiert auf</b>	DHT
<b>Beschreibung</b>	Peer-To-Peer Data-Sharing-Tool um effizient und sicher Daten zu teilen und zu synchronisieren. Die Daten werden zerlegt und im Netzwerk verteilt. Die Speicherung der Peer-Adressen basiert auf einer 'Kademlia Mainline' [41] DHT. Referenziert werden die Daten(-Chunks) durch Merkle-Tree-Hashes[38].
<b>Account benötigt?</b>	Nein
<b>Kostenlos?</b>	Ja

*Tabelle 4.7: Übersicht über Verlinkungsanbieter Dat Data*

<b>Name</b>	<b>Ethereum</b> [36]
<b>Basiert auf</b>	Blockchain
<b>Beschreibung</b>	Dezentrale Plattform um 'Smart-Contracts' auszuführen. 'Smart-Contracts' sind turing-vollständige Programme im Ethereum-Netzwerk. Über einen solchen 'Smart-Contract' können Variablen (zum Beispiel der Inhalt einer Textdatei) in einer Blockchain gespeichert werden. Die Kryptowährung [42] 'Ether' wird benötigt um 'Smart-Contracts' auszuführen.
<b>Account benötigt?</b>	Ja
<b>Kostenlos?</b>	Nein. (In privaten Testnetzwerken kostenlos.)

*Tabelle 4.8: Übersicht über Verlinkungsanbieter Ethereum*

<b>Name</b>	<b>IPFS</b> [43]
<b>Basiert auf</b>	Kombination aus Blockchain & DHT
<b>Beschreibung</b>	Peer-To-Peer Netzwerk, welches Dateien in 'Chunks' zerstückelt und dann auf die Knoten im Netzwerk verteilt. Peer-Adressen werden in 'Kademlia Mainline' [41] DHTs gespeichert, was eine effiziente Suche ermöglicht. Die Verlinkung der Daten basiert auf Merkle-Tree-Hashes, welche in einer Blockchain gespeichert werden. Dies garantiert die Integrität der Daten.
<b>Account benötigt?</b>	Nein
<b>Kostenlos?</b>	Ja

*Tabelle 4.9: Übersicht über Verlinkungsanbieter IPFS*

<b>Name</b>	<b>Storj</b> [44]
<b>Basiert auf</b>	Kombination aus Blockchain & DHT
<b>Beschreibung</b>	Peer-To-Peer Cloud-Storage Anbieter. Dateien werden erst verschlüsselt dann zerstückelt und anschließend auf die Netzwerkknoten verteilt. Gespeichert werden die Daten ebenfalls auf den Knoten im Netzwerk, welche mit einer 'Kademlia Mainline' [41] DHT adressiert werden. Zugriff auf die Daten hat nur der, der die Daten hochgeladen hat. Die Speicherung der Daten kostet Geld in Form einer Kryptowährung [42], die auf Blockchaintechnologie basiert.
<b>Account benötigt?</b>	Nein, Beta-Version: Ja
<b>Kostenlos?</b>	Ja, Beta-Version: Nein

*Tabelle 4.10: Übersicht über Verlinkungsanbieter Storj*

Beim Vergleich und der Bewertung der ermittelten Anbieter wurde IPFS als Gewinner gekrönt und für die Umsetzung in dieser Arbeit gewählt. Dies liegt vor allem daran, dass kein Account benötigt wird, die Nutzung kostenlos ist und zudem die Integrität der Dokumente durch die Speicherung deren 'Merkel-Tree-Hashes' in einer Blockchain garantiert ist. Bei dem Anbieter Dat Data ist das nicht der Fall, weshalb IPFS bevorzugt wurde. Ein weiterer Vorteil bei IPFS ist das Vorhandensein einer Schnittstelle in C++ [45], welche sehr gut dokumentiert ist.

Die Anbieter Ethereum und Storj wurden aussortiert, weil ein Account benötigt wird um sie zu Nutzen. Ein weiterer Nachteil ist die kostenpflichtige Nutzung.

### 4.2.3 Proof-Of-Concept Implementierung

Um das im Gesamtüberblick gezeigte Konzept umzusetzen, wurde ein Softwaresystem in der Programmiersprache C++ unter Ubuntu 16.04 LTS entwickelt. Da es sich um ein größeres Software Projekt handelt, wurde ein IDE-unabhängiges Build-Script, welches auf dem CMake-Build-System [46] basiert, kreiert.

In der aktuellen Version arbeitet das System, um die Signatur zu erzeugen und zu verifizieren, auf Basis des RSA-Kryptosystems und selbst signierten (self-signed) Zertifikaten nach dem X.509 Standard. Verlinkt werden die Dokumente beziehungsweise deren Verifikationsdaten in der IPFS-Blockchain.

Der Name des Systems lautet 'Printed Document Linking System' ('PDLS'). Es zählt über 100 Klassen in mehr als 4000 Zeilen C++ Code, weshalb das System in dieser Arbeit nur auf einem hohen Abstraktionsniveau erläutert wird.

Das 'PDLS' ist modular aufgebaut, das heißt, es besteht aus (insgesamt 10) Modulen, welche dedizierte Aufgaben übernehmen.

Die Architektur und der Ablauf der 'Proof-Of-Concept'-Implementierung wird im folgenden Abschnitt 4.5 genauer erläutert.

## 4.3 Tests zur Stabilität der OCR-Ergebnisse

In diesem Abschnitt wird auf Durchführung der in Abschnitt 4.2.1 angekündigten Tests zu Lösung der Anforderung R1 (siehe Tabelle 4.6), genauer eingegangen.

### 4.3.1 Beschreibung der allgemeinen Testumgebung

Alle Tests wurden durch ein C++ Programm, welches die OCR-Engine Tesseract in der Version 3.05 (siehe Abschnitt 4.5.1) nutzt, in den Standardeinstellungen durchgeführt.

Um die Übereinstimmung der OCR-Ergebnisse mit den Originaltexten zu berechnen wurde in allen Tests das Bash-Tool 'wdiff' mit Parameter '-s' genutzt. Dieses Tool vergleicht zwei Dokumente auf Wörterbasis miteinander und gibt, je nach Parameter, die Unterschiede aus. Der Parameter '-s' erzeugt eine Statistik.

Das Gerät, mit dem die Testdokumente gedruckt wurden, ist ein 'HP LaserJet 4100 Series PCL6' Drucker. Gedruckt wurde in schwarz-weiß mit den Standardeinstellungen des Geräts unter Windows 7 SP1.

Um die Scans der Testdokumente zu erzeugen wurde ein 'Kodak ScanMate i1150WN' Scanner verwendet. Sofern nicht anders angegeben wurde ebenfalls mit den Standardeinstellungen im Modus '300DPI - 8Bit Grayscale - Text Only' gescannt.

Die Nutzung der Bibliothek und den Geräten in den Standardeinstellungen wurde bewusst gewählt, da der Aufwand alle Einstellungen auszutesten und diese, sowie auch deren Zusammenhänge zu verstehen, immens ist und nicht in den zeitlichen als auch thematischen Rahmen dieser Arbeit passt. Dies führt allerdings auch dazu, dass die in diesem Abschnitt beschriebenen Tests keine allgemeingültigen Aussagen über die Stabilität der OCR-Ergebnisse bieten. Die Tests sollen lediglich dazu dienen, herauszufinden, ob es einer Optimierung der OCR-Erkennung bedarf oder ob die Standardeinstellungen bereits genügen, um ein ausreichendes Ergebnis zu bekommen.

## 4.3.2 OCR-Testphase 1

### 4.3.2.1 Beschreibung der OCR-Testphase 1 - Schriftarten

In dieser Testphase wurde getestet, welche Schriftarten (Fonts) sich gut oder schlecht für eine OCR-Analyse eignen. Dafür wurden einfache Testdokumente erstellt. Die Testdokumente variieren in den, in Tabelle 4.11 beschriebenen, Parametern.

<b>Schriftart</b>	Arial, Calibri, OpenSans, TimesNewRoman
<b>Schriftgröße</b>	9, 10, 11, 12, 14
<b>Schriftstil</b>	strong, normal

*Tabelle 4.11: Variationsparameter der einfachen Testdokumente aus OCR-Testphase 1*

Der (generische) Inhalt der Testdokumente ist wie folgt:

```
Hallo das ist ein Testdokument.  
Schriftart: [SCHRIFT_ART]  
Schriftgröße: [SCHRIFT_GROESSE] pt  
Schriftstyle: [SCHRIFT_STIL]
```

*Listing 4.1: Generischer Inhalt eines Testdokuments aus OCR-Testphase 1*

Wobei `[SCHRIFT_ART]`, `[SCHRIFT_GROESSE]` und `[SCHRIFT_STIL]` den für das jeweilige Testdokument aktuellen Parametern aus Tabelle 4.11 entsprechen. Die Parameter eines Dokuments spiegeln sich auch im Namen der jeweiligen Datei nach gleichem Schema wieder.

Aus allen möglichen Kombinationen der Parameter ergeben sich somit  $4 * 5 * 2 = 40$  verschiedene Testdokumente. Zudem wurde für jedes Testdokument der Originaltext in einer Plain-Text (\*.txt) Datei gespeichert.

Das für diesen Testfall erstellte Bash-Script speichert für jeden Scan der Testdokumente dessen OCR-Analyse ebenfalls in einer Plain-Text Datei und vergleicht daraufhin alle OCR-Ergebnisse mithilfe des 'wdiff' Bash-Tools mit dem jeweiligen Original. Die Vergleichsergebnisse werden in einer vom Script generierten CSV-Datei gespeichert. Der Inhalt dieser CSV-Datei und somit das Ergebnis dieses Tests ist im Abschnitt 5.1.1.1 und in Tabelle 4.13 zu sehen.

### 4.3.2.2 Ergebnisse der OCR-Testphase 1 - Schriftarten

Wie man in Tabelle 4.13 sehen kann, wurde bei 80% der Testdokumente eine fehlerfreie OCR-Erkennung erreicht.

In Tabelle 4.12 wird genauer auf die Fehler der Testdokumente mit fehlerbehafteten OCR-Ergebnissen eingegangen. Man kann aus der Tabelle herauslesen, dass die Schriftart 'Arial' mit Schriftstil 'Normal' am öftesten zu Fehlern führt. Die Schriftart 'Times New Roman' mit Schriftstil 'Strong' kann bei zu großer Schriftgröße zu Fehlern führen, bei der ein zusätzliches Leerzeichen zwischen zwei anderen Zeichen erkannt wird. Ist die Schriftgröße von 'Times New Roman' zu klein und im Schriftstil 'Normal', kann es zu Verwechslungen des Buchstaben 't' mit den Buchstaben 'l' oder 'i' kommen. Im Allgemeinen ist der häufigste Fehler ein zusätzliches Leerzeichen zwischen zwei Zeichen.

Testdokument	Fehler im Vergleich mit Originaltext	Fehler Beschreibung
Arial_10_Normal	Schriftgröße → Schrift-größe	extra: ‘-’
Arial_9_Normal		nicht erkannt: ‘:’
TimesNewRoman_11_Strong	11 → 1 1	extra: ‘ ’
TimesNewRoman_14_Strong	Schriftstyle → Sch riftstyle	extra: ‘ ’
Calibri_11_Normal	Schriftstyle → Sch riftstyle	extra: ‘ ’
OpenSans_10_Strong	Schriftstyle → Sch riftstyle	extra: ‘ ’
Arial_14_Normal	Schriftstyle → Sch riftstyle	extra: ‘ ’
TimesNewRoman_9_Normal	Schriftart → Schrifiart Schriftgröße → Schrifgröße Schriftstyle → Schrifstyle	verwechselt: ‘t’ mit ‘i’ oder ‘l’

*Tabelle 4.12: Fehler der OCR Erkennung aus OCR Testphase 1*

<b>Testdokument</b>	<b>OCR-Ergebnis Übereinstimmung mit Originaltext</b>
Arial_10_Strong	100.00%
Arial_11_Normal	100.00%
Arial_11_Strong	100.00%
Arial_12_Normal	100.00%
Arial_12_Strong	100.00%
Arial_14_Strong	100.00%
Arial_9_Strong	100.00%
Calibri_10_Normal	100.00%
Calibri_10_Strong	100.00%
Calibri_11_Strong	100.00%
Calibri_12_Normal	100.00%
Calibri_12_Strong	100.00%
Calibri_14_Normal	100.00%
Calibri_14_Strong	100.00%
Calibri_9_Normal	100.00%
Calibri_9_Strong	100.00%
OpenSans_10_Normal	100.00%
OpenSans_11_Normal	100.00%
OpenSans_11_Strong	100.00%
OpenSans_12_Normal	100.00%
OpenSans_12_Strong	100.00%
OpenSans_14_Normal	100.00%
OpenSans_14_Strong	100.00%
OpenSans_9_Normal	100.00%
OpenSans_9_Strong	100.00%
TimesNewRoman_10_Normal	100.00%
TimesNewRoman_10_Strong	100.00%
TimesNewRoman_11_Normal	100.00%
TimesNewRoman_12_Normal	100.00%
TimesNewRoman_12_Strong	100.00%
TimesNewRoman_14_Normal	100.00%
TimesNewRoman_9_Strong	100.00%
Arial_10_Normal	92.00%
Arial_9_Normal	92.00%
TimesNewRoman_11_Strong	87.00%
TimesNewRoman_14_Strong	87.00%
Calibri_11_Normal	86.00%
OpenSans_10_Strong	86.00%
Arial_14_Normal	85.00%
TimesNewRoman_9_Normal	79.00%

*Tabelle 4.13: OCR-Ergebnistabelle der Scans aus OCR-Testphase 1*

### 4.3.3 OCR-Testphase 2

#### 4.3.3.1 Beschreibung der OCR-Testphase 2 - Tabulator Notenspiegel

Ziel dieser Testphase war es, herauszufinden, wie gut die OCR-Erkennung bei komplexeren Testdokumenten abschneidet. Als Testdokumente wurden in Anlehnung an die Anwendungsfälle UC-1, UC-2 und UC-3 (siehe Abschnitt 4.1.1), einfache Notenspiegel-Testdokumente erstellt. Der textuelle Inhalt (siehe Listing 4.2) dieser Dokumente ist gleich, es wurden jedoch, wie auch bei OCR-Testphase 1, Parameter variiert (siehe Tabelle 4.14).

Aus allen Kombinationen der Variationsparameter ergeben sich somit  $4 \cdot 5 \cdot 2 \cdot 2 = 80$  Notenspiegel-Testdokumente. Diese wurden alle, wie in Abschnitt 4.3.1 beschrieben, ausgedruckt. Dann wurden die Dokumente einmal mit 300 DPI und einmal 600 DPI vom Scanner eingelesen.

<b>Schriftart</b>	Arial, Calibri, OpenSans, TimesNewRoman
<b>Schriftgröße</b>	9, 10, 11, 12, 14
<b>Schriftstil</b>	strong, normal
<b>Scann DPI</b>	300, 600

*Tabelle 4.14: Variationsparameter der Notenspiegel-Testdokumente aus OCR-Testphase 2 und 3*

Für diese Testphase wurde eine Minimalversion des im Rahmen dieser Arbeit erstellten Proof-Of-Concept Programms zur OCR-Analyse verwendet, welches den Inhalt der Dokumente extrahiert, anschließend normiert und als Plain-Text (\*.txt) Datei gespeichert. Der Originaltext aus Listing 4.2 wurde ebenfalls normiert und in einer Plain-Text Datei auf der Festplatte abgelegt. Das für diesen Testfall erstellte Bash-Script vergleicht, analog zur OCR-Testphase 1 mithilfe des 'wdiff'-Kommando, den normierten Originaltext mit den normierten OCR-Ergebnissen und speichert das Resultat in einer CSV-Datei. Das Ergebnis des Tests (also der Inhalt der CSV-Datei) ist in Abschnitt 5.1.1.1, Tabelle 4.15 und Tabelle 4.16 sowie den Abbildungen 4.3 und 4.3 einzusehen.



Notenspiegel	
Florian Schneider Hochschule Ulm Technische Informatik Semester 7	
Mathematik 1	1,0
Programmieren 3	1,3
Betriebssysteme	1,7
Verteilte Systeme	2,0
Programmieren 3	2,3
Elektrotechnik	2,7
Kryptologie	3,0
Webengineering	3,3
Datenbanken	3,7
Englisch	4,0
Software Technik	4,3
Embedded Systems	4,7
Digital Technik	5,0
Mikro Computer	5,3
Netzwerk Technik	5,7
Theoretische Informatik	6,0

*Listing 4.2: Inhalt eines Notenspiegel-Testdokuments aus OCR-Testphase 2 und 4*

#### 4.3.3.2 Ergebnisse der OCR-Testphase 2 - Tabulator Notenspiegel

Wie man in den Tabellen 4.15 und 4.16 sowie den Diagrammen in Abbildung 4.3 und Abbildung 4.4 sehen kann, gibt es kein einziges der insgesamt 80 Notenspiegel-Testdokumente aus dieser Testphase, dessen OCR-Ergebnis 80% oder mehr Übereinstimmung mit dem Originaltext hat. Da es sich um eine große Anzahl an fehlerbehafteten Testdokumenten handelt, wird an dieser Stelle nicht auf die einzelnen Fehler eingegangen, sondern nur die Fehlerarten beschrieben.

Die am häufigsten aufgetretene Fehlerart ist die falsche Reihenfolge der erkannten Wörter. Die OCR-Engine hat in allen Fällen, welche über 60% Erkennungsrate liegen, zuerst die linke Spalte des Testdokuments erkannt und anschließend die rechte Spalte (siehe Listing 4.2). Außerdem wurde die in der Mitte des Dokuments stehende Überschrift nicht als erstes, sondern erst nach dem Anschriftsblock erkannt. Die Wörter selbst wurden (bis auf einzelne Ausnahmen) richtig erkannt.

Die Fehlerart bei den falsch erkannten Wörtern ähnelt der OCR-Testphase 1. Es wurden zusätzliche Leerzeichen zwischen Buchstaben erkannt und/oder Buchstaben mit anderen Buchstaben verwechselt.

Was man beim Vergleich der Abbildungen 4.3 und 4.4 sehen kann, ist, dass es bei den Scans mit 600 DPI mehr OCR-Ergebnisse mit Erkennungsraten zwischen 70% - 79% gibt als bei den Scans mit 300 DPI. Allerdings wurden bei den 600 DPI Scans weniger OCR-Ergebnisse zwischen 60% - 69% erreicht. Außerdem gibt es einen Ausreißer mit einer Erkennungsrate zwischen 20% - 29% und einen zwischen 50% - 59%.

Bei den beiden Ausreißer-Testdokumenten 'Calibri\_10\_S\_300\_DPI' und 'Calibri\_10\_S\_600\_DPI' wurde aus bisher unerklärlichen Gründen (auch bei erneuten einzelnen Tests der Dokumente) festgestellt, dass die OCR-Engine in der Mitte des Dokuments willkürliche Buchstaben anstatt der korrekten Buchstaben erkannt hat (sehr ähnlich zum Beispiel aus der nächsten Testphase in Abbildung 4.7 und Abbildung 4.8). Sehr merkwürdig ist hier vor allem, dass es nicht die kleinste Schriftgröße der Dokumente mit Schriftart 'Calibri' und Schriftstil 'Strong' ist und, dass diese Dokumente ('Calibri\_9\_S\_300\_DPI' und 'Calibri\_9\_S\_300\_DPI') deutlich besser abgeschnitten haben. Dies macht es sehr unwahrscheinlich, dass die Fehler mit der Schriftart beziehungsweise Schriftgröße in Zusammenhang stehen. Eine mögliche Erklärung wäre, dass der Scanner gerade bei diesen Testdokumenten ein zufälliges Rauschen erzeugt hat, welches bei Tesseract zu Problemen führt. Diese Möglichkeit ist allerdings ebenfalls sehr unwahrscheinlich, weil es sich um zwei unterschiedliche Scans handelt und der Scanner dann zweimal, zufällig bei diesen Dokumenten das Rauschen erzeugt haben müsste.

<b>Testdokument</b>	<b>OCR-Ergebnis Übereinstimmung mit Originaltext</b>
Arial_11_N_300_DPI	71.00%
Arial_12_N_300_DPI	71.00%
Arial_14_N_300_DPI	71.00%
Calibri_12_N_300_DPI	71.00%
Calibri_12_S_300_DPI	71.00%
Calibri_14_N_300_DPI	71.00%
Calibri_14_S_300_DPI	71.00%
OpenSans_11_N_300_DPI	71.00%
OpenSans_11_S_300_DPI	71.00%
OpenSans_12_N_300_DPI	71.00%
OpenSans_14_N_300_DPI	71.00%
TimesNewRoman_11_S_300_DPI	71.00%
TimesNewRoman_12_S_300_DPI	71.00%
TimesNewRoman_14_N_300_DPI	71.00%
Arial_10_N_300_DPI	69.00%
Arial_12_S_300_DPI	69.00%
Arial_9_S_300_DPI	69.00%
Calibri_10_N_300_DPI	69.00%
Calibri_11_N_300_DPI	69.00%
Calibri_11_S_300_DPI	69.00%
Calibri_9_N_300_DPI	69.00%
Calibri_9_S_300_DPI	69.00%
OpenSans_10_N_300_DPI	69.00%
OpenSans_14_S_300_DPI	69.00%
OpenSans_9_N_300_DPI	69.00%
OpenSans_9_S_300_DPI	69.00%
TimesNewRoman_12_N_300_DPI	69.00%
TimesNewRoman_14_S_300_DPI	69.00%
Arial_11_S_300_DPI	67.00%
Arial_14_S_300_DPI	67.00%
Arial_9_N_300_DPI	67.00%
OpenSans_10_S_300_DPI	67.00%
OpenSans_12_S_300_DPI	67.00%
TimesNewRoman_10_S_300_DPI	67.00%
Arial_10_S_300_DPI	65.00%
TimesNewRoman_10_N_300_DPI	65.00%
TimesNewRoman_11_N_300_DPI	65.00%
TimesNewRoman_9_N_300_DPI	65.00%
TimesNewRoman_9_S_300_DPI	63.00%
Calibri_10_S_300_DPI	16.00%

*Tabelle 4.15: OCR-Ergebnistabelle der Scans mit 300 DPI aus OCR-Testphase 2*

<b>Testdokument</b>	<b>OCR-Ergebnis Übereinstimmung mit Originaltext</b>
Arial_11_N_600_DPI	71%
Arial_14_N_600_DPI	71%
Calibri_12_N_600_DPI	71%
Calibri_12_S_600_DPI	71%
Calibri_14_N_600_DPI	71%
Calibri_14_S_600_DPI	71%
OpenSans_11_N_600_DPI	71%
OpenSans_11_S_600_DPI	71%
OpenSans_12_N_600_DPI	71%
OpenSans_12_S_600_DPI	71%
OpenSans_14_N_600_DPI	71%
TimesNewRoman_11_S_600_DPI	71%
TimesNewRoman_12_N_600_DPI	71%
TimesNewRoman_12_S_600_DPI	71%
TimesNewRoman_14_N_600_DPI	71%
TimesNewRoman_14_S_600_DPI	71%
Arial_10_N_600_DPI	69%
Arial_12_N_600_DPI	69%
Calibri_11_N_600_DPI	69%
Calibri_11_S_600_DPI	69%
Calibri_9_N_600_DPI	69%
Calibri_9_S_600_DPI	69%
OpenSans_14_S_600_DPI	69%
OpenSans_9_N_600_DPI	69%
OpenSans_9_S_600_DPI	69%
OpenSans_10_S_600_DPI	68%
Arial_10_S_600_DPI	67%
Arial_11_S_600_DPI	67%
Arial_12_S_600_DPI	67%
Arial_9_N_600_DPI	67%
Arial_9_S_600_DPI	67%
OpenSans_10_N_600_DPI	67%
TimesNewRoman_10_S_600_DPI	67%
TimesNewRoman_9_N_600_DPI	67%
Arial_14_S_600_DPI	65%
TimesNewRoman_10_N_600_DPI	65%
TimesNewRoman_11_N_600_DPI	65%
TimesNewRoman_9_S_600_DPI	61%
Calibri_10_N_600_DPI	51%
Calibri_10_S_600_DPI	25%

*Tabelle 4.16: OCR-Ergebnistabelle der Scans mit 600 DPI aus OCR-Testphase 2*

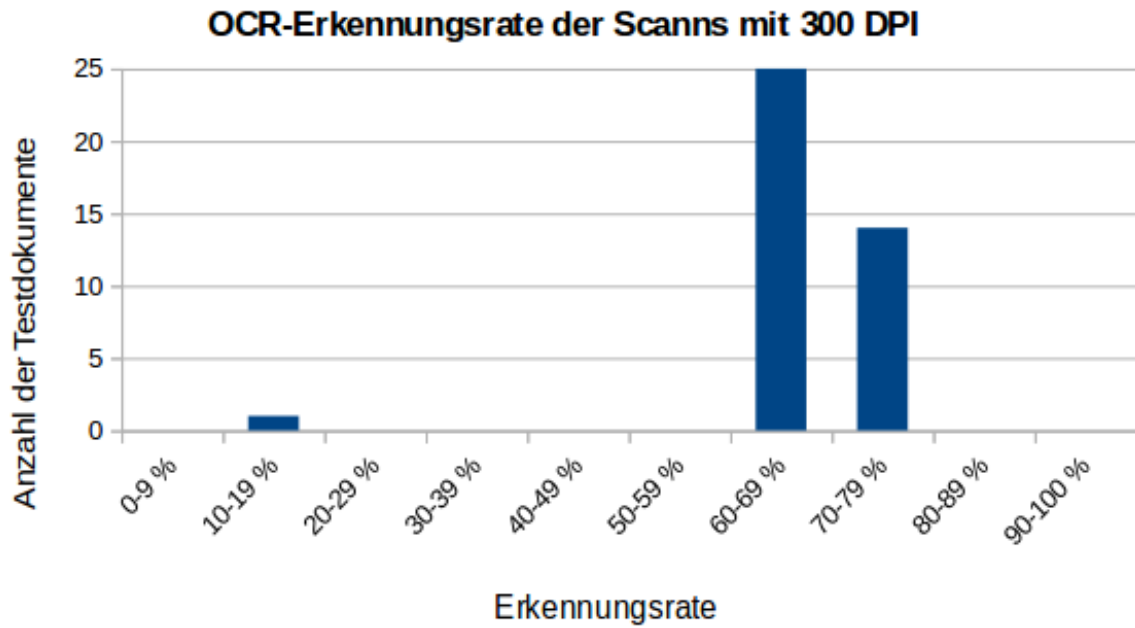


Abbildung 4.3: Säulendiagramm der OCR-Erkennungsrate der Scans aus OCR-Testphase 2 mit 300 DPI

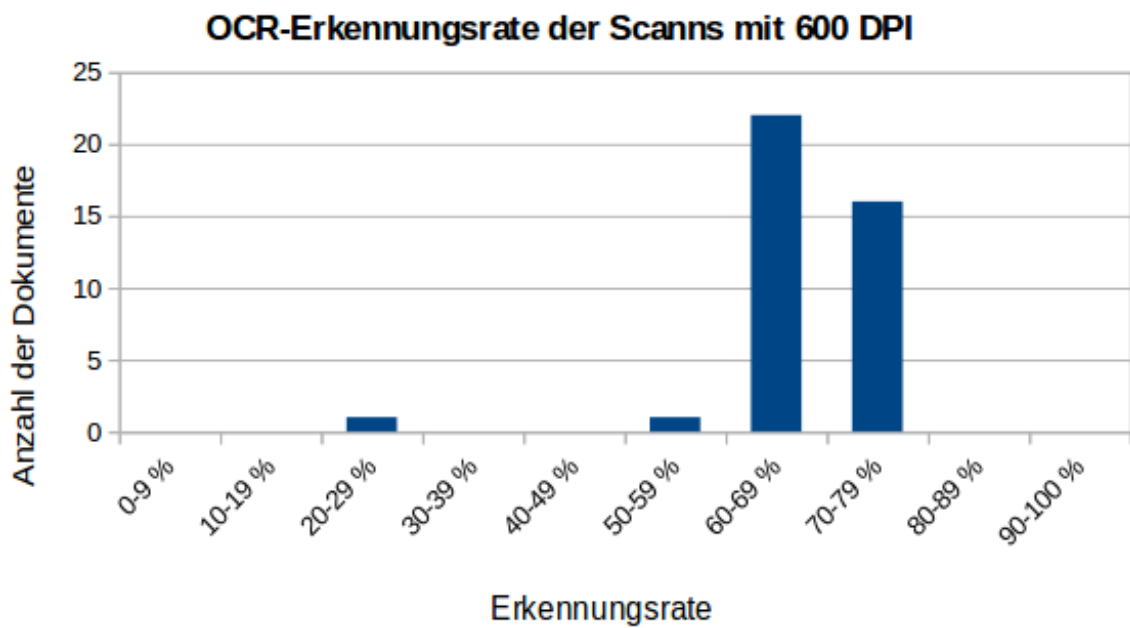


Abbildung 4.4: Säulendiagramm der OCR-Erkennungsrate der Scans aus OCR-Testphase 2 mit 600 DPI

## 4.3.4 OCR-Testphase 3

### 4.3.4.1 OCR-Testphase 3 - Tabellen Notenspiegel

Da die Ergebnisse aus OCR-Testphase 2 (Abschnitt 4.3.3.1) nicht zufriedenstellend ausfielen, wurde eine weitere Testphase eingeleitet, welche prüfen sollte ob Notenspiegel mit Tabellen zu besseren OCR-Ergebnissen führen. Der Inhalt dieser Notenspiegel-Testdokumente ist in Listing 4.3 zu sehen, wobei die Tabellenstruktur aus den Zeichen '-', '|' und '+', nicht als Zeichenkette verstanden werden darf, sondern nur zur Veranschaulichung dienen soll.

Der Test an sich verlief exakt analog zum Test aus OCR-Testphase 3, weshalb hier nicht mehr auf den Ablauf eingegangen wird. Das Ergebnis des Tests ist in Abschnitt 5.1.1.1, Tabelle 4.17 und Tabelle 4.18 sowie den Abbildungen 4.5 und 4.6 einzusehen.

Notenspiegel		
Florian Schneider Hochschule Ulm Technische Informatik Semester 7		
Mathematik 1	1,0	
Programmieren 3	1,3	
Betriebssysteme	1,7	
Verteilte Systeme	2,0	
Programmieren 3	2,3	
Elektrotechnik	2,7	
Kryptologie	3,0	
Webengineering	3,3	
Datenbanken	3,7	
Englisch	4,0	
Software Technik	4,3	
Embedded Systems	4,7	
Digital Technik	5,0	
Mikro Computer	5,3	
Netzwerk Technik	5,7	
Theoretische Informatik	6,0	

*Listing 4.3: Inhalt eines Notenspiegel-Testdokuments aus OCR-Testphase 3 und 4*

#### 4.3.4.2 Ergebnisse der OCR-Testphase 3 - Tabellen Notenspiegel

Aus den Tabellen 4.17 und 4.18 sowie den Diagrammen in Abbildung 4.5 und Abbildung 4.6 lässt sich (wie auch bei OCR-Testphase 2) erkennen, dass es wieder keine zufriedenstellenden Ergebnisse aus dieser Testphase gibt.

Wie auch bei der vorherigen Testphase wird aufgrund der großen Anzahl an Test-

dokumenten nicht auf die einzelnen Fehler eingegangen, sondern nur auf die Fehlerklassen der am häufigsten aufgetretenen Fehler.

Ähnlich wie in OCR-Testphase 2 wurde bei allen Dokumenten Fehler bei der Erkennungsreihenfolge gemacht. In den Fällen einer Erkennungsrate zwischen 50%-79% wurde die linke Spalte der Tabelle als erste ausgewertet und danach die rechte Spalte (siehe Listing 4.3). Wie in Testphase 2 wurde oft die Überschrift nicht als erstes, sondern nach dem Anschriftsblock erkannt. Es hat also keinen Vorteil, eine Tabelle zu nutzen.

Des Weiteren wurden ebenfalls wie in OCR-Testphase 2 einzelne Zeichen verwechselt. Dabei sind vor allem die Zeichen 'i', 'l', 't', '1', ',', ';', 'm', 'u' und 'I' betroffen. Bei den OCR-Ergebnissen mit unter 50% Erkennungsrate gab es, wie auch in der vorherigen Testphase, eine scheinbar willkürliche, eventuell durch Rauschen verursachte Verwechslung von Zeichen. Dieses Verhalten wurde näher untersucht indem das Dokument mehrfach einzeln getestet wurde. Sowohl mit dem für die Testphasen erstellen Programm, als auch mit dem CLI von Tesseract selbst. In Abbildung 4.7 ist ein Ausschnitt des Scans zusehen, welcher die Ausgabe aus Abbildung 4.8 erzeugt. Zum Zeitpunkt des Verfassens dieses Abschnitts konnte keine Ursache für diese Art von Problem eindeutig festgemacht werden.



<b>Testdokument</b>	<b>OCR-Ergebnis Übereinstimmung mit Originaltext</b>
Arial_10_S_300_DPI	71%
Arial_11_N_300_DPI	71%
Arial_12_N_300_DPI	71%
Arial_14_N_300_DPI	71%
Arial_14_S_300_DPI	71%
Calibri_11_S_300_DPI	71%
Calibri_12_N_300_DPI	71%
Calibri_12_S_300_DPI	71%
Calibri_14_N_300_DPI	71%
Calibri_14_S_300_DPI	71%
OpenSans_10_N_300_DPI	71%
OpenSans_10_S_300_DPI	71%
OpenSans_11_S_300_DPI	71%
OpenSans_12_N_300_DPI	71%
OpenSans_12_S_300_DPI	71%
OpenSans_14_N_300_DPI	71%
OpenSans_14_S_300_DPI	71%
TimesNewRoman_11_S_300_DPI	71%
TimesNewRoman_12_S_300_DPI	71%
TimesNewRoman_14_N_300_DPI	71%
TimesNewRoman_14_S_300_DPI	71%
Arial_11_S_300_DPI	69%
Arial_12_S_300_DPI	69%
Arial_9_N_300_DPI	69%
Calibri_9_N_300_DPI	69%
Calibri_9_S_300_DPI	69%
OpenSans_9_N_300_DPI	69%
Calibri_10_N_300_DPI	67%
Calibri_11_N_300_DPI	67%
OpenSans_9_S_300_DPI	67%
TimesNewRoman_10_S_300_DPI	65%
TimesNewRoman_9_S_300_DPI	65%
TimesNewRoman_10_N_300_DPI	63%
OpenSans_11_N_300_DPI	61%
TimesNewRoman_12_N_300_DPI	61%
TimesNewRoman_9_N_300_DPI	51%
Arial_10_N_300_DPI	43%
Calibri_10_S_300_DPI	33%
Arial_9_S_300_DPI	28%
TimesNewRoman_11_N_300_DPI	23%

*Tabelle 4.17: OCR-Ergebnistabelle der Scans mit 300 DPI aus OCR-Testphase 3*

<b>Testdokument</b>	<b>OCR-Ergebnis Übereinstimmung mit Originaltext</b>
TimesNewRoman_12_N_600_DPI	72%
Arial_10_S_600_DPI	71%
Calibri_12_S_600_DPI	71%
Calibri_14_N_600_DPI	71%
Calibri_14_S_600_DPI	71%
OpenSans_11_S_600_DPI	71%
OpenSans_14_N_600_DPI	71%
TimesNewRoman_12_S_600_DPI	71%
TimesNewRoman_14_S_600_DPI	71%
TimesNewRoman_9_N_600_DPI	70%
TimesNewRoman_9_S_600_DPI	70%
Arial_12_N_600_DPI	69%
Arial_12_S_600_DPI	69%
Arial_14_N_600_DPI	69%
Arial_14_S_600_DPI	69%
Calibri_10_N_600_DPI	69%
Calibri_11_N_600_DPI	69%
OpenSans_10_N_600_DPI	69%
OpenSans_14_S_600_DPI	69%
TimesNewRoman_10_N_600_DPI	68%
Arial_11_S_600_DPI	67%
OpenSans_12_S_600_DPI	67%
OpenSans_9_N_600_DPI	67%
Calibri_9_N_600_DPI	65%
Calibri_9_S_600_DPI	65%
OpenSans_10_S_600_DPI	61%
TimesNewRoman_11_N_600_DPI	56%
Arial_11_N_600_DPI	55%
Calibri_11_S_600_DPI	51%
TimesNewRoman_14_N_600_DPI	51%
Calibri_10_S_600_DPI	46%
Arial_9_S_600_DPI	38%
OpenSans_12_N_600_DPI	38%
TimesNewRoman_10_S_600_DPI	36%
Arial_9_N_600_DPI	34%
OpenSans_11_N_600_DPI	31%
Arial_10_N_600_DPI	29%
Calibri_12_N_600_DPI	29%
OpenSans_9_S_600_DPI	24%
TimesNewRoman_11_S_600_DPI	8%

*Tabelle 4.18: OCR-Ergebnistabelle der Scans mit 600 DPI aus OCR-Testphase 3*

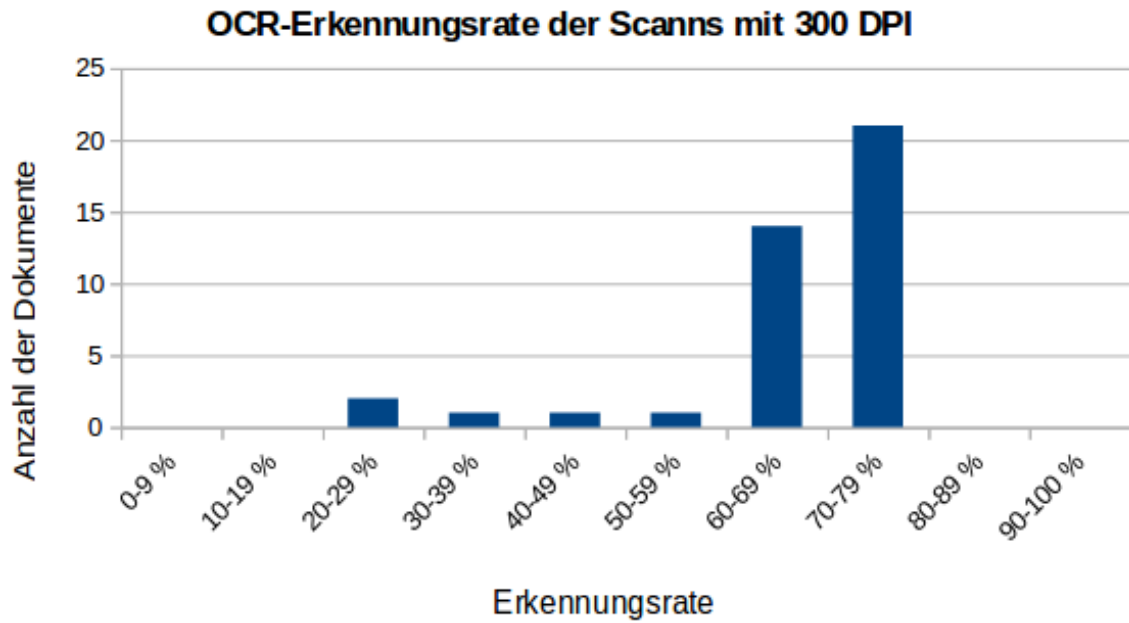


Abbildung 4.5: Säulendiagramm der OCR-Erkennungsrate der Scans aus OCR-Testphase 3 mit 300 DPI

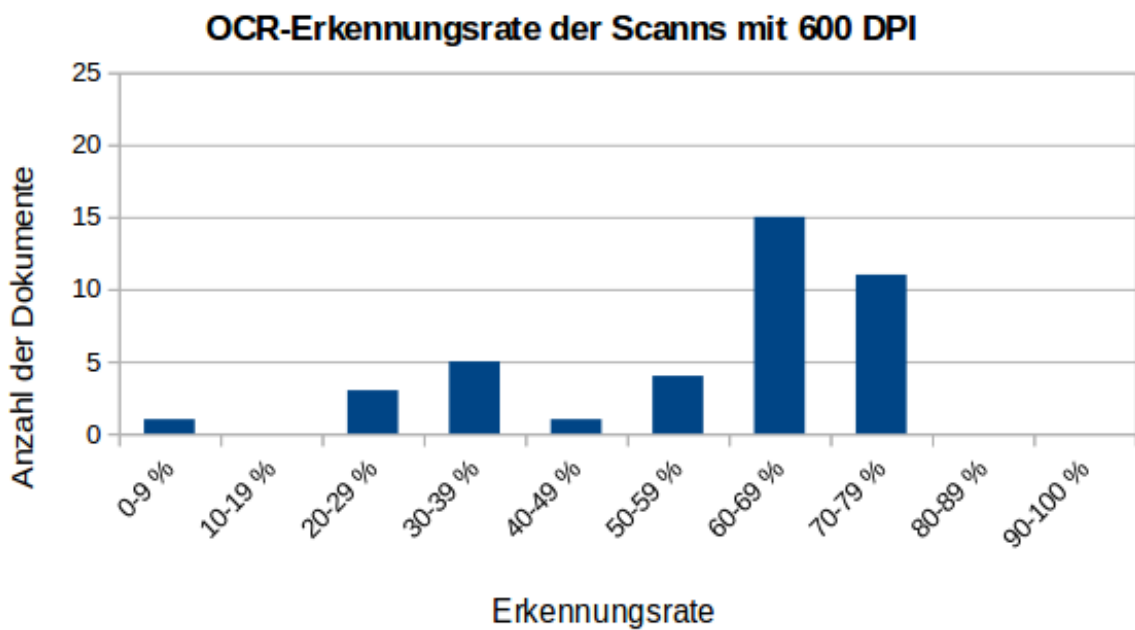


Abbildung 4.6: Säulendiagramm der OCR-Erkennungsrate der Scans aus OCR-Testphase 3 mit 600 DPI

Kryptologie	3,0
Webengineering	3,3
Datenbanken	3,7
Englisch	4,0
Software Technik	4,3
Embedded Systems	4,7
Digital Technik	5,0
Mikro Computer	5,3
Netzwerk Technik	5,7
Theoretische Informatik	6,0

Abbildung 4.7: Ausschnitt des Testdokuments 'Calibri\_12\_N\_600\_DPI' aus OCR-Testphase 3

```
Kryptologie W WWW W W W WWW W3,0
WWéberigWiHeWérthgW WW W W W 3,2;
WbWawténbaakéhWW W W WWW W W W W W W W W35
W WEWntIisch W W W 4,0
WswaEQJAewTWéEEnWakW W W W W W WW W 5
WériinéAAQsEWswyWswtEEAsW W W W W WW 2,7
WWbiglia'leée'fiWhik' W WWWWWWWWWWWWWWWWWWWWWWW WW WW W WWWstW}
MlérWIawcWérhbWJté? W W W Wsw,W3W5
WNetWwszerkTeer1WrWiWinW< WWW 5,7'
theékéflééíéWmWforWrflééíkW W W W W WWWWéWbW:
```

Abbildung 4.8: OCR-Ergebnis der Tesseract CLI des Dokumentenausschnitts aus Abbildung 4.7

### **4.3.5 OCR-Testphase 4**

#### **4.3.5.1 OCR-Testphase 4 - Notenspiegel Original-PDFs**

Diese Testphase wurde eingeschoben, weil die Ergebnisse der vorherigen Tests nicht zufriedenstellend waren. Bei diesem Test sollte geprüft werden, wie negativ sich das Scannen der Testdokumente auf deren OCR-Ergebnisse auswirkt beziehungsweise, ob es einen Unterschied macht, wenn der OCR-Algorithmus auf einer rein digitalen PDF (siehe Abbildung 4.11) arbeitet.

Bei dem Test dieser Phase wurden die originalen PDFs, also die Dateien, welche in den OCR-Testphasen 2 und 3 ausgedruckt wurden, verwendet. Ansonsten ist der Ablauf wieder analog zu Phase 2 und 3. Das Ergebnis wird in Abschnitt 5.1.1.1, Tabelle 5.13 und Tabelle 4.19 sowie den Abbildungen 4.10 und 4.9 erläutert.

#### **4.3.5.2 Ergebnisse der OCR-Testphase 4 - Notenspiegel Original-PDFs**

Wie man in den Tabellen 4.19 und 5.13 sowie den Diagrammen in Abbildung 4.9 und Abbildung 4.10 sehen kann, sind die Erkennungsraten bei den rein digitalen PDFs deutlich besser als bei den Scans.

Die am häufigsten vorkommenden Fehler sind Fehler in der Reihenfolge der erkannten Wörter. Bei den Notenspiegeln mit Tabellen wurde wieder (wie in den vorherigen Testphasen) die Überschrift nach dem Block der Anschrift oder vor Beginn der rechten Spalte erkannt. Eine falsche Reihenfolge der Noten ist bei den Notenspiegeln mit Tabellen nicht aufgetreten.

Bei den Notenspiegeln mit Tabulator wurde die linke Spalte zuerst und daraufhin die rechte Spalte erkannt.

Die Verwechslung von Zeichen ist eher selten, tritt jedoch gerade bei der Schriftart 'Times New Roman' häufig auf. Das am häufigsten verwechselte oder nicht erkannte Zeichen ist das Komma (',').

Der Fehler bei dem scheinbar willkürlich Zeichen falsch erkannt wurden, trat in diesem Test nicht auf. Wie bereits in OCR-Testphase 2 und 3 vermutet, verstärkt sich der Verdacht, dass der Fehler durch ein Rauschen in den Scans verursacht wird.

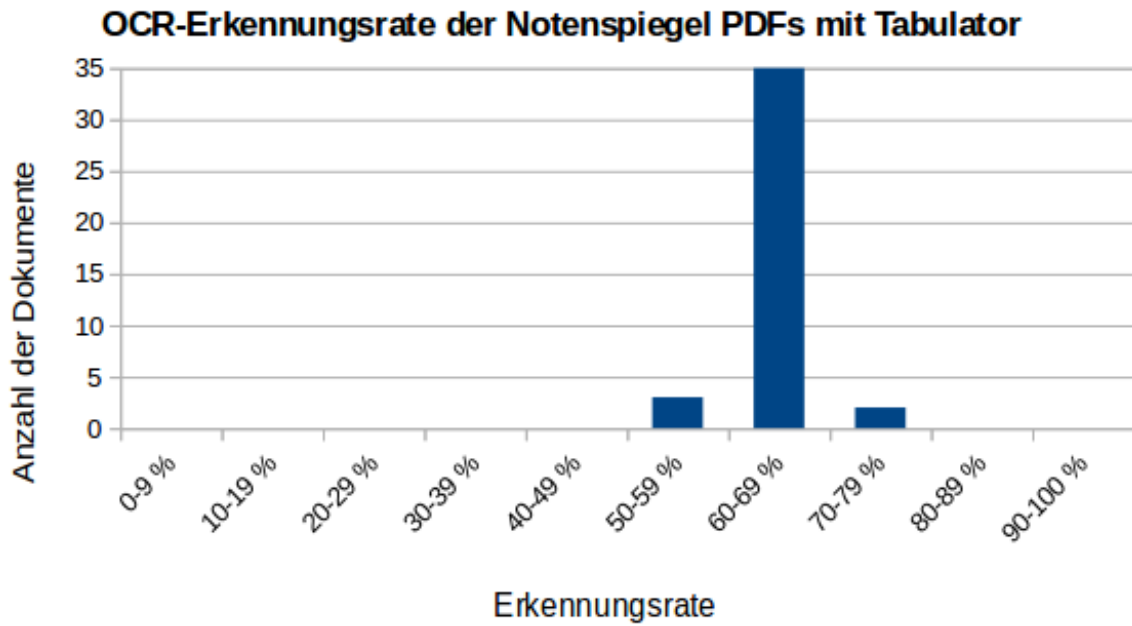


Abbildung 4.9: Säulendiagramm der OCR-Erkennungsrate der Notenspiegel-Testdokumente mit Tabulator aus OCR-Testphase 4

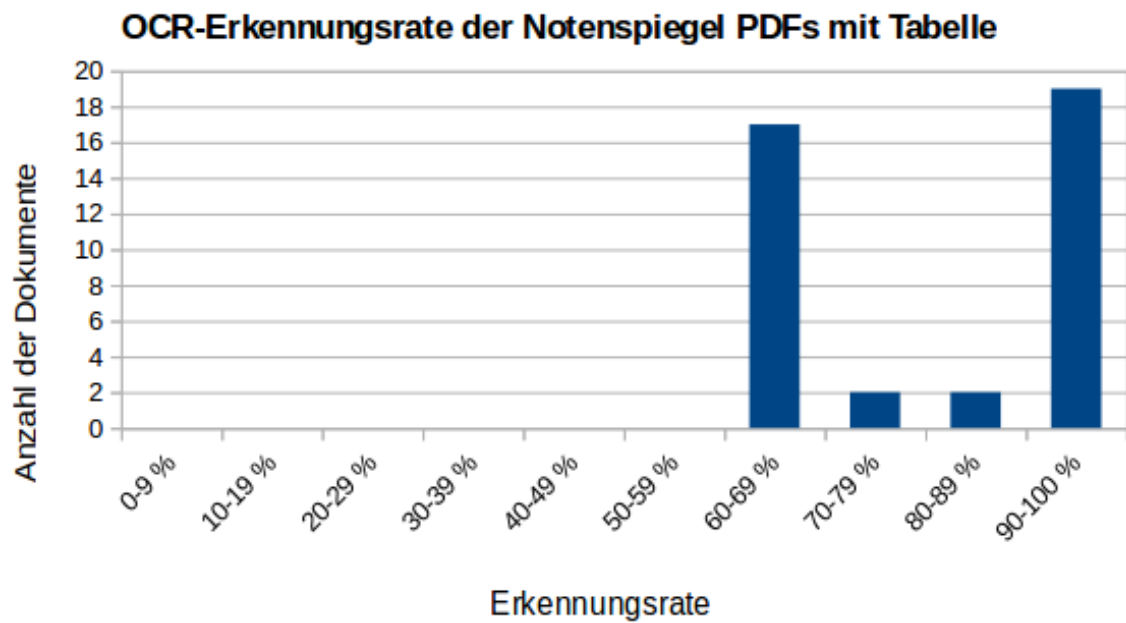


Abbildung 4.10: Säulendiagramm der OCR-Erkennungsrate der Notenspiegel-Testdokumente mit Tabelle aus OCR-Testphase 4

<b>Testdokument</b>	<b>OCR-Ergebnis Übereinstimmung mit Originaltext</b>
Calibri_14_N_Tabulator	71.00%
OpenSans_14_N_Tabulator	71.00%
Arial_11_N_Tabulator	69.00%
Arial_12_S_Tabulator	69.00%
Arial_14_S_Tabulator	69.00%
Calibri_10_N_Tabulator	69.00%
Calibri_11_N_Tabulator	69.00%
Calibri_11_S_Tabulator	69.00%
Calibri_12_N_Tabulator	69.00%
Calibri_12_S_Tabulator	69.00%
Calibri_9_S_Tabulator	69.00%
OpenSans_10_N_Tabulator	69.00%
OpenSans_10_S_Tabulator	69.00%
OpenSans_11_N_Tabulator	69.00%
OpenSans_12_S_Tabulator	69.00%
OpenSans_14_S_Tabulator	69.00%
OpenSans_9_N_Tabulator	69.00%
OpenSans_9_S_Tabulator	69.00%
TimesNewRoman_10_S_Tabulator	69.00%
TimesNewRoman_11_S_Tabulator	69.00%
TimesNewRoman_12_S_Tabulator	69.00%
TimesNewRoman_14_S_Tabulator	69.00%
TimesNewRoman_9_S_Tabulator	69.00%
Arial_10_N_Tabulator	67.00%
Arial_10_S_Tabulator	67.00%
Arial_14_N_Tabulator	67.00%
Arial_9_N_Tabulator	67.00%
Calibri_10_S_Tabulator	67.00%
Calibri_9_N_Tabulator	67.00%
OpenSans_12_N_Tabulator	67.00%
Arial_11_S_Tabulator	65.00%
Arial_12_N_Tabulator	65.00%
Arial_9_S_Tabulator	65.00%
Calibri_14_S_Tabulator	65.00%
OpenSans_11_S_Tabulator	65.00%
TimesNewRoman_9_N_Tabulator	65.00%
TimesNewRoman_12_N_Tabulator	60.00%
TimesNewRoman_11_N_Tabulator	58.00%
TimesNewRoman_14_N_Tabulator	57.00%
TimesNewRoman_10_N_Tabulator	52.00%

*Tabelle 4.19: OCR-Ergebnistabelle der Notenspiegel-Testdokumente mit Tabulator aus OCR-Testphase 4*

<b>Testdokument</b>	<b>OCR-Ergebnis Übereinstimmung mit Originaltext</b>
Arial_11_N_Table	98.00%
Arial_12_S_Table	98.00%
Calibri_10_S_Table	98.00%
Calibri_12_N_Table	98.00%
Calibri_14_N_Table	98.00%
OpenSans_11_N_Table	98.00%
OpenSans_11_S_Table	98.00%
OpenSans_12_N_Table	98.00%
OpenSans_14_N_Table	98.00%
OpenSans_14_S_Table	98.00%
TimesNewRoman_10_S_Table	98.00%
Calibri_9_S_Table	96.00%
Arial_14_N_Table	94.00%
Arial_9_N_Table	94.00%
Arial_9_S_Table	94.00%
Calibri_11_N_Table	94.00%
Calibri_9_N_Table	94.00%
TimesNewRoman_12_S_Table	92.00%
TimesNewRoman_12_N_Table	91.00%
Arial_12_N_Table	89.00%
TimesNewRoman_14_N_Table	85.00%
Arial_10_N_Table	71.00%
OpenSans_10_S_Table	71.00%
Arial_10_S_Table	69.00%
Arial_11_S_Table	69.00%
Calibri_10_N_Table	69.00%
Calibri_12_S_Table	69.00%
Calibri_14_S_Table	69.00%
OpenSans_10_N_Table	69.00%
OpenSans_12_S_Table	69.00%
OpenSans_9_N_Table	69.00%
OpenSans_9_S_Table	69.00%
TimesNewRoman_10_N_Table	69.00%
TimesNewRoman_11_N_Table	69.00%
TimesNewRoman_11_S_Table	69.00%
TimesNewRoman_14_S_Table	69.00%
TimesNewRoman_9_N_Table	69.00%
Arial_14_S_Table	67.00%
Calibri_11_S_Table	67.00%
TimesNewRoman_9_S_Table	67.00%

*Tabelle 4.20: OCR-Ergebnistabelle der Notenspiegel-Testdokumente mit Tabelle aus OCR-Testphase 4*



### 4.3.6 Mögliche Ursachen der schlechten Testergebnisse

In diesem Abschnitt wird kurz auf die (erkannten) möglichen Ursachen der schlechten Ergebnisse aus den OCR-Testphasen eingegangen.

Als aller erstes muss darauf hingewiesen werden, dass das Tool 'wdiff', welches verwendet wurde, um ein OCR-Ergebnis mit dem Originaltext zu vergleichen, einen wörterbasierten Vergleich macht. Das bedeutet, dass ganze Wörter und nicht einzelne Zeichen verglichen werden. Das heißt, wenn ein Zeichen in einem Wort falsch ist, wird das ganze Wort als falsch gerechnet. Dies führt dazu, dass die Tests negativer ausfallen, als wenn man einen zeichenbasierten Vergleich machen würde. Da jedoch selbst ein einziges falsches Zeichen zu einem komplett veränderten Content-Hash führt, spielt dieser Grund an sich keine Rolle und ist auch kein wirklicher Grund, da es sich um ein Darstellungsproblem handelt.

Ein tatsächliches Problem könnte ein möglicher Qualitätsverlust bei der Kovertierung von PDF- in Bilddokumente sein. Es wird zwar standardmäßig mit 600 DPI konvertiert, aber eine 100% verlustfreie Umwandlung ist (nach Verständnis des Autors) nicht möglich, da in eine Rastergrafik konvertiert wird und die Angabe der gewünschten DPI auch keinen Sinn ergeben würde. Um 100% verlustfrei zu konvertieren, bräuchte man theoretisch unendlich DPI.

Da mit 600 DPI konvertiert wird, sind die Ausgangsbilder der Konvertierung sehr groß. Dies könnte wiederum ein Problem für den OCR-Algorithmus sein, da nun die Buchstaben so groß sind, dass diese eventuell als mehrere Zeichen erkannt werden oder nicht mehr auf die entsprechenden Vergleichsmuster passen und somit falsch erkannt werden.

Ein Grund für das Problem betreffend der falsche Reihenfolge der erkannten Wörter, könnte eine falsch eingestellte Segmentierungsmethode des OCR-Algorithmus sein. Über die Segmentierungsmethode kann eingestellt werden, wie ein Bild intern von Tesseract unterteilt wird. Ist die falsche Methode eingestellt, kann dies zu einer falschen Reihenfolge und sogar zu allgemein fehlerhaft erkannten Wörtern führen.

Was definitiv auch zu fehlerhaften Resultaten führt, ist dass kein benutzerdefiniertes Wörterbuch verwendet wird, womit Tesseract eventuelle Fehler durch Vergleiche mit den Wörtern des Wörterbuchs ausbessern könnte.

Alles in Allem, wird der Hauptgrund der schlechten Testergebnisse die Verwendung von Tesseract in den Standardeinstellungen sein. Wie aber bereits in Abschnitt 4.3.1 erklärt, lag es nicht im Themenbereich und zeitlichem Spektrum dieser Bachelorarbeit, die optimalen Einstellungen für Tesseract zu finden.

Vorschläge und Ideen was getan werden sollte, um die Ergebnisse zu optimieren sind in Abschnitt 6.2.1 erläutert.

## 4.4 Gesamtüberblick über das entwickelte Konzept

In diesem Abschnitt soll ein Überblick über das im Rahmen dieser Bachelorarbeit entwickelte Konzept vermittelt werden.

Als Erstes gilt es, die Begriffe 'printed Document' beziehungsweise 'gedrucktes Dokument' und 'digital Document' beziehungsweise 'digitales Dokument' zu definieren. Im folgenden Text und in den folgenden Graphiken steht der Begriff 'digital Document' beziehungsweise 'digitales Dokument' für ein Textdokument jeglicher Art, welches auf einem oder mehreren Computern, Servern, SmartPhones, Tablets oder sonstigen digitalen Geräten gespeichert ist.



Abbildung 4.11: Erstellen eines digitalen Dokuments mit Hilfe eines Textverarbeitungsprogramms

Ein solches digitales Dokument kann entweder ein durch einen Scanner erzeugter Scan eines gedruckten Dokuments sein (siehe Abbildung 4.11) oder ein mit einem Textverarbeitungsprogramm neu erstelltes (rein digitales) Dokument sein (siehe Abbildung 4.12).

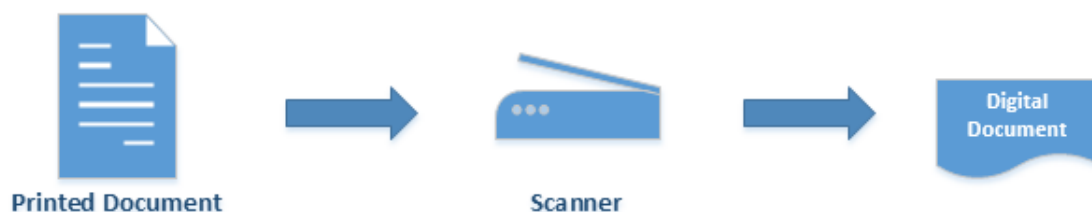


Abbildung 4.12: Erzeugung eines digitalen Dokuments durch Scannen eines gedruckten Dokuments

Der Begriff 'printed Document' beziehungsweise 'gedrucktes Dokument' steht für ein Textdokument jeglicher Art, welches durch einen Drucker aus einem digitalen Dokument auf ein Blatt Papier erzeugt wurde (siehe Abbildung 4.13).



Abbildung 4.13: Erzeugung eines gedruckten Dokuments durch Drucken eines digitalen Dokuments

#### 4.4.1 Verlinkung von Dokumenten

Der Prozess der Verlinkung von Dokumenten ist in Abbildung 4.14 schematisch dargestellt. Ein digitales Dokument wird zusammen mit einem Privaten Schlüssel (in diesem Fall aus den RSA-Kryptosystem) und dem dazugehörigen Digitalen Zertifikat (nach X.509 Standard) bereitgestellt. Der Private Schlüssel und das Zertifikat dienen zum Signieren des Dokuments. Somit kann bei einer späteren Überprüfung des Dokuments sichergestellt werden, wer der Unterzeichner (oder Aussteller) des Dokuments ist und ob die Kopie gefälscht wurde. Das im Umfang dieser Arbeit erstellte 'Printed Document Linking System' verarbeitet diese Eingangsdaten zu einer Digitalen Signatur und einem aus dem textuellen Inhalt ('Content') abgeleiteten, kryptographischen Hashwert ('Content-Hash'). Die Signatur, der Content-Hash und das Zertifikat werden zusammen Verlinkungsdaten genannt. Diese Verlinkungsdaten werden beim Verlinkungs-Provider IPFS gespeichert. Der IPFS-Link, über den die Verlinkungsdaten bei IPFS heruntergeladen werden können, wird, je nach Typ des Dokuments in Form eines QR-Codes oder als reiner Text in das Dokument injiziert und als neues Ausgabedokument auf der Festplatte gespeichert.

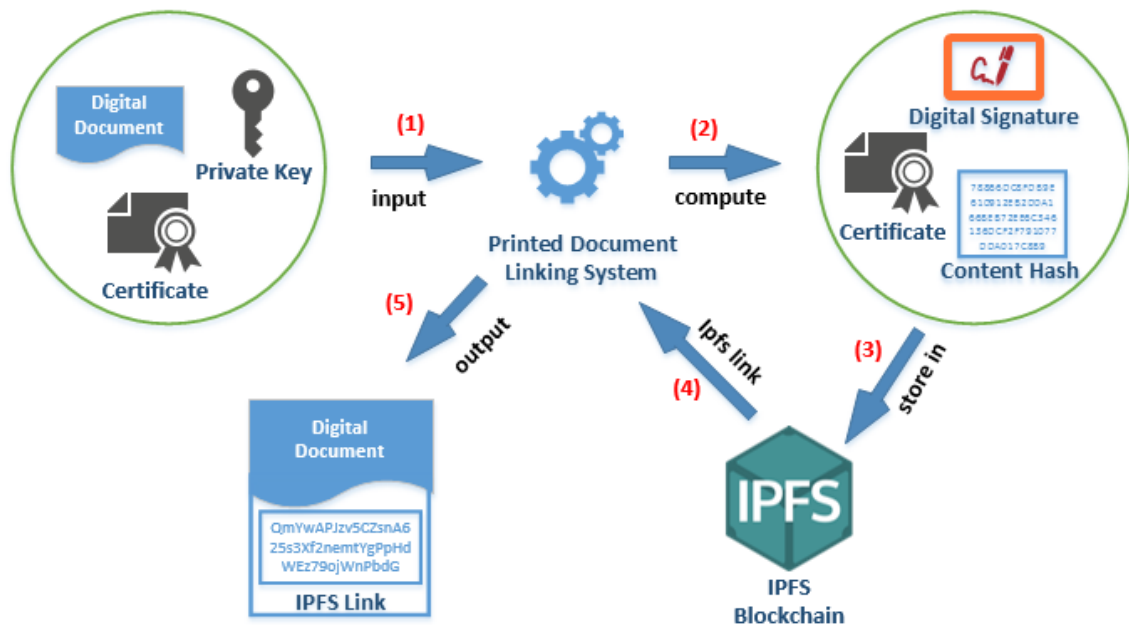


Abbildung 4.14: Schematische Darstellung der Verlinkung von Dokumenten

In Abbildung 4.15 ist der Prozess zur Verlinkung, wie er innerhalb des 'Printed Document Linking System' (siehe Abschnitt 4.2.3) Schritt für Schritt verläuft, (auf einem hohen Abstraktionsniveau) dargestellt.

Als Erstes wird das digitale Dokument vom System geladen. Danach wird der textuelle Inhalt des Dokuments extrahiert. Dies wird abhängig vom Format des Dokuments durch verschiedene Algorithmen bewerkstelligt. Der extrahierte Inhalt wird daraufhin normalisiert. Aus diesem normalisierten Inhalt wird der Content-Hash erzeugt. Um die Digitale Signatur zu erzeugen, wird der Private Schlüssel in den Speicher geladen. Ist es ein geschützter Schlüssel, muss der User das Passwort zur Freigabe eingeben. Wurde der Schlüssel erfolgreich geladen, wird aus ihm und dem Content-Hash die Signatur generiert. Daraufhin wird das Digitale Zertifikat geladen und zusammen mit der Signatur und dem Content-Hash bei IPFS gespeichert. Der erhaltene IPFS-Link wird dann mit dem Eingabedokument kombiniert und in einem neuen Ausgabedokument auf der Festplatte gespeichert.

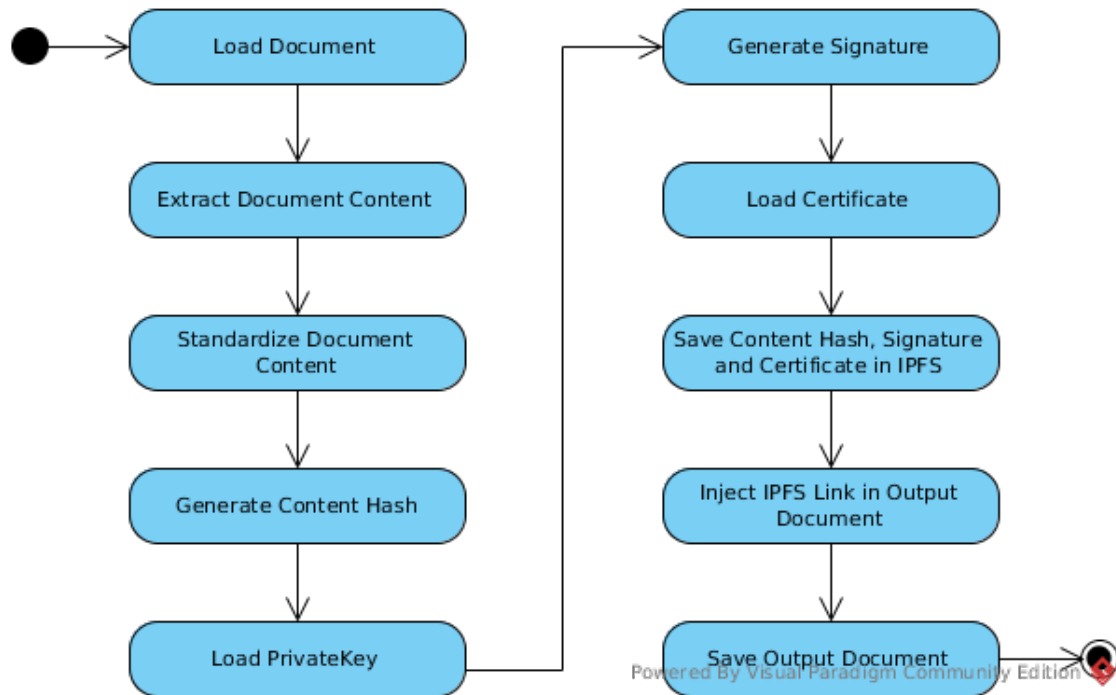


Abbildung 4.15: 'High-Level' Aktivitätsdiagramm der Verlinkung von Dokumenten

#### 4.4.2 Verifikation von Dokumenten

Bei der Verifikation von Dokumenten wird dem 'Printed Document Linking System' ein digitales Dokument, welches einen IPFS-Link enthält, als Eingabe bereitgestellt. Der Link wird extrahiert und die Signatur, der Content-Hash und das Zertifikat von IPFS heruntergeladen. Dann wird ein zweiter Content-Hash aus dem Eingabedokument, analog zum Vorgang in Abschnitt 4.4.1, generiert. Durch Vergleichen der beiden Content-Hashes und Prüfung der Signatur mit Hilfe des Zertifikats, wird der Verifikationsstatus des Dokuments ermittelt. Dieser Vorgang ist in Abbildung 4.15 schematisch beschrieben.

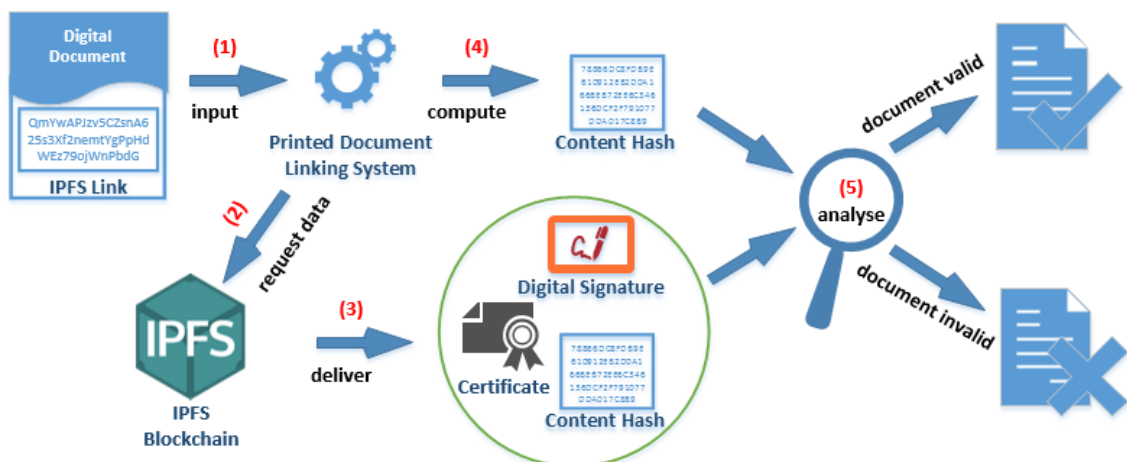


Abbildung 4.16: Schematische Darstellung der Dokumenten Verifikation

Der Verifikationsprozess eines Dokuments, wie er innerhalb des Printed Document Linking System abläuft, ist in Abbildung 4.17 dargestellt.

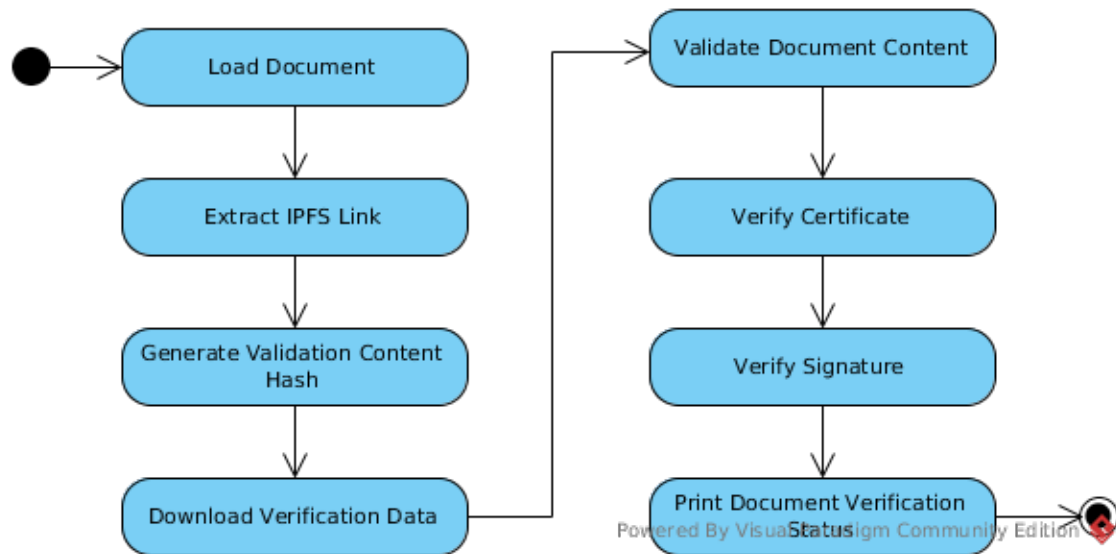


Abbildung 4.17: Highlevel Aktivitätsdiagramm der Dokumenten Verifikation

Als erster Schritt wird das Dokument geladen. Darauf folgend wird der im Dokument enthaltene IPFS-Link extrahiert. Als Nächstes wird, analog zum Vorgang in Abbildung 4.15 der Content-Hash des Eingabedokuments berechnet. Die Verifikationsdaten (Content-Hash, Signatur und Zertifikat) werden über den IPFS-Link von IPFS heruntergeladen. Durch den Vergleich der beiden Content-Hashes wird der Inhalt des Dokuments auf Änderung überprüft. Danach wird das Zertifikat validiert. Mit Hilfe des Zertifikats wird dann die Signatur verifiziert. Als Letztes werden die drei Analyseergebnisse überprüft und der gesamt Verifikationsstatus des Dokuments evaluiert und ausgegeben.

#### 4.4.3 Abgrenzung zu den Arbeiten aus Kapitel 2

Das entwickelte Konzept ist eine Kombination der, in Kapitel 2 beschriebenen, Ansätze. Übernommen wurde die Idee, die Daten zur Verifikation eines Dokuments zu bündeln. Jedoch werden diese Daten, im Unterschied zu den Arbeiten aus Abschnitt 2.2.2, nicht direkt, sondern indirekt über einen Link im Dokument eingebettet. Ebenfalls übernommen wurde die Idee, einen QR-Code zu verwenden, der die Daten zur Verifikation referenziert. Im Gegensatz zu den Arbeiten aus Abschnitt 2.2.3, werden die Verifikationsdaten jedoch nicht in einer zentralen Datenbank, sondern in der dezentralen IPFS-Blockchain gespeichert. Die Möglichkeiten der Wasserzeichenmethoden aus Abschnitt 2.2.1 wurden gänzlich verworfen.

Die Vorteile des in dieser Arbeit entwickelten Konzepts im Vergleich zu den Ansätzen aus Kapitel 2 sind klar erkennbar: Durch Speicherung der Daten zur Verifikation in der dezentralen IPFS-Blockchain ist das System nicht auf einen zentralen Namensdienst oder eine zentrale Datenbank angewiesen. Außerdem können beliebig große Datensätze zu einem Dokument gelinkt werden und man ist nicht mehr an die limitierte Speicherkapazität eines QR-Codes gebunden, da der QR-Code nur den Link zu den Daten im IPFS-Netzwerk beinhaltet. Der Nachteil hierbei ist, dass immer eine Verbindung zur IPFS-Blockchain nötig ist, um Dokumente zu verlinken oder zu verifizieren. Dieser Nachteil spielt jedoch im Rahmen dieser Arbeit keine Rolle, da das Ziel ist Dokumente online zu verlinken.

## 4.5 Proof-Of-Concept Software Design

Bei den Designentscheidungen der 'Proof-Of-Concept'-Implementierung wurde neben den 'GRASPs' (siehe [47]) stets die Modularität und Erweiterbarkeit des Systems sowie die Einheitlichkeit der Gesamtarchitektur als oberste Prioritäten gesehen. Um diese Prioritäten in der Architektur umzusetzen, wurde an vielen Stellen auf die Entwurfsmuster der 'Gang of Four' aus deren Buch 'Design Patterns. Elements of Reusable Object-Oriented Software' (siehe [48]) zurückgegriffen.

Dies führt allerdings dazu, dass das Design dem KISS-Prinzip (siehe [49]) oft widerspricht. Eine kompromisslose Lösung für diesen Konflikt gibt es leider nicht.

Da es sich um eine Proof-Of-Concept Implementierung handelt, wird kein Versuch unternommen, einen fehlerhaften Zustand zu verbessern. Mit anderen Worten: Es gilt das 'Exception-Safety-Level' 'No exception safety' ([50]).

Außerdem wurde C++ Idiome wie zum Beispiel 'Rule of three/five/zero' (siehe [51]), 'Pimpl' (siehe [52]), RAII (siehe [53]) sowie das Testen der Software durch Unit-Tests aus zeitlichen und thematischen Gründen ignoriert.

### 4.5.1 Verwendete Software-Bibliotheken

In diesem Abschnitt werden die in der 'Proof-Of-Concept'-Implementierung verwendeten Software Bibliotheken kurz zusammengefasst und deren Wahl begründet. Ein Grund, der in den folgenden Tabellen nicht aufgelistet, wird aber einer der Hauptgründe für jede der Bibliotheken (außer GhostScript), besteht in der kostenlosen Nutzung und Open-Source Entwicklung.

<b>Name</b>	Tesseract [29]
<b>Beschreibung</b>	Open-Source Bibliothek zur Texterkennung (OCR). Siehe Abschnitt 3.1.
<b>Einsatzbereich</b>	Extraktion des textuellen Inhalts von PDF- und Bilddokumenten.
<b>Begründung</b>	Zum einen ist Tesseract die am weitesten verbreitete und am besten dokumentierte Bibliothek. Außerdem ist Tesseract vollständig in C++ entwickelt und bietet viele Einstellungs- und Erweiterungsmöglichkeiten.

Tabelle 4.21: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Tesseract

<b>Name</b>	Boost C++ Libraries [54]
<b>Beschreibung</b>	Sammlung von 'high-quality' Open-Source Bibliotheken in C++.
<b>Einsatzbereich</b>	An vielen Stellen. Vorallem für Dateioperationen, RegEx, String-Manipulation und Program-Options (für das CLI)
<b>Begründung</b>	Viele der Boost Bibliotheken sind zum Teil des ISO C++ Standards geworden (oder werden es noch). Außerdem sind sie sehr weit verbreitet, werden von großen Namen wie Herb Sutter, Bjarne Stroustrup, Scott Meyers und Andrei Alexandrescu empfohlen und sind zudem sehr gut dokumentiert.

Tabelle 4.22: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Boost

<b>Name</b>	OpenCV [55]
<b>Beschreibung</b>	Open-Source Computer Vision Bibliothek. Beinhaltet über 7500 Algorithmen zur maschinellen Bildverarbeitung.
<b>Einsatzbereich</b>	An mehreren Stellen zur Manipulation und zum Laden und Speichern von Bildern.
<b>Begründung</b>	Standard Bibliothek wenn es um Bildverarbeitung geht. Des weiteren ist die Bibliothek in C++ geschrieben und sehr gut dokumentiert.

*Tabelle 4.23: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: OpenCV*

<b>Name</b>	Ghostscript Interpreter API [56]
<b>Beschreibung</b>	Die Ghostscript Interpreter API, ist eine Schnittstelle zur Ansteuerung des bekannten GhostScript Tools zur PDF Manipulation.
<b>Einsatzbereich</b>	Konvertierung von PDF nach Bild und PDF nach Text.
<b>Begründung</b>	Diese API wurde wegen Recherche im Internet und Hinweisen von Usern auf Softwareentwicklerforen und weil das Linux Terminal Tool bekannt war, gewählt. Außerdem ist die Schnittstelle in C geschrieben und kann daher leicht eingesetzt werden.

*Tabelle 4.24: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Ghostscript Interpreter API*

<b>Name</b>	OpenSSL [57]
<b>Beschreibung</b>	OpenSSL ist ein Open-Source Projekt und Standard Bibliothek für sämtliche kryptographische Operationen.
<b>Einsatzbereich</b>	Erzeugung des Content-Hash sowie Erzeugung und Verifikation der Signatur eines Dokuments.
<b>Begründung</b>	Diese Bibliothek wurde gewählt, weil sie als sehr robust gilt und der Standard für sämtliche sicherheitskritischen Operationen ist. Außerdem ist die Bibliothek in C geschrieben, was den Einsatz im System erleichtert.

*Tabelle 4.25: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: OpenSSL*

<b>Name</b>	PugiXML [58]
<b>Beschreibung</b>	Open-Source Bibliothek zur Verarbeitung und Manipulation von XML-Dateien.
<b>Einsatzbereich</b>	Extraktion des Inhalts aus XML-Dateien und Injektion eines Links in XML-Dateien.
<b>Begründung</b>	Wurde verwendet, weil PugiXML eine moderne, effiziente und sehr gut dokumentierte Bibliothek in C++ ist. Außerdem unterstützt sie XPath, was die Extraktion des Inhalts und Injektion eines Links bei XML-Dokumenten deutlich erleichtert.

*Tabelle 4.26: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: PugiXML*

<b>Name</b>	IPFS C++ API [45]
<b>Beschreibung</b>	Open-Source Bibliothek zur Ansteuerung des IPFS-Client.
<b>Einsatzbereich</b>	Kommunikation mit dem IPFS-Netzwerk
<b>Begründung</b>	Diese Schnittstelle wird verwendet da sie sehr gut dokumentiert und sehr einfach zu handhaben ist. Außerdem ist sie die einzige und offizielle in C++ entwickelte Schnittstelle für IPFS.

*Tabelle 4.27: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: IPFS C++ API*

<b>Name</b>	Nlohmann JSON [59]
<b>Beschreibung</b>	Open-Source Bibliothek zum Parsen und Erzeugen von JSON-Dateien beziehungsweise JSON-Objekten.
<b>Einsatzbereich</b>	Erzeugen der JSON-formatierten Verlinkungsdaten eines Dokuments
<b>Begründung</b>	Diese Bibliothek wurde aufgrund ihrer sehr einfachen und komfortablen Handhabung, Effizienz und der sehr guten Dokumentation gewählt. Des weiteren wurde sie in C++ entwickelt und bietet alle nötigen Funktionalitäten.

*Tabelle 4.28: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Nlohmann JSON*

<b>Name</b>	img2pdf [60]
<b>Beschreibung</b>	Open-Source Python Bibliothek um Bilddateien in PDFs umzuwandeln.
<b>Einsatzbereich</b>	Wird genutzt um Bilddateien in PDFs zu konvertieren.
<b>Begründung</b>	Wurde gewählt, weil die Bibliothek die einzig kostenlose ist, mit der das Vorhaben ohne großen Aufwand umgesetzt werden kann. Ein weiterer Grund ist, dass es möglich ist Python Code in C++ einzubetten, was die Bibliothek im System einsetzbar macht.

*Tabelle 4.29: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: img2pdf*

<b>Name</b>	Nayuki QR Code generation library [61]
<b>Beschreibung</b>	Kompakte Open-Source Bibliothek zur Generierung von QR-Codes
<b>Einsatzbereich</b>	Wird genutzt, um aus einem Link einen QR-Code zu erzeugen, der dann in ein PDF- oder Bilddokument eingefügt werden kann.
<b>Begründung</b>	Wird verwendet, weil die Bibliothek sehr kompakt und einfach zu bedienen ist. Außerdem ist sie in C++ verfügbar und es kann auf die Module des generierten QR-Code zugegriffen werden, was es einfacher macht, den QR-Code in die OpenCV Bibliothek zu überführen.

*Tabelle 4.30: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Nayuki QR Code generation library*



<b>Name</b>	ZXing (C++ Port)[62]
<b>Beschreibung</b>	C++ Portierung der Open-Source Java Bibliothek zur Detektion und Dekodierung von QR-Codes.
<b>Einsatzbereich</b>	Detektion und Dekodierung von QR-Codes auf verlinkten PDF- und Bilddokumenten.
<b>Begründung</b>	Wird verwendet, weil die Bibliothek sehr komfortabel zu bedienen ist und alle gängigen QR-Code Versionen unterstützt.

*Tabelle 4.31: Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: ZXing*

## 4.5.2 Überblick über die Datentypen

In diesem Abschnitt werden die wichtigsten Datentypen sowie deren Zusammenhänge, mit denen das 'Printed Document Linking System' intern arbeitet, anhand Tabelle 4.32 und Abbildung 4.18 erklärt.

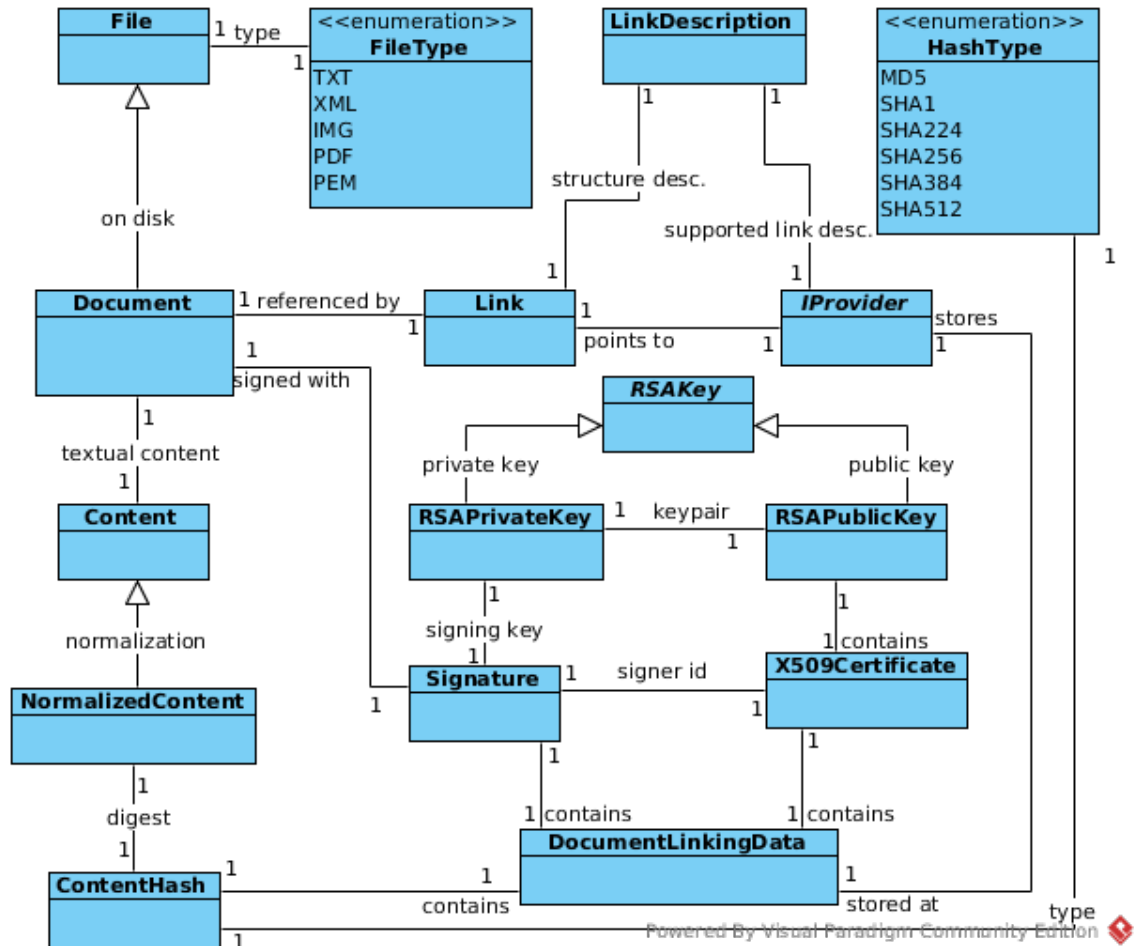


Abbildung 4.18: Wichtigste Datentypen des 'Printed Document Linking System' und deren Zusammenhänge

Klassenname	Beschreibung
File	Repräsentiert eine Datei auf einer Festplatte. Eine Datei lädt sich selbst und verwaltet intern einen <code>std::fstream</code> . Dateien werden als einer der folgenden Typen interpretiert: reiner Text, XML, Bild, PDF oder PEM.
Document	Ist die Hauptarbeitsklasse des Systems und hat die Datentypen <code>Content</code> , <code>NormalizedContent</code> , <code>ContentHash</code> , <code>Signature</code> und <code>Link</code> als Membervariablen. Ein Dokument ist eine spezialisierte Datei.
Content	Repräsentiert den textuellen Inhalt eines Dokuments.
NormalizedContent	Stellt den Inhalt eines Dokuments nach der Normalisierung dar.
ContentHash	Durch eine kryptographische Hashfunktion wird aus einem normalisierten Inhalt ein kryptographischer Hashwert erzeugt. Mögliche Hash-Typen sind MD5, SHA1, SHA224, SHA256, SHA384 und SHA512.
X509Certificate	Wrapperklasse für ein Zertifikat nach X.509-Standard. Ein X.509-Zertifikat beinhaltet den Public-Key des Zertifikatinhabers.
RSAPublicKey	Wrapperklasse für Public-Keys aus dem RSA-Kryptosystem.
RSAPrivateKey	Wrapperklasse für Private-Keys aus dem RSA-Kryptosystem
Signature	Wrapperklasse für eine digitale Signatur aus dem RSA-Kryptosystem. Sie wird durch einen RSA-Private-Key erzeugt und kann mit einem X.509-Zertifikat beziehungsweise einem RSA-Public-Key verifiziert werden. Unterzeichnet wird der Content-Hash eines Dokuments.
LinkingData	Diese Klasse bündelt Informationen über ein X.509-Zertifikat, eine Signatur und einen Content-Hash eines Dokuments. Die Klasse kann in und vom JSON-Format serialisiert werden und dient zur Verlinkung eines Dokuments bei einem Provider sowie zur späteren Verifikation.
Link	Ein Link zeigt auf die bei einem Provider gespeicherten, JSON-formatierten 'LinkingData' eines Dokuments. Über diesen Link können die Daten bei dem entsprechenden Provider abgerufen werden.
IProvider	Interface eines Providers, bei dem die 'DocumentLinkingData' gespeichert und (über den entsprechenden Link) wieder abgerufen werden können.
LinkDescription	Eine Hilfsklasse um den Aufbau eines Links zu beschreiben. Ein IProvider beschreibt somit, welche Art von Link unterstützt wird. Wird auch genutzt, um einen Link auf einem bereits verlinkten Dokumenten zu suchen.

*Tabelle 4.32: Namen und Beschreibungen der wichtigsten Datentypen im 'PDLS'*

### 4.5.3 'High-Level' Architektur

Das 'Printed Document Linking System' besteht aus den in Tabelle 4.33 beschriebenen Modulen.

Jedes dieser Module ist ein Subsystem mit ebenfalls modularem Aufbau, sodass das Gesamtsystem mit wenig Implementierungsaufwand verändert oder erweitert werden kann. Um die Softwarestruktur übersichtlich und einfach zu halten, ist jedes der Module als 'Facade' (siehe 'Facade-Pattern' [48]) implementiert. Die Hauptklasse ('PrintedDocumentLinkingSystem') beinhaltet die 'High-Level'-Algorithmen (siehe Abbildung 4.21), welche die einzelnen Module ansteuern (siehe Abbildung 4.19). Diese Designentscheidung hat Vorteile für den/die Entwickler\*in. Es lässt sich schnell und einfach ein Überblick über das System verschaffen und es ist sofort klar, an welcher Stelle gewünschte Änderungen vorgenommen werden müssen.

Name	Abkürzung	Aufgabe
Content Extraction Module	CEM	Extraktion des textuellen Inhalts eines Dokuments
Content Normalization Module	CNM	Normalisierung des extrahierten Inhalts eines Dokuments
Content-Hash Generation Module	CHGM	Generierung des Content-Hash aus dem normalisierten Inhalt eines Dokuments
Signature Generation Module	SGM	Generierung einer Digitalen Signatur für ein Dokument
Document Linking Module	DLM	Verlinkung eines Dokuments beziehungsweise dessen Verifikationsdaten in einer Blockchain
Link Injection Module	LIM	Injektion des Links zu den Verlinkungsdaten eines Dokuments in ein Ausgabedokument
Link Extraction Module	LEM	Extraktion des Links zu den Verlinkungsdaten eines Dokuments aus dessen Inhalt
Linking-Data Loading Module	LDLM	Herunterladen der Verlinkungsdaten eines Dokuments
Content Validation Module	CVM	Validierung des Inhalts eines Dokuments
Signature Verification Module	SVM	Verifikation einer Digitalen Signatur eines Dokuments

*Tabelle 4.33: Namen und Aufgaben der Module des 'Printed Document Linking System'*

Des weiteren ist anzumerken, dass jedes der Subsysteme sowie auch das 'Printed Document Linking System' (beziehungsweise dessen Hauptklasse) als 'Singletons' (siehe 'Singleton-Pattern' [48]) umgesetzt sind. Das hat zum Vorteil, dass keine überflüssigen Instanzen erstellt werden können, was zu undefiniertem Verhalten oder 'Race-Conditions' führen kann. Wobei es in der aktuellen Version zu keinen 'Race-Conditions' kommen kann, da es sich um eine 'Single-Threaded-Application' handelt. Vorschläge zur Umwandlung des Systems in eine 'Multi-Threaded-Application' finden sich in Abschnitt 6.2.2.5.

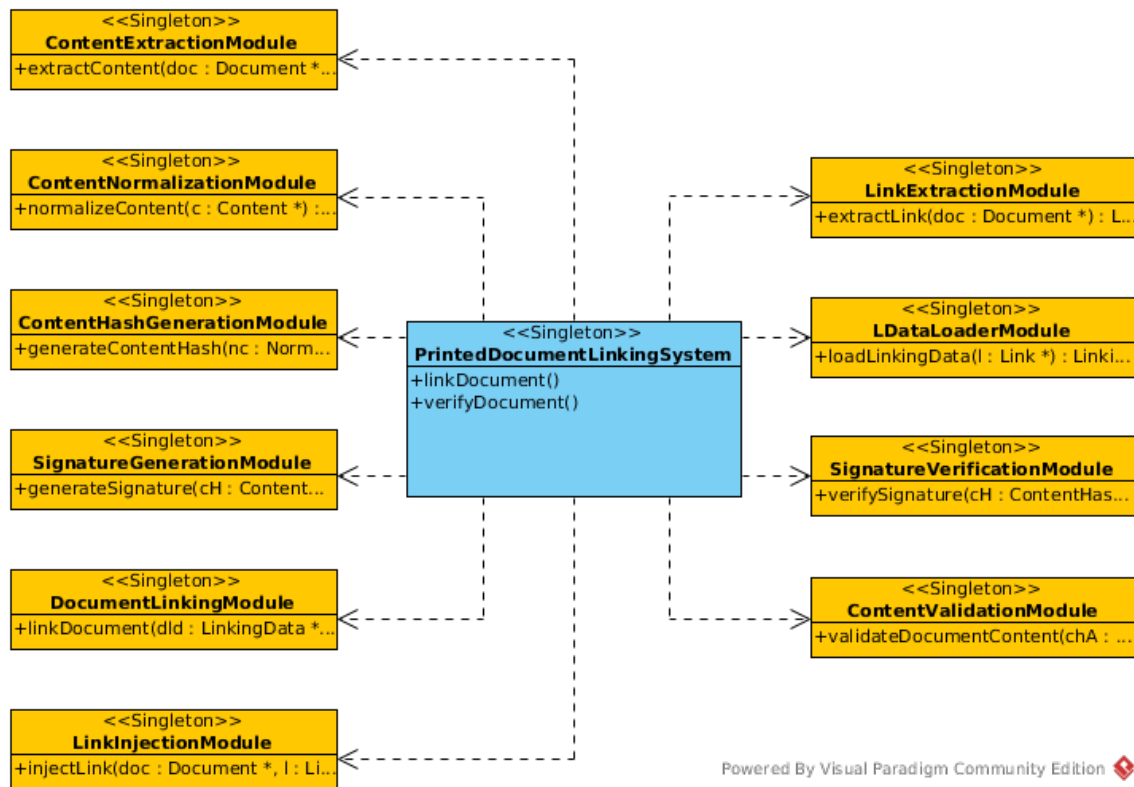


Abbildung 4.19: Module des 'Printed Document Linking System' mit 'Facade-' und 'Singleton-Pattern' Architektur

#### 4.5.4 'High-Level' Algorithmus

Der in Abbildung 4.21 dargestellte Algorithmus, ist der 'High-Level'-Algorithmus in der Hauptklasse des 'PDLs'. Wie bereits erwähnt, wird das 'Facade-Pattern' ausgenutzt, sodass nur die Funktionen der Module angesteuert beziehungsweise aufgerufen werden müssen. Tritt an irgendeiner Stelle im Ablauf ein Fehler auf, wird eine dem Fehler entsprechende Exception geworfen und der Vorgang mit Fehlermeldung abgebrochen.

Um die Bedienung des Programms möglichst nutzerfreundlich zu gestalten und die Möglichkeit zu bieten das Programm über ein Script zu steuern, wurden mit Hilfe der Boost Bibliothek [54] verschiedene Programmoptionen implementiert. Über diese in Abbildung 4.20 gezeigte Optionen, kann der Nutzer das Verhalten des Systems steuern. Hat ein/e Nutzer\*in eine zwingend notwendige Information, wie beispielsweise der Pfad zum Eingabedokument, nicht geliefert, wird er/sie vom System aufgefordert dies zu tun.

```
Program Options:
-h [ --help ]           Print this help Message
-l [ --link ]           Create a verifiable document
-v [ --verify ]         Verify a document
-d [ --document ] arg   /path/to/document
-k [ --privateKey ] arg  /path/to/rsa/privateKey.pem
-c [ --x509-certificate ] arg /path/to/x509/certificate.pem
-p [ --provider ] arg   Provider where the Document should be linked or
                        where it is linked (only IPFS is supported
                        right now so this option can be ignored for
                        now)
-o [ --output ] arg     /path/to/output/doc
-t [ --chType ] arg     Content Hash Type (MD5 | SHA1 | SHA224 | SHA256
                        | SHA384 | SHA512)
--chPrefix arg          Content Hash Prefix, Default : #HSU#
--chPostfix arg         Content Hash Postfix, Default : #CH#
--textPDF               Flag that indicates that the input PDF doc is a
                        PDF that contains textual (and searchable)
                        content
--noLIM                 Don't run Link-Injection-Module -> Link gets
                        printed in Terminal and is not injected into
                        Document!
--noLEM                 Don't run Link-Extraction-Module -> User has to
                        provide the Link via Terminal!
--saveConvertedFile     If the input document is a PDF it gets
                        converted. If this option is set, the converted
                        file gets saved in the directory of the input
                        document!
```

Abbildung 4.20: Hilfeanzeige des 'Printed Document Linking System'

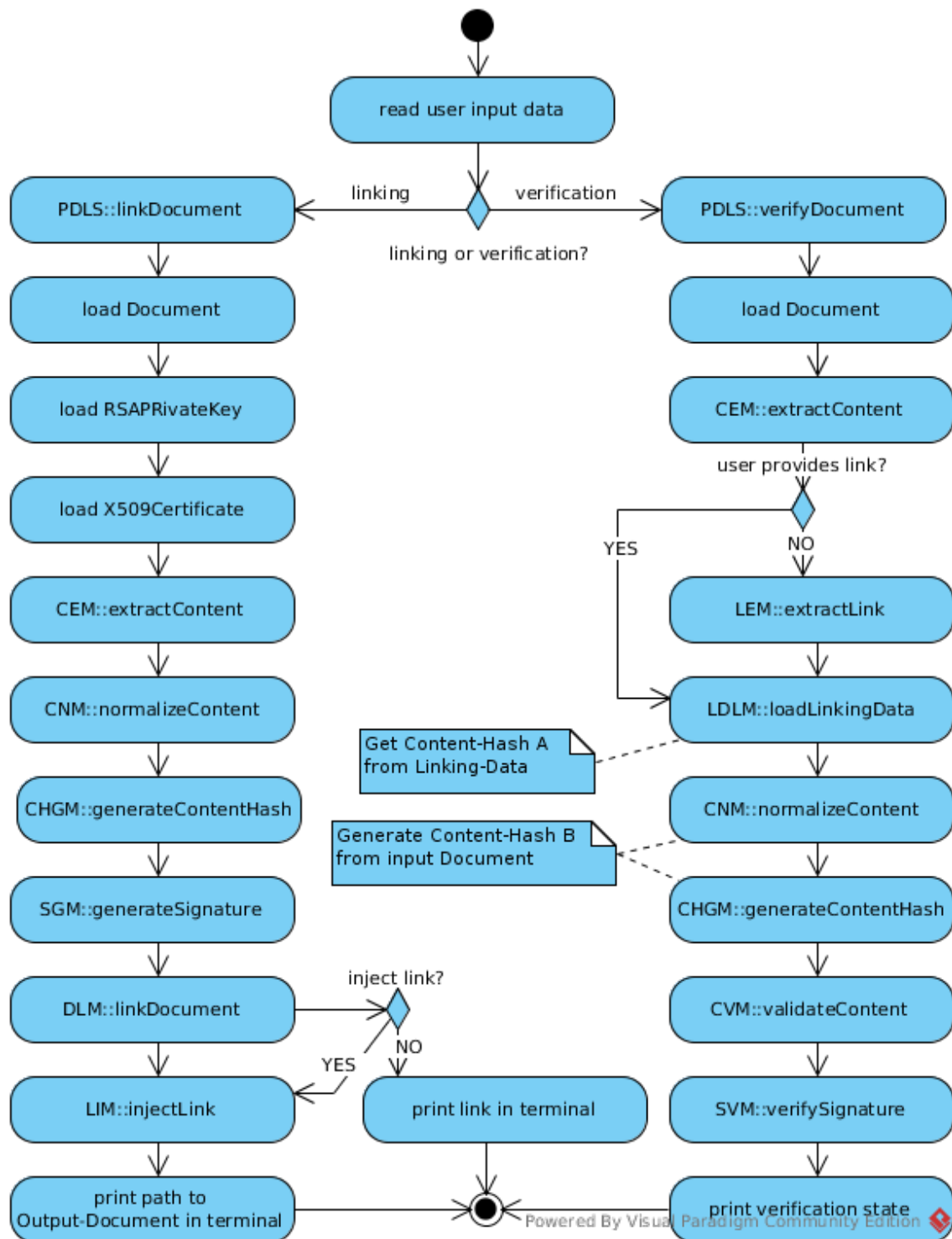


Abbildung 4.21: Aktivitätsdiagramm der 'High-Level'-Algorithmen in der Hauptklasse des 'Printed Document Linking System'

### 4.5.5 Architektur und Algorithmen der einzelnen Module

In diesem Abschnitt wird die Architektur und die Algorithmen der Module des 'Printed Document Linking System' beschrieben.

Die Architektur wird anhand eines Klassendiagramms, welches die wichtigsten Klassen, Funktionen und Attribute des Moduls sowie deren Zusammenhänge skizziert, beschrieben. Um die Klassendiagramme genau zu verstehen, ist es wichtig zu wissen, dass die Namen der Klassen deren Funktionalität ausdrücken. Besonderer Fokus muss auch auf den Anfangsbuchstaben und den Schriftstil einer Klasse beziehungsweise deren Methoden gerichtet werden. Wenn der Anfangsbuchstabe einer Klasse ein 'I' ist, handelt es sich um ein Interface (auch abstrakte Klasse genannt). Ist der Schriftstil 'kursiv', deutet dies ebenfalls auf eine abstrakte Klasse oder (rein virtuelle) Methode hin. In C++ werden Kompositionen (  $\blacklozenge$  ) oft in Membervariablen des Typs T und Aggregationen (  $\blacklozenge$  ) in Membervariablen des Typs T\* übersetzt. Mit einer Komposition wird ein Member einer Klasse gekennzeichnet, ohne dessen Dasein die Klasse selbst keinen Sinn mehr ergibt und nicht mehr funktionsfähig ist. Dies sagt jedoch nichts über die eigentliche Implementierung in C++ aus. Wenn es sich um Kompositionen von 'Singleton'- oder 'Strategy'-Klassen handelt, sind diese in Membervariablen vom Typ T\* im Code umgesetzt. Außerdem werden die wichtigsten Klassen und Interfaces sowie Besonderheiten der jeweiligen Architektur kurz erläutert. Hierbei ist zu beachten, dass in den meisten Fällen, wenn von einer 'Instanz' oder einem 'Objekt' gesprochen wird, ein Zeiger auf eben diese Objekte oder Instanzen gemeint ist. Ein Blick in das Klassendiagramm liefert die genaue Information.

Anhand eines Aktivitätsdiagramms wird der interne Ablauf der Module (sehr) grob geschildert. Bei diesen Aktivitätsdiagrammen gilt es sich im klaren darüber zu sein, dass die Darstellungen nur einen groben Ablauf schildern und eine Aktion in einem Diagramm in (fast) allen Fällen deutlich mehr als ein Funktionsaufruf im Code darstellt. Insbesondere wird die Nutzung von APIs nur durch den Text einer Aktion angedeutet und die einzelnen Schritte, die nötig sind um die APIs anzusteuern werden ignoriert.

Was ebenfalls an dieser Stelle ignoriert wird, ist das Verhalten im Fehlerfall, da immer das gleiche Schema angewandt wird: tritt ein Fehler auf, wird eine dem Fehler entsprechende Exception geworfen und das Programm mit der entsprechenden Fehlermeldung beendet. In den allermeisten Fällen treten Fehler auch nur dann auf, wenn der Nutzer eine falsche Eingabe (zum Beispiel falsche Pfadangaben oder Dokumententypen) macht.



#### 4.5.5.1 Content-Extraction-Module ↔ CEM

Dieses Modul verlangt als Eingabe ein 'Document'-Objekt. Die Ausgabe stellt eine 'Content'-Instanz dar. Um verschiedene Dokumententypen zu unterstützen, wurde in der Architektur des 'CEM' das 'Strategy'-Pattern (siehe [48]) angewandt. Dies ermöglicht es zur Laufzeit das Verhalten des Moduls je nach Dokumententyp zu verändern. Das Basisinterface des 'Patterns' ist '*IExtractionStrategy*' (siehe Abbildung 4.22).

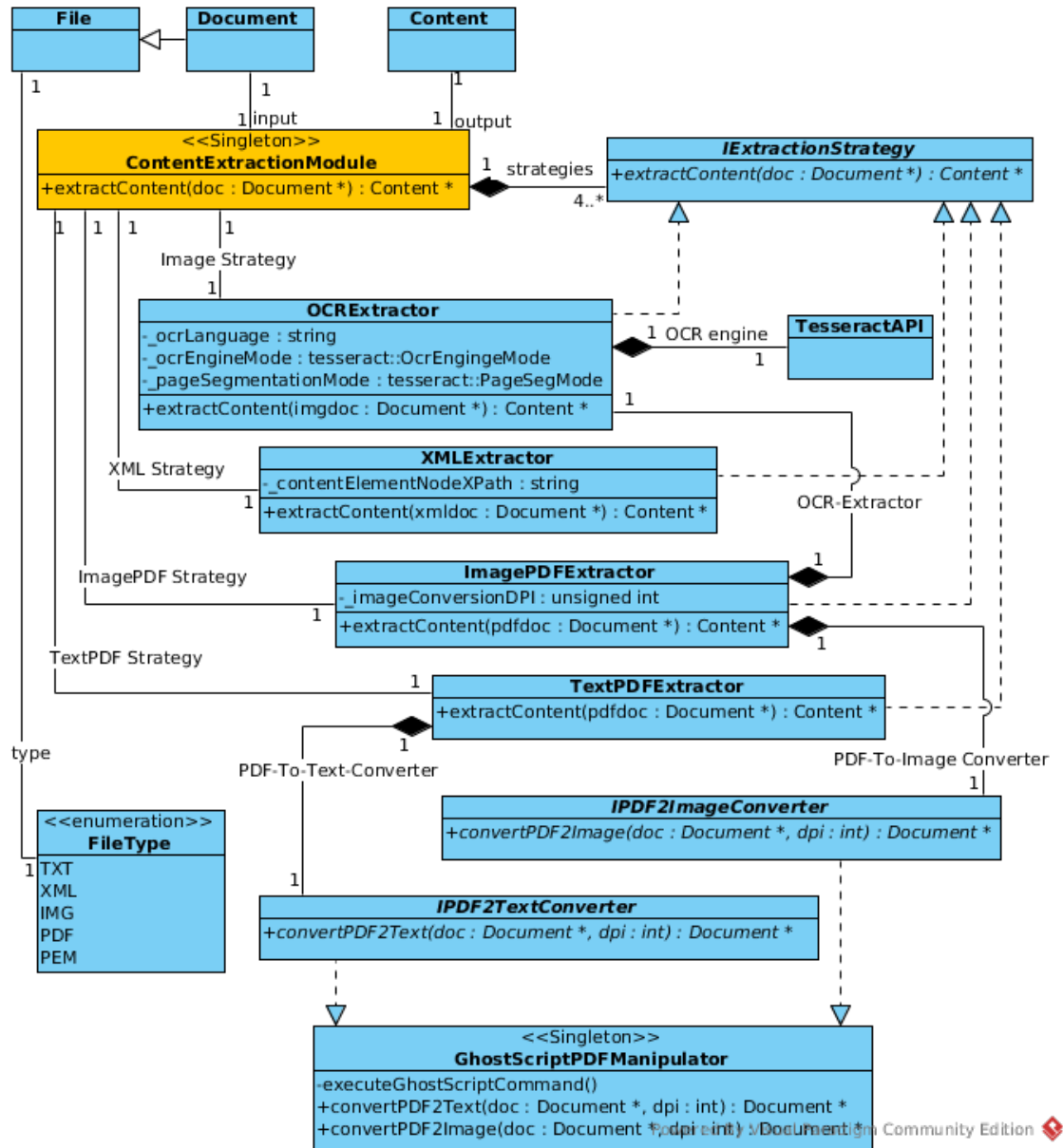


Abbildung 4.22: Klassendiagramm des 'Content-Extraction-Module'

In der Klasse 'OCRExtractor' ist der Algorithmus zur OCR-Erkennung mit Hilfe der Tesseract Bibliothek [29] umgesetzt. Um ein Bild zu laden und/oder zu bearbeiten wird die Bibliothek OpenCV [55] genutzt. Über die Membervariablen der Klasse kann das Verhalten der Tesseract-API eingestellt werden.

Die Klasse 'XMLExtractor' realisiert die Extraktion eines Inhaltes aus einem XML-Dokument mit Hilfe der pugiXML Bibliothek [58]. Über die Membervariable 'XML-Extractor.\_contentElementNodeXPath' wird dem Algorithmus über einen XPath-Ausdruck [63] mitgeteilt, wie beziehungsweise wo die XML-Elemente zu finden sind,

welche den Inhalt der XML-Datei beinhalten.

Um die Klassen 'ImagePDFExtractor' und 'TextPDFExtractor' zu verstehen, muss man wissen, dass eine PDF-Datei aus mehreren Ebenen bestehen kann, welche unterschiedliche Informationen beinhalten. Eine Ebene ist die Text-Ebene. Sie enthält Informationen zum tatsächlichen textuellen Inhalt der PDF. Eine andere Ebene ist die Bild-Ebene, welche ausschließlich Bild-Informationen (Farbe, Kompression, etc.) beinhaltet (siehe Abbildung 4.23).

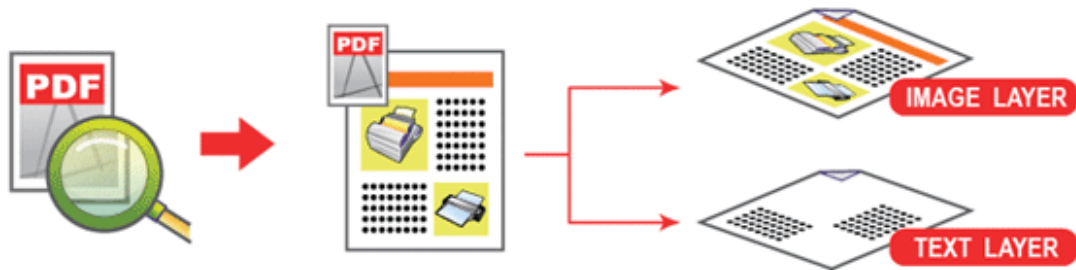


Abbildung 4.23: Ebenen-Aufbau einer PDF-Datei (Quelle: [64])

Welche Ebenen eine PDF-Datei enthält, ist nicht vorgeschrieben und somit wird im 'CEM' zwischen zwei 'Arten' von PDF unterschieden. Eine Art sind PDF's, welche die textuelle Ebene beinhalten und die andere Art sind PDF's, die ausschließlich aus der Bildebene bestehen. PDF-Dateien, die die Textebene beinhalten, werden auch 'druchsuchbare' PDFs genannt. Das kommt daher, dass die Datei in PDF-Viewern nach Text durchsucht werden kann. (vgl. [64] & [65])

Da PDF's intern weitaus komplexer aufgebaut und dadurch deutlich schlechter zu bearbeiten sind als Bild- oder Text-Dateien, werden PDF's zur Bearbeitung umgewandelt. Um unabhängig von der Wahl der Bibliothek zu bleiben, gibt es die zwei Interfaces 'IPDF2ImageConverter' und 'IPDF2TextConverter'. In der aktuellen Version wurde die Bibliothek GhostScript [56] zur Manipulation von PDF-Dateien gewählt. Diese Bibliothek bietet leider keine Funktionalität zur automatischen Detektion der Ebenen einer PDF, weshalb der/die Anwender\*in die Information über ein Flag liefern muss.

Die Klasse 'ImagePDFExtractor' ist für PDF's ohne Textebene und die Klasse 'TextPDFExtractor' für PDF's mit Textebene zuständig. Über die Membervariable 'ImagePDFExtractor.\_imageConversionDPI' kann die Auflösung (DPI) bei der Umwandlung von PDFs in Bilddateien angegeben werden. Standardmäßig sind 300 DPI eingestellt.

Der interne Ablauf des Moduls erfolgt wie in Abbildung 4.24 dargestellt durch ein Aktivitätsdiagramm.

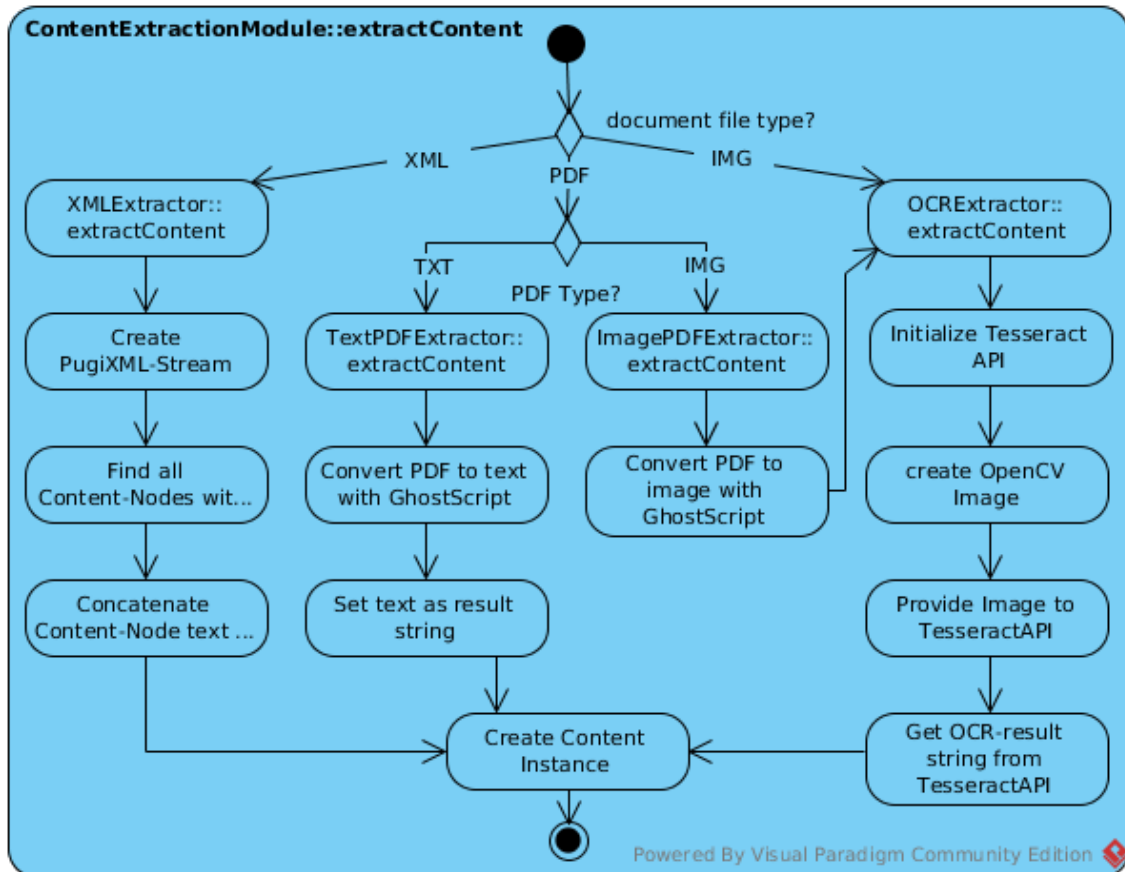


Abbildung 4.24: Aktivitätsdiagramm des des 'Content-Extraction-Module'

#### 4.5.5.2 Content-Normalization-Module ↔ CNM

Die Eingabe des Moduls ist eine 'Content'-Instanz und die Ausgabe eine 'NormalizedContent'-Instanz. Um das Modul möglichst flexibel zu halten wurde das Interface 'INormalizationAlgorithm' für den Algorithmus zur Normalisierung eines Content eingeführt (siehe Abbildung 4.25). Die Nutzung von 'Regular Expressions' [66] bietet sogar noch zusätzliche Flexibilität, denn über die Klasse 'RegexMatchReplacement' lässt sich ein Content durch eine beliebige Anzahl an Ausdrücken normalisieren.

Die Membervariable 'RegexMatchReplacement.\_matcher' stellt einen solchen regulären Ausdruck dar. Wird der Ausdruck gefunden, werden alle Vorkommen durch den String 'RegexMatchReplacement.\_replacement' ersetzt. In der Klasse 'Regex-NormalizationAlgorithm' wird dieses Verhalten mit Hilfe der Boost Bibliothek [54] umgesetzt.

Der Ablauf erfolgt wie in Abbildung 4.26 dargestellt.

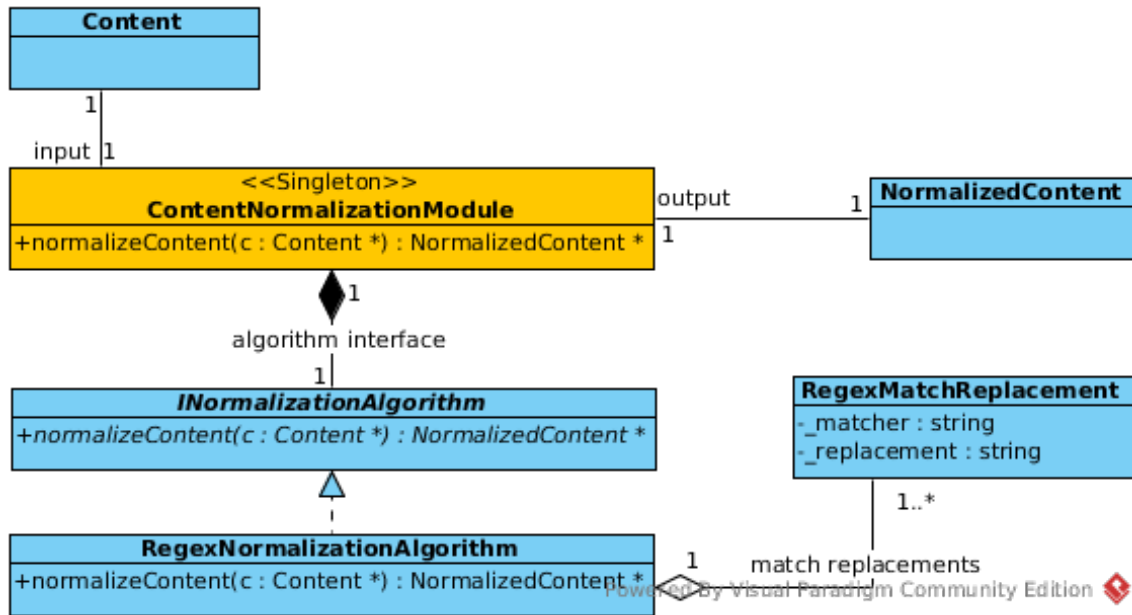


Abbildung 4.25: Klassendiagramm des 'Content-Normalization-Module'

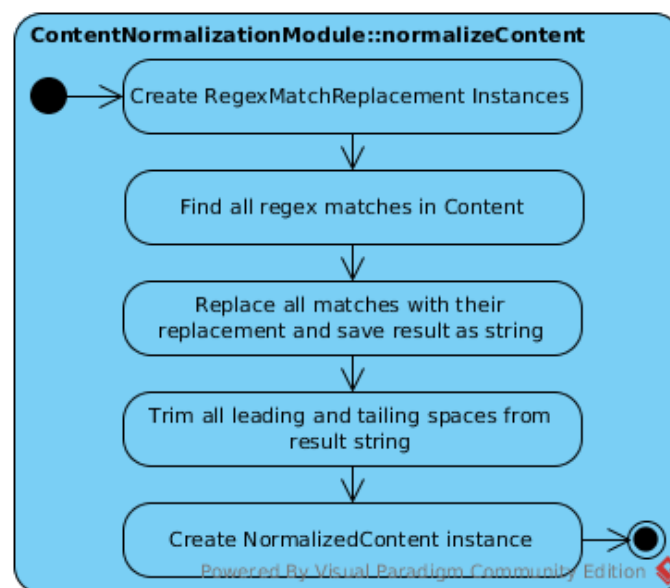


Abbildung 4.26: Aktivitätsdiagramm des 'Content-Normalization-Module'

#### 4.5.5.3 Content-Hash-Generation-Module ↔ CHGM

Ein Objekt der Klasse 'NormalizedContent' ist Eingabeparameter des Moduls. Die Ausgabe ist eine 'ContentHash' Instanz. Ebenfalls aus Flexibilitätsgründen wurde ein Interface 'IContentHashGenerator' zur Generierung von Content-Hashes bereitgestellt (siehe Abbildung 4.27).

In der aktuellen Version wird die OpenSSL-Bibliothek [57] verwendet um Content-Hashes der Typen, die in der Enumeration 'ContentHashType' gelistet sind zu generieren. Dazu muss die Klasse 'OpenSSLContentHashGenerator' mit den gewünschten Eigenschaften des Ausgabe-ContentHash (Pre-

fix und Postfix und Typ) initialisiert werden (siehe Abbildung 4.28).

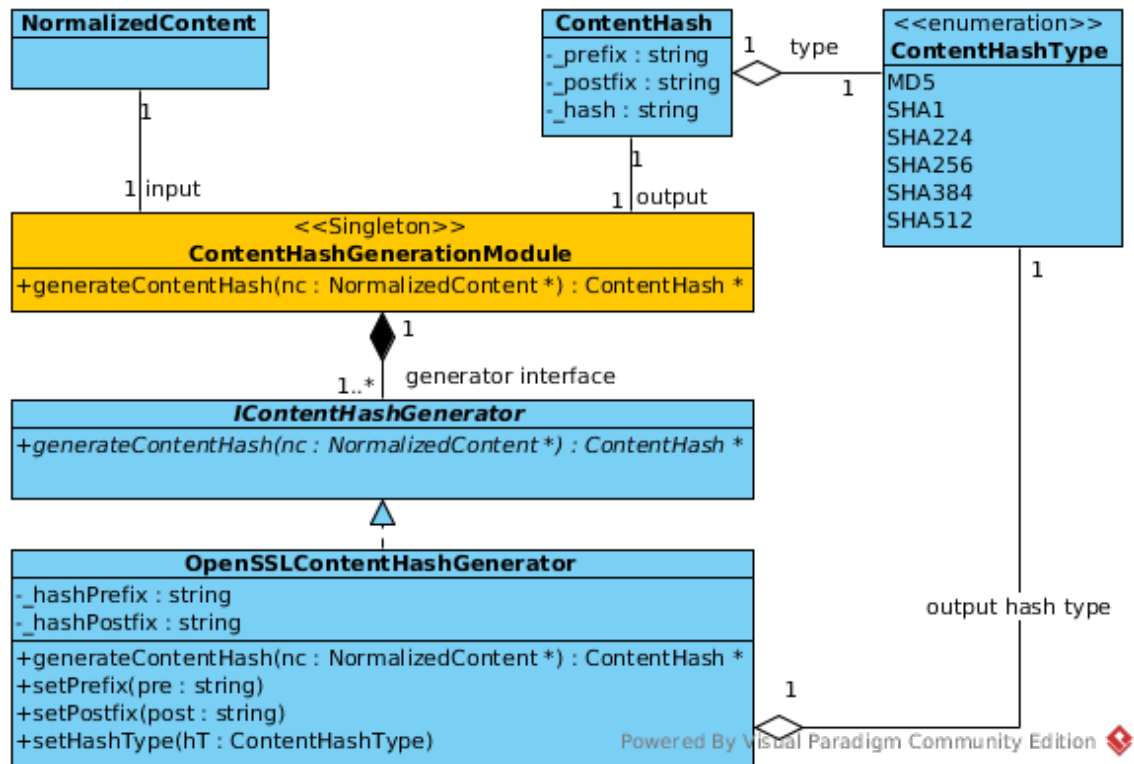


Abbildung 4.27: Klassendiagramm des 'Content-Hash-Generation-Module'

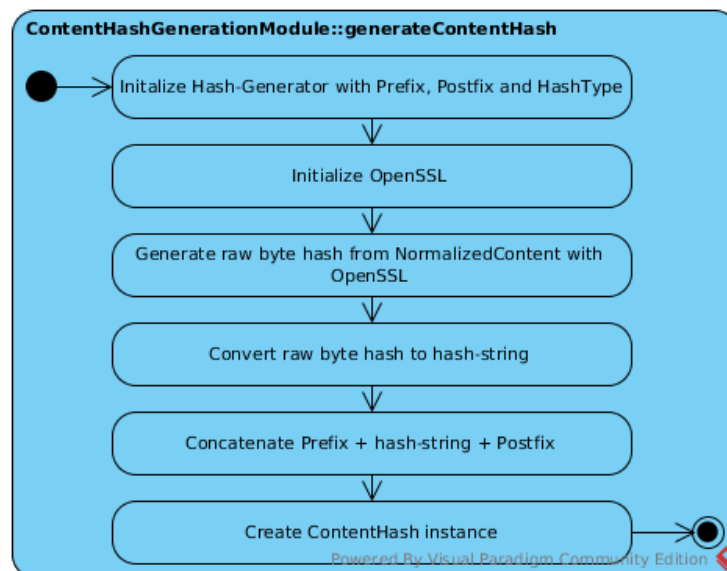


Abbildung 4.28: Aktivitätsdiagramm des 'Content-Hash-Generation-Module'

#### 4.5.5.4 Signature-Generation-Module ↔ SGM

Wieder wurde, um bibliotheksunabhängig und flexibel zu sein, ein Interface 'ISignatureGenerator' zur Generierung einer Signatur bereitgestellt (siehe Abbildung 4.29). Die Klasse 'OpenSSLSignatureGenerator' realisiert, wie schon der Name andeutet, das Interface mit Hilfe der OpenSSL Bibliothek.

Die Methode 'RSAKey.createOpenSSLRSAKeyObject()' ist eine Hilfsfunktion, um ein 'RSAKey'-Objekt in ein Objekt wie es von der OpenSSL-Bibliothek benötigt wird, umzuwandeln. Da dieser Vorgang für RSA-Public-Keys ein anderer ist als für RSA-Private-Keys, ist die Basisklasse 'RSAKey' eine abstrakte Klasse. Die Subklassen 'RSAPublicKey' und 'RSAPrivateKey' beinhalten den entsprechenden Algorithmus.

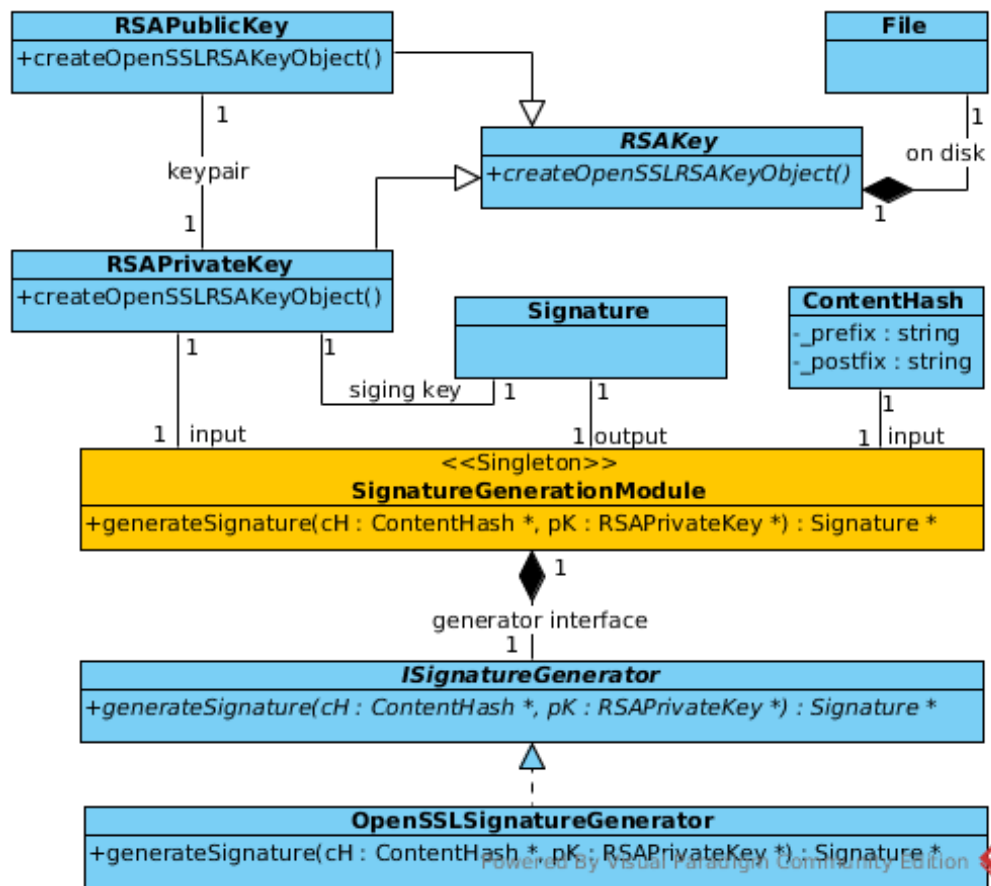


Abbildung 4.29: Klassendiagramm des 'Signature-Generation-Module'

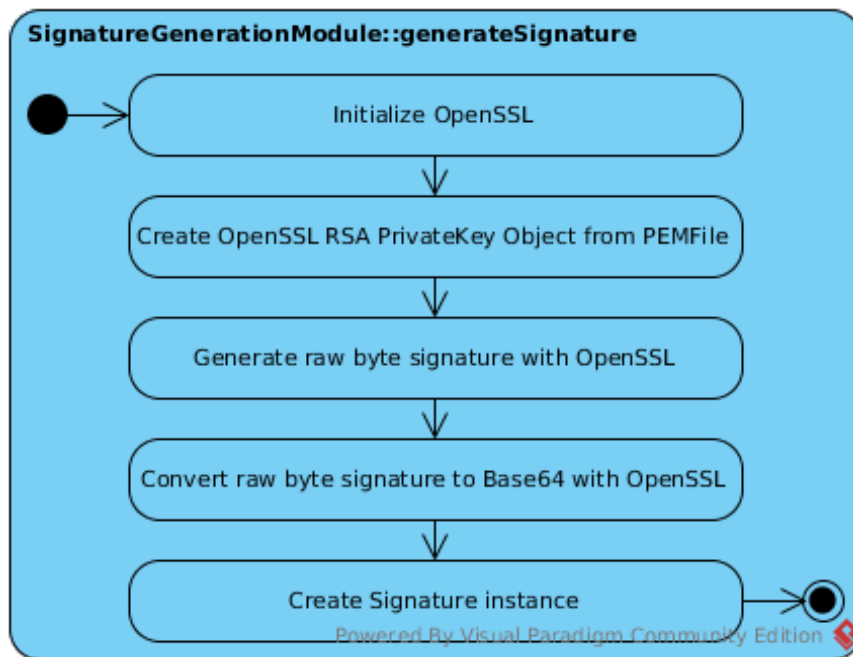


Abbildung 4.30: Aktivitätsdiagramm des 'Signature-Generation-Module'

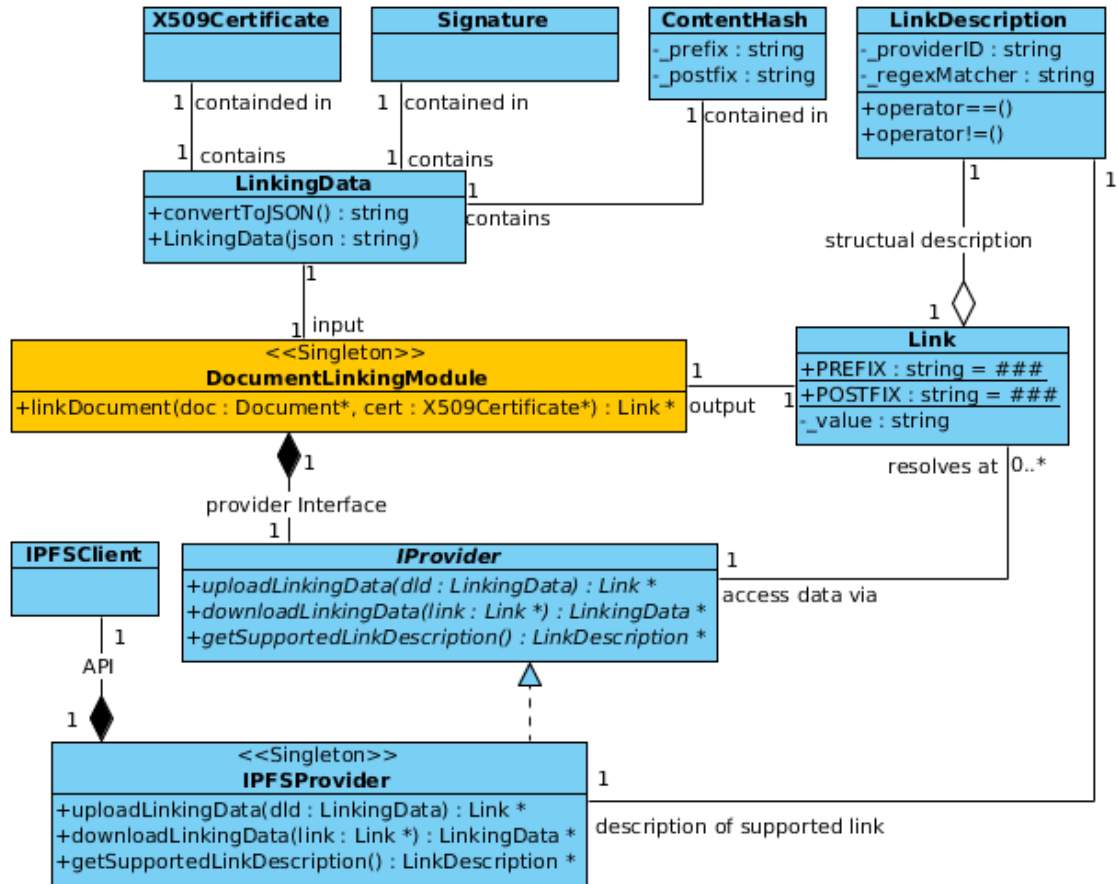
#### 4.5.5.5 Document-Linking-Module ↔ DLM

Als Eingabe dieses Moduls wird ein Objekt der Klasse 'LinkingData' erwartet. Die Ausgabe des 'DLM' ist eine 'Link'-Instanz, über die diese Daten wieder abgerufen werden können.

Um verschiedene Provider zur Verlinkung der Dokumente zu unterstützen, gibt es das Interface 'IProvider' (siehe Abbildung 4.31).

Momentan ist ausschließlich der Provider IPFS in der Klasse 'IPFSProvider' implementiert. Die Klasse wurde als ein 'Singleton' entworfen, da es die IPFS API repräsentiert, welche die Kommunikation mit dem IPFS-Netzwerk realisiert wird. Das Modul kann aber prinzipiell um jeden Provider erweitert werden, der das Hoch- und Runterladen von JSON- beziehungsweise Textdateien unterstützt, da die Klasse 'LinkingData' von und nach JSON umgewandelt wird. Eine Liste von Providern die die Anforderungen erfüllen, findet sich in Abschnitt 4.2.2.1. Die JSON-Struktur eines serialisierten 'LinkingData' Objekts entspricht der Form wie in Listing 4.4. Das Schlüsselwort '<string>' drückt aus, dass der Typ des Feldes als String interpretiert wird.

Der Ablauf erfolgt wie in Abbildung 4.32. Wurde die Datei erfolgreich hochgeladen, wird ein Link, mit dem die Daten bei IPFS abrufbar sind, erstellt.



Powered By Visual Paradigm Community Edition

Abbildung 4.31: Klassendiagramm des 'Document-Linking-Module'

```

{
  "Certificate": {
    "Data": "<string>",
    "Type": "<string>"
  },
  "ContentHash": {
    "Data": "<string>",
    "Postfix": "<string>",
    "Prefix": "<string>",
    "Type": "<string>"
  },
  "DocumentName": "<string>",
  "Signature": {
    "Data": "<string>",
    "Encoding": "<string>"
  }
}

```

Listing 4.4: JSON-Struktur eines serialisierten Objekts der Klasse 'LinkingData'



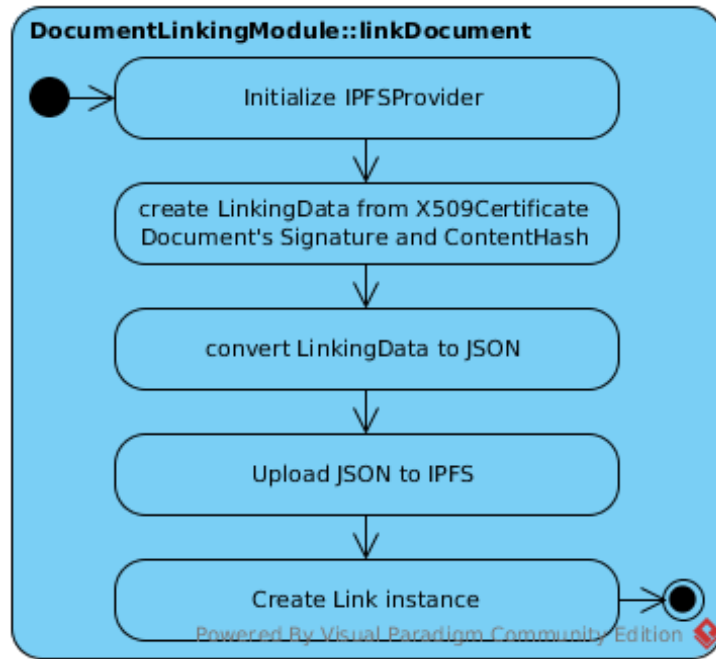


Abbildung 4.32: Aktivitätsdiagramm des 'Document-Linking-Module'

#### 4.5.5.6 Link-Injection-Module ↔ LIM

Das 'Link-Injection'-Modul benötigt ein Objekt der Klasse 'Link' und ein Objekt der Klasse 'Document' als Eingabe. Der 'Link' wird durch das Modul in eine Kopie des 'Document' eingefügt. Diese Kopie wird auf der Festplatte gespeichert und ist somit die Ausgabe des Moduls (wenn auch nur der Pfad, unter welchem die Ausgabedatei zu finden ist ausgegeben wird.)

Je nach Datei-Typ des Dokuments muss unterschiedlich vorgegangen werden, um einen Link in einem Dokument zu platzieren. Durch die Anwendung des 'Strategy-Pattern' in der Architektur des 'Link-Injection'-Moduls kann zur Laufzeit der zum Dokumententyp passende Algorithmus ausgewählt werden.

Das Basis Interface der Strategien ist 'ILinkInjector' (siehe Abbildung 4.33).

Die Klasse 'ImagaInjector' realisiert die Injektion des Link-Textes in eine Bilddatei mit Hilfe der OpenCV Bibliothek. Über die Membervariable 'ImagaInjector.\_injectQR' kann gesteuert werden, ob ein Link als QR-Code oder als Text in ein Bild eingefügt werden soll. Mit den restlichen Membervariablen der Klasse werden Eigenschaften des QR-Codes beziehungsweise Link-Textes gesteuert. Um einen QR-Code zu erzeugen, wird der 'Project Nayuki QR-Code Generator' [61] für C++ genutzt. Um einen Text oder QR-Code in ein Bild einzufügen wurde die Bibliothek OpenCV [55] verwendet. Mit der Klasse 'PDFInjector' wird ein Link in ein PDF-Dokument injiziert. Aufgrund des komplexen internen Aufbaus einer PDF-Datei wird der Link nicht direkt in die Datei eingefügt sondern es wird die PDF vorher in ein Bild umgewandelt. Der Link wird in das Bild eingefügt und danach wird das Bild mit dem Link wieder zurück in eine PDF transformiert. Die Umwandlung vom PDF- in ein Bildformat geschieht durch die Klasse 'GhostScriptManipulator'. Um ein Bild wieder in eine PDF zu transformieren, wird die Klasse 'SimplePythonImage2PDFConverter', welche das Konvertierungsinterface 'IImage2PDFConverter' realisiert, genutzt. Diese Klasse nutzt die Python-Schnittstelle für C / C++ [67].

Mit Hilfe dieser API wird ein in den C++ Code eingebetteter Python Code ausgeführt, der das Pythonpaket 'img2pdf' [60] ansteuert. Dieses Paket kann verlustfrei Rasterbilder in PDFs transformieren. Der grobe interne Ablauf wird über ein Aktivitätsdiagramm in Abbildung 4.34 dargestellt.

Um einen Link in eine XML-Datei einzufügen, wird die Klasse 'XMLInjector' verwendet. Diese realisiert den Algorithmus mit Hilfe der PugiXML-Bibliothek. Die Membervariable 'XMLInjector.\_injectionElementNodeNames' ist eine Liste mit den Namen der XML-Elementen, welche in das Dokument eingefügt werden sollen. Das letzte Element der Liste enthält den Link-Text als PC\_DATA. Über 'XMLInjector.\_injectionRootElementXPath' wird dem Algorithmus über einen XPath-Ausdruck mitgeteilt, an welcher Stelle im XML-Dokument die Elemente eingefügt werden sollen.

Die Klasse 'TextInjector' enthält den Algorithmus, um einen Link in Textdokumente einzufügen. Ist die Membervariable 'TextInjector.\_injectAtTail' 'true', wird der Link-Text am Ende der Datei angefügt, steht die Variable auf 'false' wird der Text am Anfang des Dokuments eingefügt.

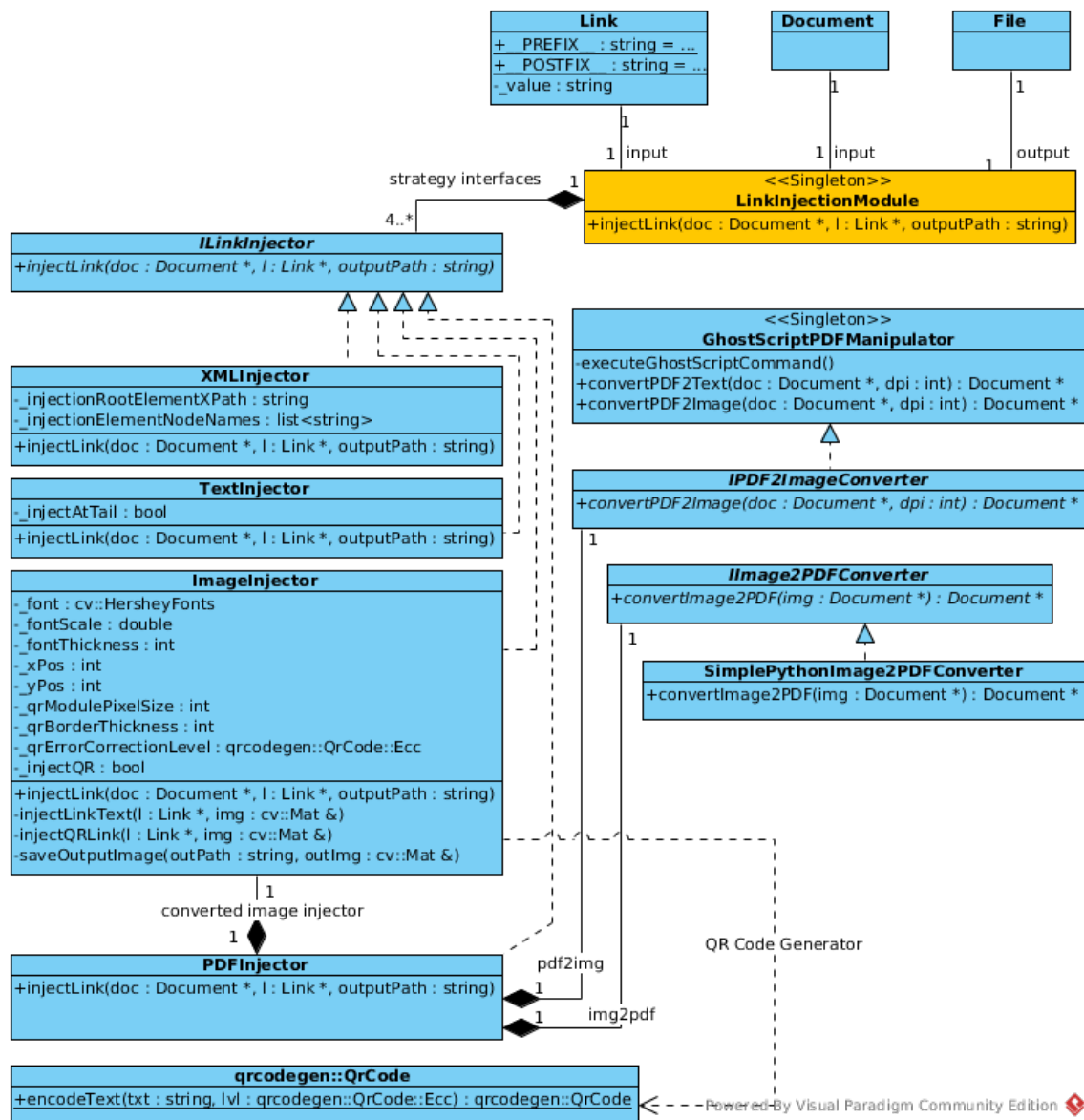


Abbildung 4.33: Klassendiagramm des 'Link-Injection-Module'

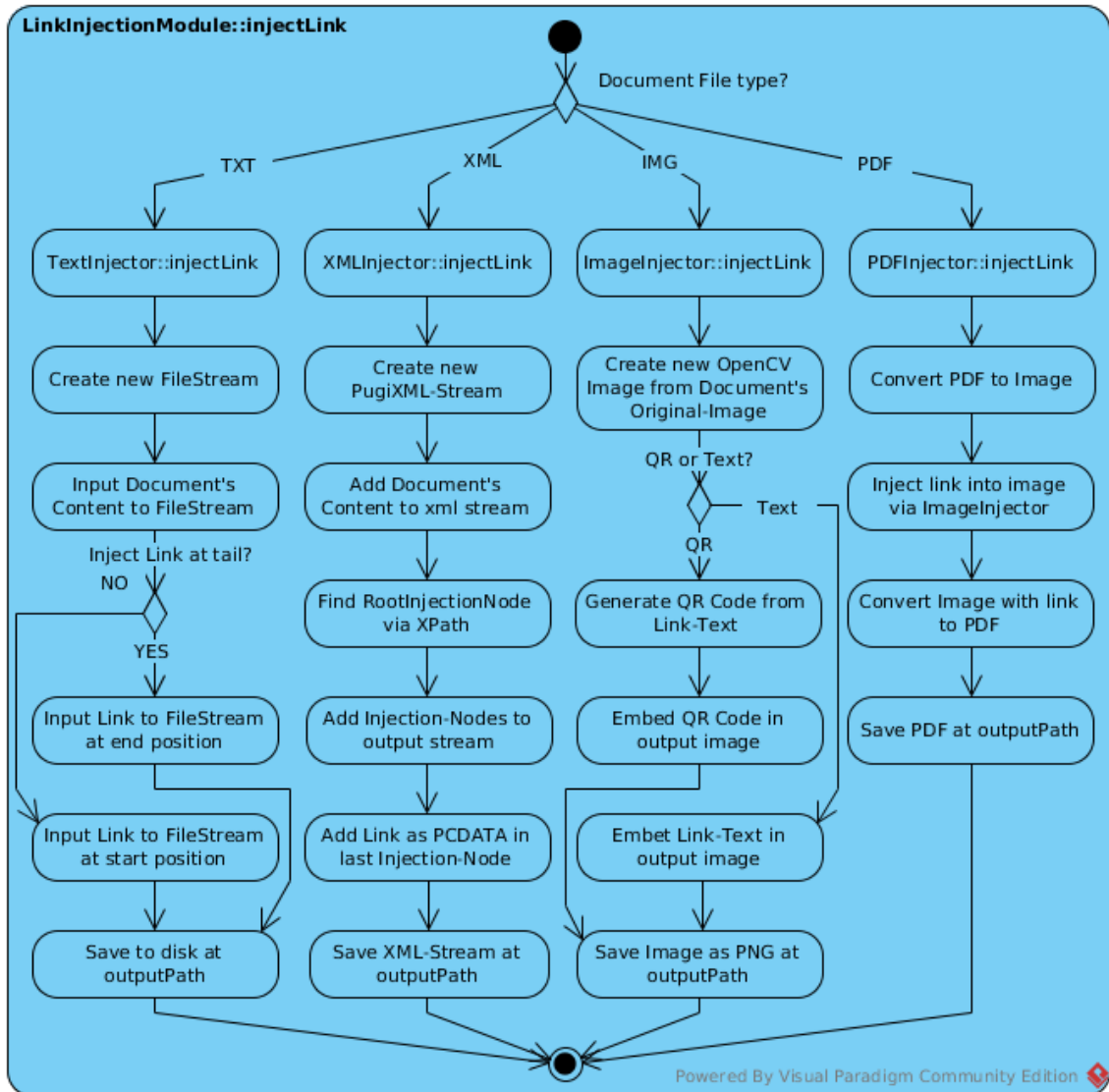


Abbildung 4.34: Aktivitätsdiagramm des 'Link-Injection-Module'

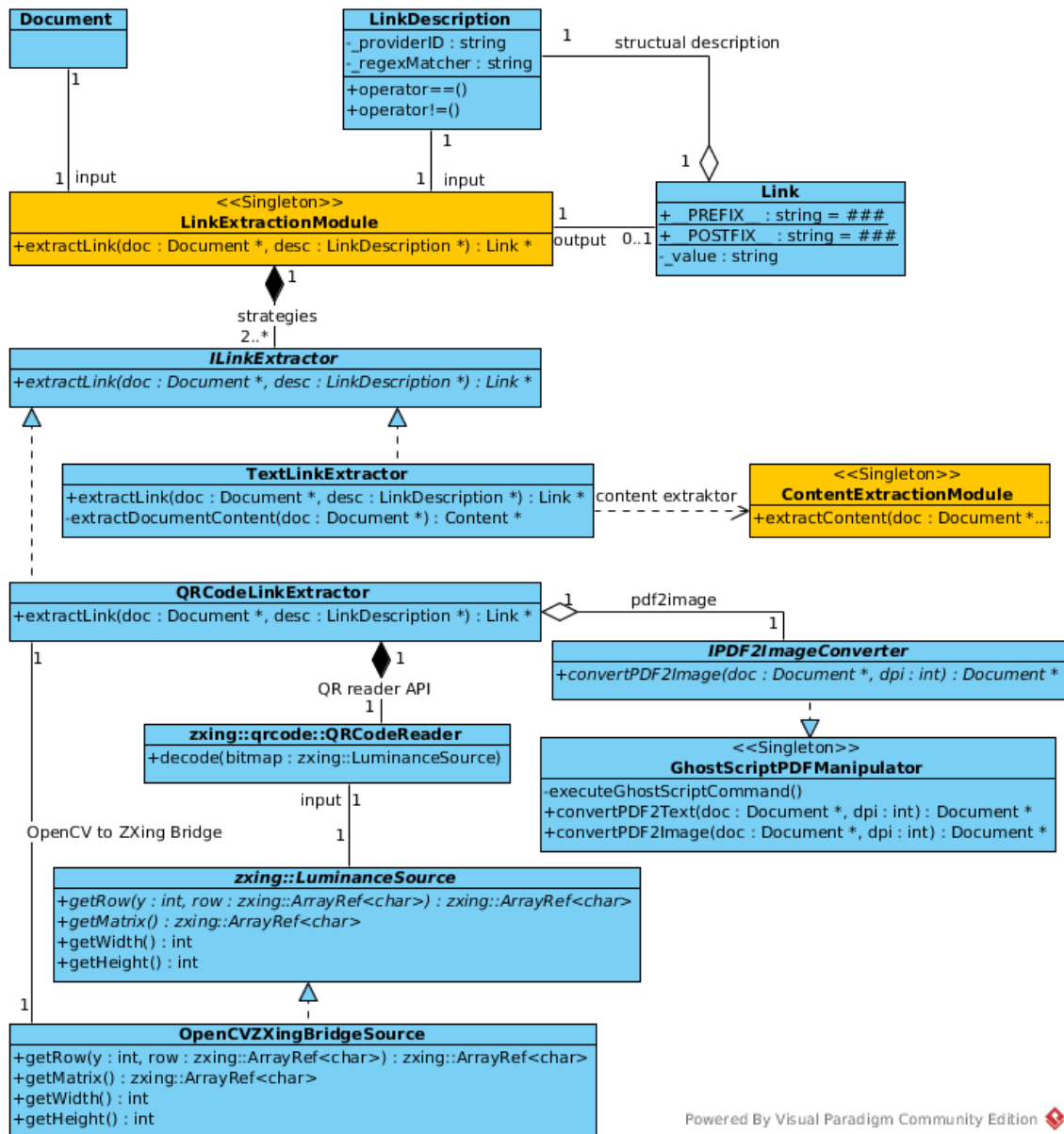
#### 4.5.5.7 Link-Extraction-Module ↔ LEM

Die Eingabeparameter dieses Moduls sind jeweils eine Instanz der Klassen 'Document' und 'LinkDescription'. Die Ausgabe stellt eine Instanz der Klasse 'Link' dar. Um die Möglichkeit zu bieten, den Link aus einem Dokument auf verschiedene Arten zu extrahieren, wurde das 'Strategy'-Pattern angewandt. Das Interface 'ILinkExtractor' in der Architektur des 'Link-Extraction'-Moduls dient als Basisklasse der Strategien.

Die Klasse 'TextLinkExtraktor' realisiert dieses Interface mit einem Algorithmus, welcher auf regulären Ausdrücken aus der Boost Bibliothek [54] basiert. Der Regex-Ausdruck, der gebraucht wird, um einen Link zu finden, wird aus den Informationen aus der 'LinkDescription' Klasse und den statischen und konstanten Mitgliedern 'Link.\_\_PREFIX\_\_' und 'Link.\_\_POSTFIX\_\_' generiert. Da reguläre Ausdrücke nur auf Strings arbeiten können, muss, falls nicht schon geschehen, der Inhalt des Dokuments erst extrahiert werden, was mit Hilfe des 'Content-Extraction'-Moduls geschieht. Da dieses Modul alle Dokumententypen unterstützt, kann der 'TextLinkExtraktor' ebenfalls mit allen Arten von Dokumenten, die das 'CEM' unterstützt arbeiten.

Da ein QR-Code nur in Bild- oder PDF-Dokumente eingefügt werden kann, kann auch die Klasse 'QRCodeLinkExtractor' nur mit diesen Dokumententypen arbeiten. Um einen QR-Code auf einem Bild ausfindig zu machen und zu dekodieren, wird die C++ Portierung der (Java) Bibliothek ZXing [62] verwendet. Da zur Bildverarbeitung die Bibliothek OpenCV [55] genutzt wird, muss eine sogenannte 'Bridge'-Klasse implementiert werden. Mit dieser Klasse ist es möglich ein OpenCV Bild der ZXing Bibliothek zur Verfügung zu stellen. Das Interface hierfür ist 'zxing::LuminanceSource', welches durch die Klasse 'OpenCVZXingBridgeSource' realisiert ist. Handelt es sich beim Dokument um eine PDF-Datei, muss die PDF erst in ein Bild-Dokument konvertiert werden. Dies geschieht mit der Klasse 'GhostScriptPDFManipulator', welcher das allgemeine Konvertierungsinterface 'IPDF2ImageConverter' realisiert.

Eine etwas genauerer Beschreibung des internen Vorgangs ist in Abbildung 4.36 zu sehen.



Powered By Visual Paradigm Community Edition

Abbildung 4.35: Klassendiagramm des 'Link-Extraction-Module'

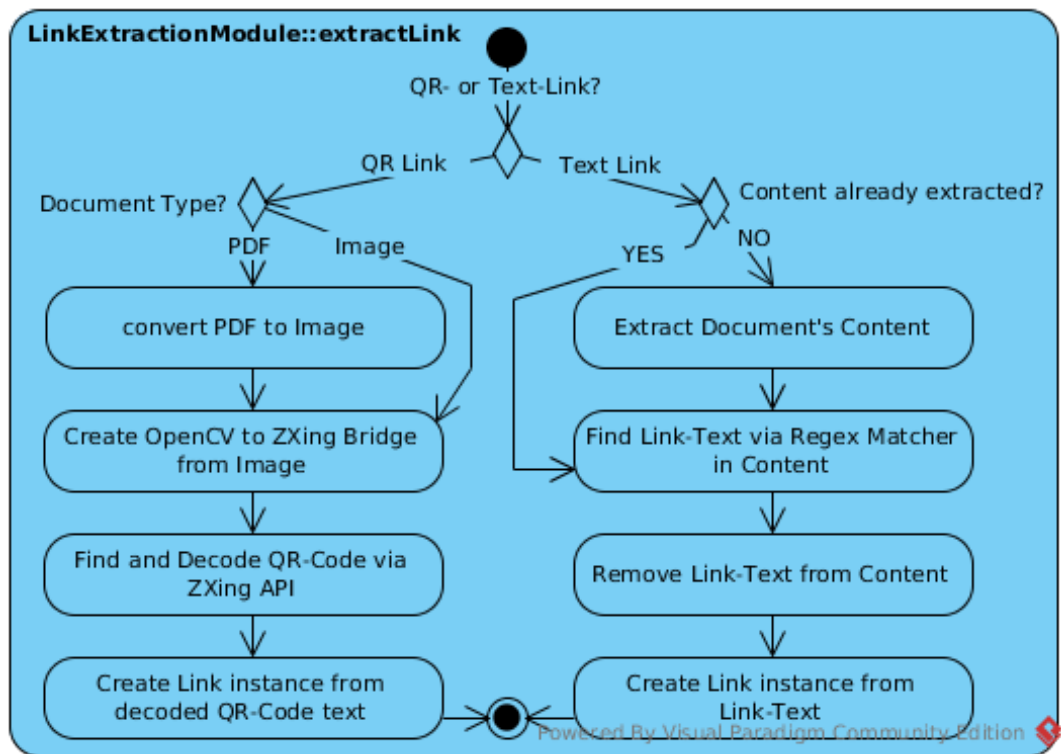


Abbildung 4.36: Aktivitätsdiagramm des 'Link-Extraction-Module'

#### 4.5.5.8 Linking-Data-Loader-Module ↔ LDLM

Die Architektur des 'Linking-Data-Loader'-Modul ist sehr ähnlich wie die des 'Document-Linking'-Modul. Es wird wieder das Interface 'IProvider' genutzt um verschiedene Verlinkungs-Provider zu unterstützen. Über einen Link als Eingabe wird in der Klasse 'IPFSProvider' der JSON-Formatierte String aus dem IPFS-Netzwerk heruntergeladen (siehe Abbildung 4.38). Die Ausgabe ist eine Instanz der Klasse 'LinkingData'.

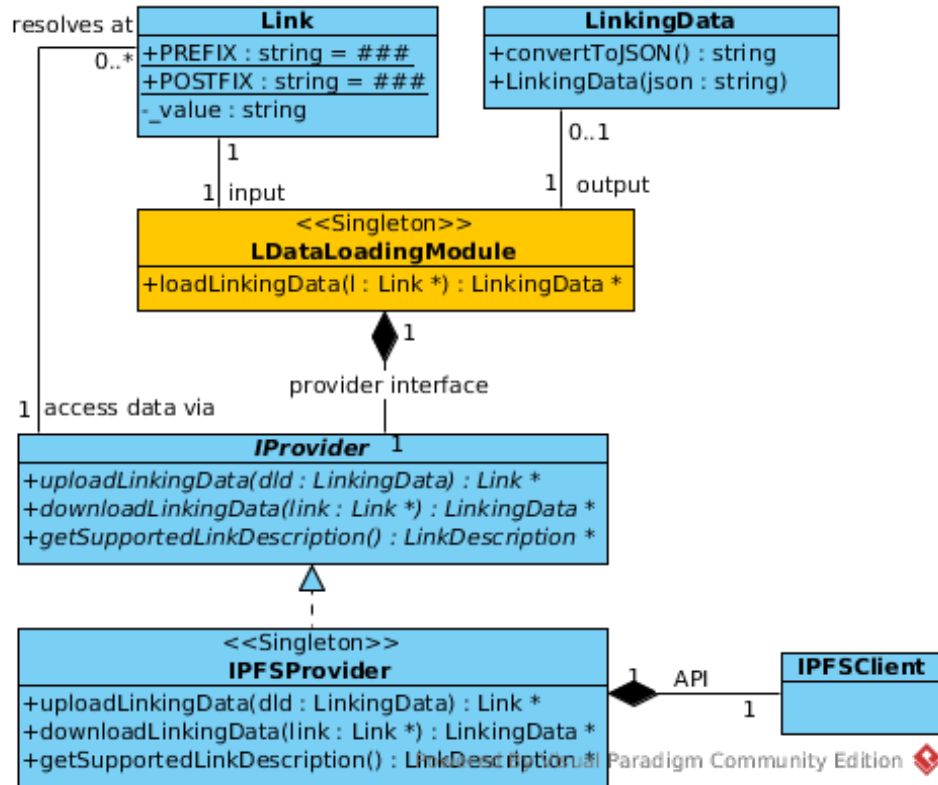


Abbildung 4.37: Klassendiagramm des 'Linking-Data-Loading-Module'

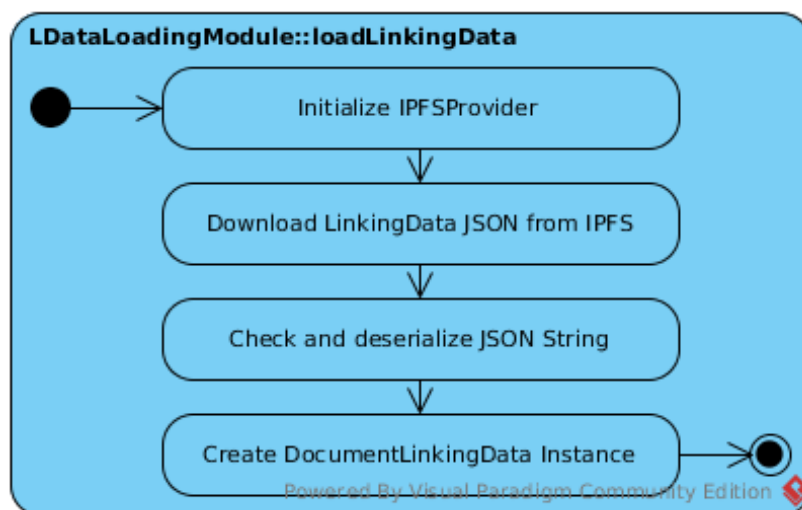


Abbildung 4.38: Aktivitätsdiagramm des 'Linking-Data-Loading-Module'

#### 4.5.5.9 Content-Validation-Module ↔ CVM

Dieses Module ist, wie man unschwer an dessen Aufbau sehen kann, das einfachste Modul im 'Printed Document Linking System'. Da auch der Ablauf trivial ist, wird er an dieser Stelle textuell beschrieben anstatt über ein Aktivitätsdiagramm wie bei den anderen Modulen.

Es wird lediglich ein Vergleich der Membervariablen der zwei Eingabe-'ContentHash'es gemacht. Sind alle Member gleich, ist der Ausgabe 'ValidationState' 'VALID' und wenn nicht 'INVALID'.

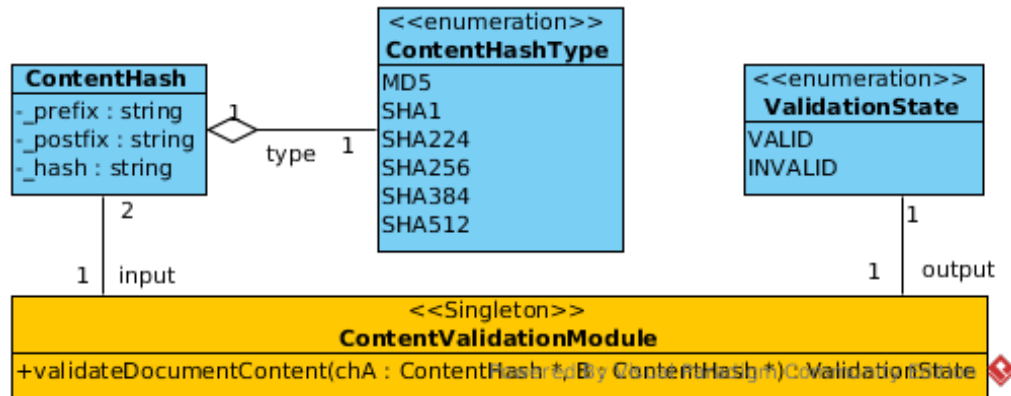


Abbildung 4.39: Klassendiagramm des 'Content-Validation-Module'

#### 4.5.5.10 Signature-Verification-Module ↔ SVM

Damit das 'Signature-Verification'-Modul eine Signatur verifizieren kann, werden jeweils ein Objekt der Klassen 'X509Certificate', 'Signature' und 'ContentHash' (siehe Abbildung 4.40) als Eingabedaten benötigt. Eine 'ContentHash'-Instanz wird benötigt weil dies die 'signierte Nachricht' ist, welche benötigt wird, um die Signatur zu überprüfen. Als Ausgabe hat das Modul einen 'SignatureVerificationState', welcher das Gesamtergebnis der Überprüfung widerspiegelt. Da um eine Signatur zu prüfen auch das zugehörige Zertifikat überprüft werden muss, gibt es in der Architektur des 'Signature-Verification'-Modul die zwei Interfaces 'ISignatureValidator' und 'ICertificateValidator'.

In der aktuellen Version werden nur sogenannte 'self-signed certificates' [68] unterstützt, da es sich beim 'Printed Document Linking System' nur eine 'Proof-Of-Concept'-Implementierung handelt. Diese Art von Zertifikat wird durch die Klasse 'OpenSSLSelfSignedX509Validator' mit Hilfe der OpenSSL Bibliothek überprüft. Als Ergebnis wird ein 'ValidationState' geliefert.

Der Algorithmus, der in der Klasse 'OpenSSLSignatureValidator' realisiert ist, überprüft eine Signatur (mit OpenSSL) und liefert ebenfalls als Ergebnis einen 'ValidationState'. Das Gesamtergebnis wird ermittelt in dem die beiden Zwischenergebnisse kombiniert werden. Der Ablauf des Moduls ist in 4.41 zu sehen.



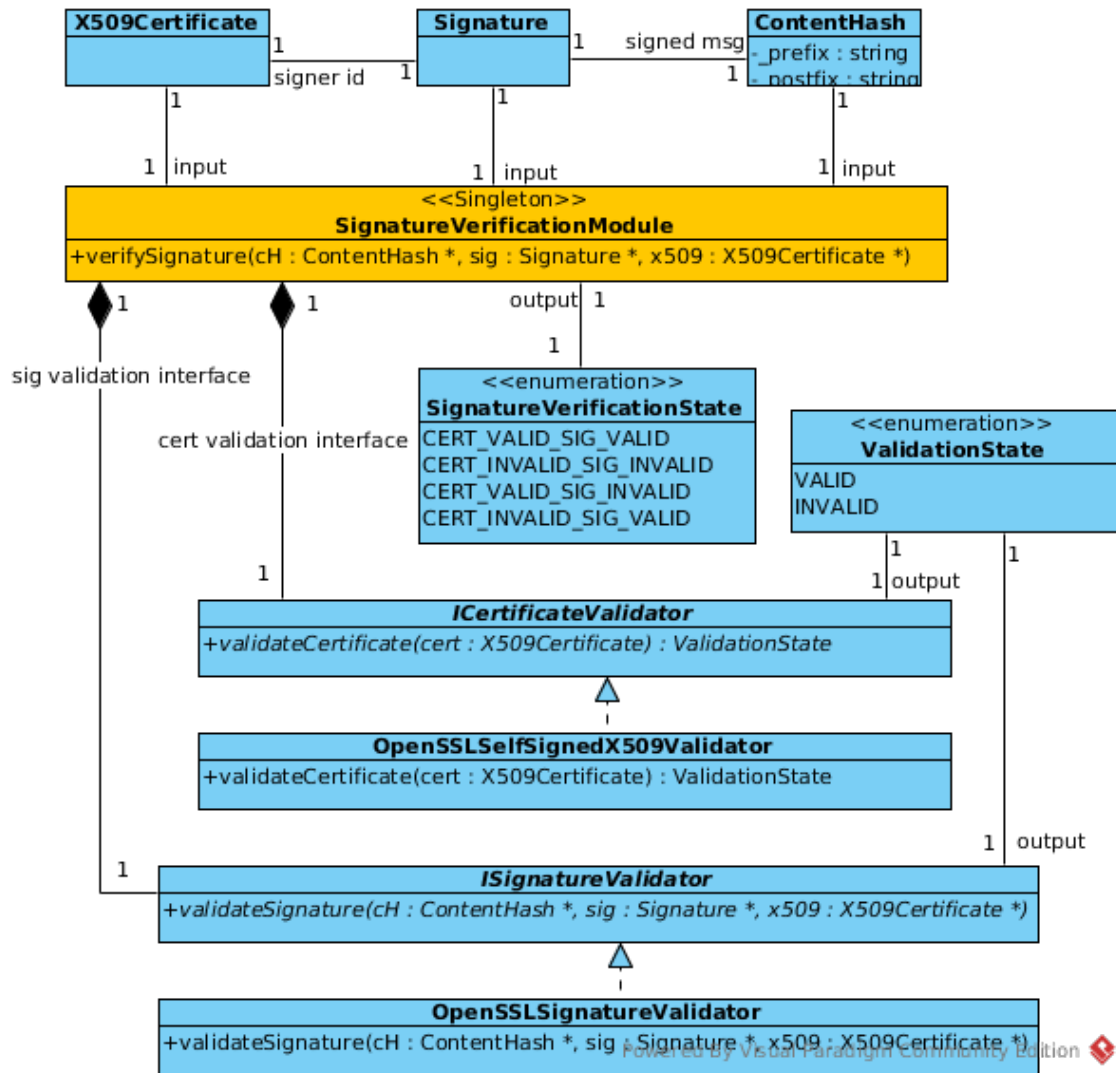


Abbildung 4.40: Klassendiagramm des 'Signature-Verification-Module'

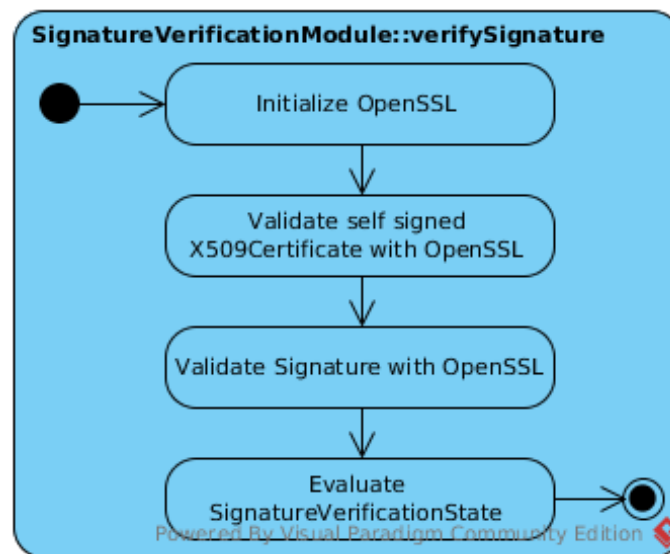


Abbildung 4.41: Aktivitätsdiagramm des 'Signature-Verification-Module'

# Kapitel 5

## Ergebnisse

## 5.1 Überprüfung der Anforderungen

In diesem Abschnitt werden die Anforderungen aus Abschnitt 4.1.2, die aus der Anforderungsanalyse aus Abschnitt 4.1 entstanden sind, überprüft. Zuerst werden die Anforderungen einzeln betrachtet und dann, in einem Fazit eine kurze Übersicht der Ergebnisse aller Anforderungen gezeigt.

### 5.1.1 Anforderungen des Typs 'Frage'

#### 5.1.1.1 Anforderung R1 - Wie stabil ist die OCR Erkennung?

Auf die Frage 'Wie stabil ist die OCR-Erkennung?' (Anforderung R1) kann nur geantwortet werden, dass die Erkennung unter den in Abschnitt 4.3.1 beschriebenen Testbedingungen von komplexeren Dokumenten sehr instabil ist und sich so nicht für den Einsatz eignet.

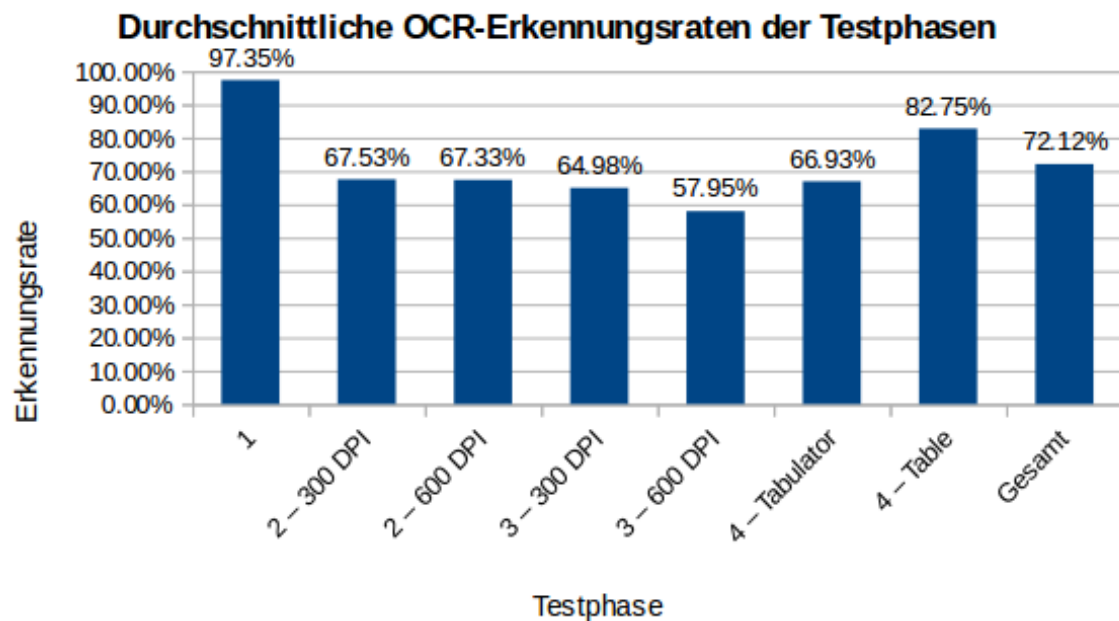


Abbildung 5.1: Säulendiagramm der durchschnittlichen OCR-Erkennungsraten aus den verschiedenen OCR-Testphasen

Insgesamt wurde wie in Abbildung 5.1 dargestellt, in allen OCR-Testphasen eine gesamte durchschnittliche OCR-Erkennungsrate von nur 72.12% erreicht. Was ebenfalls auffällt, ist der positive Ausreißer OCR-Testphase 1 (siehe Abschnitt 4.3.2.1) mit 97.35%. Dieses positive Ergebnis wurde erreicht, weil es sich um sehr simple Testdokumente im Vergleich zu den Testdokumenten aus den anderen Testphasen handelt. Gerade bei der Berechnung des Content-Hash ist eine 100%ige Erkennungsrate notwendig, da sich bei kleinster Abweichung des Inhalts beziehungsweise des OCR-Ergebnisses der Content-Hash stark verändert.

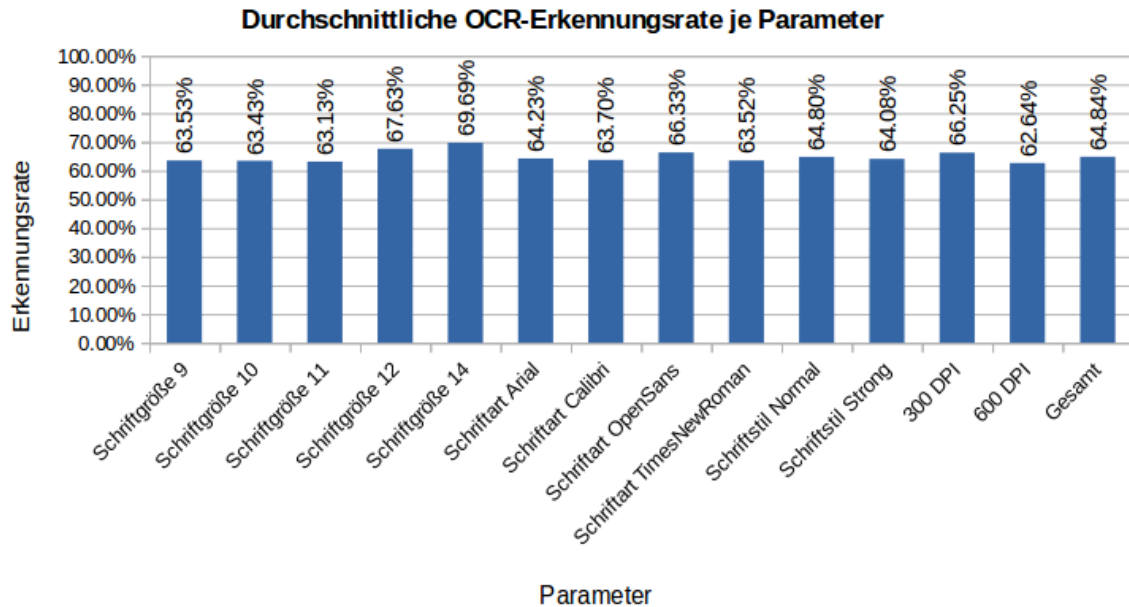


Abbildung 5.2: Säulendiagramm der durchschnittlichen OCR-Erkennungsraten aus den OCR-Testphasen 2 bis 4 bezogen auf die Variationsparameter der Testdokumente

In Abbildung 5.2 sind die durchschnittlichen Erkennungsraten der einzelnen Variationsparameter aus den OCR-Testphasen 2 bis 4 bezogen auf die Variationsparameter der Testdokumente zu sehen. Das heißt, in dieses Diagramm sind die einfachen Testdokumente aus OCR-Testphase 1 nicht miteinbezogen worden, was eine realistischere Aussage für das Vorhaben dieser Arbeit bietet. Das Diagramm ist folgendermaßen zu lesen: Dokumente mit Schriftgröße 9 haben durchschnittlich 63.53% erreicht, mit Schriftgröße 10 63.43% und so weiter. Der Parameter 'Gesamt' meint alle Parameter zusammen, also die durchschnittliche Erkennungsrate aller Dokumente der Testphasen.

Wie bereits erwähnt, eignet sich der in dieser Arbeit getestete Algorithmus zur OCR-Erkennung für das entwickelte Konzept (siehe Abschnitt 4.4) nicht. Vorschläge und Ideen zur Verbesserung der OCR Ergebnisse befinden sich im Abschnitt 6.2.1.

#### **5.1.1.2 Anforderung R2 - Wie kann ein Dokument in einer DHT oder Blockchain referenziert werden?**

Um die Frage 'Wie kann ein Dokument in einer DHT oder Blockchain referenziert werden?' zu beantworten wurde ein Konzept entwickelt und die Recherche aus Abschnitt 4.2.2 durchgeführt.

Das entwickelte Konzept, welches in der 'Proof-Of-Concept'-Implementierung aus Abschnitt 4.5 umgesetzt wurde, ist in Abschnitt 4.4 beschrieben.

Kurz zusammengefasst lässt es sich wie folgt ausdrücken: Der Content-Hash, die Signatur und das zugehörige Zertifikat eines Dokuments werden in einer JSON-Datei gebündelt. Diese Datenbündelung wird in dieser Arbeit 'Verlinkungsdaten' eines Dokuments genannt. Diese Verlinkungsdaten werden im JSON-Format auf den Verlinkungsanbieter IPFS hochgeladen. Der Link, welcher auf die Datei zeigt, wird in das Dokument eingefügt. Ein Dokument, welches einen solchen Link enthält, wird als verlinkt bezeichnet und kann dadurch verifiziert werden.

#### **5.1.1.3 Anforderung R3 - Können Kopien von Dokumenten über die gleiche ID referenziert werden?**

Ja, verschiedene Kopien können prinzipiell durch die gleiche ID referenziert werden. Die ID entspricht in dem in dieser Arbeit vorgeschlagenen Konzept einem IPFS-Link, welcher auf die JSON-formatierten Verlinkungsdaten eines Dokuments zeigt.

Da IPFS-Links aus 'Merkle-Tree-Hashes'[38] bestehen, welche aus dem Inhalt der referenzierten Datei generiert werden, wird für Dateien mit gleichem Inhalt immer derselbe Hash und damit Link generiert werden. Das heißt, wenn der (extrahierte) Inhalt einer Kopie der gleiche ist (wie bei einer anderen Kopie oder dem Originaldokument), und der Unterzeichner des Dokuments beziehungsweise der private Schlüssel und das Zertifikat, welche benutzt wurden, um die Signatur zu erzeugen, die gleichen sind, sind auch die Verlinkungsdaten und damit der IPFS-Link (= ID), der auf die Daten zeigt, gleich (wie im Originaldokument).

#### **5.1.1.4 Anforderung R4 - Kann ein Dokument über seinen Content-Hash referenziert werden?**

Da die Links bei IPFS, wie bereits erwähnt aus 'Merkle-Tree-Hashes' bestehen, welche aus dem Inhalt einer Datei generiert werden, ist es nicht möglich eigene Schlüssel (in dem Fall Content-Hashes) zur Referenzierung von Dokumenten zu verwenden. David Wheeler hat hier jedoch mit seinem berühmten Aphorismus 'All problems in computer science can be solved by another level of '[69] einen Lösungsvorschlag gemacht, der immer angewendet werden kann.

Was also gemacht werden könnte, ist ein Mapping im System zu implementieren, bei dem Content-Hashes auf IPFS-Links abgebildet werden.

#### **5.1.1.5 Anforderung R5 - Wie kann eine Abbildung vom Content-Hash auf den Binär-Hash eines Dokuments verwaltet werden?**

Zum Start dieser Bachelorarbeit war die Idee, dass von jeder Kopie eines Master-Dokuments beziehungsweise Originaldokument, ein Binary-Hash berechnet wird. Dabei entstand die Frage, wie eine Abbildung des Content-Hash eines Master-Dokuments auf eine Liste der Binary-Hashes der verschiedenen Kopien verwaltet werden kann. Dies ist im entwickelten Konzept nicht mehr notwendig, da jede Kopie

einfach den Link zum Dokument beinhaltet. Deshalb wurde im Laufe der Arbeit entschieden, dieses Problem nicht weiter zu untersuchen.

## 5.1.2 Anforderung des Typs 'Funktionalität'

### 5.1.2.1 Anforderung R6 - Extraktion des Inhaltes eines Dokuments

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Durch das 'Content-Extraction'-Modul kann der Inhalt aus Dokumenten auf verschiedene Arten, je nach Dokumententyp, extrahiert werden. Architektur sowie Funktionsweise werden im Abschnitt 4.5.5.1 erklärt.
<b>Anmerkungen</b>	<ul style="list-style-type: none"> <li>• Wenn der Inhalt einer PDF-Datei extrahiert werden soll, wird die PDF in eine Bild- beziehungsweise Textdatei umgewandelt. Die Extraktion erfolgt dann aus den umgewandelten Dateien.</li> <li>• Bei Bild- beziehungsweise PDF-Dateien wird der Inhalt über einen OCR-Algorithmus extrahiert. Dabei sind die Probleme und Einschränkungen aus Abschnitt 5.1.1.1 zu beachten!</li> </ul>

*Tabelle 5.1: Ergebnis der Anforderung R7*

### 5.1.2.2 Anforderung R7 - Normalisierung des Inhaltes eines Dokuments

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Durch das, in Abschnitt 4.5.5.2 vorgestellte, 'Content-Normalization'-Modul kann ein Inhalt mithilfe von Regex-Ausdrücken normalisiert werden. In der aktuellen Version wird als erstes jedes Zeichen, welches nicht zu einem Wort gehört (reguläre Ausdruck: '\s'), durch einen 'Newline-Character' ersetzt. Als nächstes werden alle mehrfachen Vorkommen eines 'Newline-Character's (reguläre Ausdruck: '(\s*\n){2,}') durch einen Einzelnen ersetzt. Das Ergebnis ist eine Liste aller Wörter des Dokuments.
<b>Anmerkungen</b>	Das Vorgehen ist sehr praktisch, da beim Extrahieren des Inhaltes aus einer XSL.FO-Datei alle Elemente nacheinander extrahiert werden, das heißt es entsteht automatisch eine Liste aller Wörter. Wird diese XSL.FO-Datei später in eine PDF-Datei umgewandelt und der Inhalt der PDF-Datei (wo natürlich nicht jedes Wort in einer Liste nacheinander kommt, sondern der Text formatiert ist) extrahiert, ist der normalisierte Inhalt und damit der Content-Hash der gleiche.

*Tabelle 5.2: Ergebnis der Anforderung R7*

### 5.1.2.3 Anforderung R8 - Erzeugung des Content-Hash eines Dokuments

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Durch das, in Abschnitt 4.5.5.3 vorgestellte, 'Content-Hash-Generation'-Modul kann aus einem normalisierten Inhalt ein kryptographischer Hash mithilfe der OpenSSL-Bibliothek generiert werden.
<b>Anmerkungen</b>	<p>Je nach Wunsch des/der Nutzers*in kann einer der folgenden Hash-Typen generiert werden.</p> <ul style="list-style-type: none"> <li>• MD5</li> <li>• SHA1</li> <li>• SHA224</li> <li>• SHA256</li> <li>• SHA384</li> <li>• SHA512</li> </ul> <p>Außerdem kann ein benutzerdefinierter Prefix und Postfix des Hashes angegeben werden.</p>

*Tabelle 5.3: Ergebnis der Anforderung R8*

### 5.1.2.4 Anforderung R9 - Erzeugen einer digitalen Signatur eines Dokuments

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Durch das 'Signature-Generation'-Modul kann eine digitale Signatur für ein Dokument erstellt werden. Dafür muss ein privater Schlüssel aus dem RSA-Kryptosystem sowie das zugehörige Zertifikat nach X.509 Standard bereitgestellt werden. Eine Beschreibung des Moduls befindet sich in Abschnitt 4.5.5.4
<b>Anmerkungen</b>	Mit der Signatur wird nicht das Dokument selbst unterschrieben, sondern dessen Content-Hash. Das hat den Vorteil, dass man dasselbe Dokument in verschiedenen Formaten verifizieren lassen kann. Zum Beispiel kann ein XSL.FO-Dokument verlinkt werden und die, daraus erstellte PDF, später verifiziert werden, da beide denselben Content-Hash haben.

*Tabelle 5.4: Ergebnis der Anforderung R9*

#### 5.1.2.5 Anforderung R10 - Verlinkung eines Dokuments in einer 'Distributed-Hash-Table' oder Blockchain

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Nach den Recherchen von Anforderung R2, wurde im 'Document-Linking'-Modul aus Abschnitt 4.5.5.5 ein Algorithmus zur Verlinkung eines Dokuments in der IPFS-Blockchain realisiert. Durch die Architektur des Moduls können mit wenig Implementierungsaufwand weitere Verlinkungs-Provider zur Verfügung gestellt werden (siehe Abschnitt 4.5.5.5).
<b>Anmerkungen</b>	Um ein Dokument in der IPFS-Blockchain zu Verlinken, muss der IPFS-Client ('ipfs daemon') mit Verbindung zum IPFS-Netzwerk gestartet sein. Außerdem kann es zu Verzögerungszeiten kommen um einen Link im IPFS-Netzwerk zu verteilen.

*Tabelle 5.5: Ergebnis der Anforderung R10*

#### 5.1.2.6 Anforderung R11 - Injektion eines Links in ein Dokument

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Mit dem 'Link-Injection'-Modul aus Abschnitt 4.5.5.6 wurden, je nach Dokumententyp, verschiedene Verfahren zur Injektion eines Links in ein Dokument implementiert.
<b>Anmerkungen</b>	Aufgrund der Instabilität der OCR-Ergebnisse (siehe Abschnitt 5.1.1.1) kann es zu schwerwiegenden Problemen kommen, wenn ein Link-Text per Regex- beziehungsweise OCR-Verfahren aus einem Dokument extrahiert werden soll. Per Default ist daher bei PDFs und Bilddokumenten das QR-Code Verfahren eingestellt.

*Tabelle 5.6: Ergebnis der Anforderung R11*

#### 5.1.2.7 Anforderung R12 - Extraktion eines Links aus dem Inhalt eines Dokuments

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Durch das 'Link-Extraction'-Modul aus Abschnitt 4.5.5.7 wurden, je nach Dokumententyp, verschiedene Verfahren zur Injektion eines Links in ein Dokument implementiert.
<b>Anmerkungen</b>	Aufgrund der Instabilität der OCR-Ergebnisse (siehe Abschnitt 5.1.1.1) kann es zu schwerwiegenden Problemen kommen, wenn ein Link-Text per Regex- beziehungsweise OCR-Verfahren aus einem Dokument extrahiert werden soll. Per Default ist daher bei PDFs und Bilddokumenten das QR-Code Verfahren eingestellt.

*Tabelle 5.7: Ergebnis der Anforderung R12*



#### 5.1.2.8 Anforderung R13 - Auflösen eines Links und Laden der Verlinkungsdaten eines Dokuments

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Durch das 'Linking-Data-Loader'-Modul aus Abschnitt 4.5.5.8 können die Verlinkungsdaten eines Dokuments mit einem Link aus dem IPFS-Netzwerk heruntergeladen werden. Durch die Architektur des Moduls können mit wenig Implementierungsaufwand weitere Verlinkungs-Provider zur Verfügung gestellt werden (siehe Abschnitt 4.5.5.5) .
<b>Anmerkungen</b>	Aufgrund der eventuellen Verzögerungszeit um einen Link im IPFS-Netzwerk zu verteilen kann es zu längeren Wartezeiten beim Auflösen des Links kommen.

*Tabelle 5.8: Ergebnis der Anforderung R13*

#### 5.1.2.9 Anforderung R14 - Überprüfung des Inhalts eines Dokuments

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Mit dem in Abschnitt 4.5.5.9 vorgestellten 'Content-Validation'-Modul kann der Inhalt eines Dokuments anhand dessen Verlinkungsdaten überprüft werden.
<b>Vorraussetzung</b>	Verlinkungsdaten müssen zur Verfügung stehen.

*Tabelle 5.9: Ergebnis der Anforderung R14*

#### 5.1.2.10 Anforderung R15 - Überprüfung der Signatur eines Dokuments

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Mit dem 'Signature-Verification'-Modul aus Abschnitt 4.5.5.10, kann die Signatur eines Dokuments (anhand dessen Verlinkungsdaten) überprüft werden. Außerdem wird die Identität des Unterzeichners mit Hilfe eines X.509 Zertifikats überprüft.
<b>Vorraussetzung</b>	Verlinkungsdaten müssen zur Verfügung stehen.

*Tabelle 5.10: Ergebnis der Anforderung R15*

#### 5.1.2.11 Anforderung R16 - Unterstützung von Dokumenten im PDF Format

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Da der interne Aufbau von PDF-Dateien komplex und schlecht handhabbar ist, wurden in den Modulen verschiedene Bibliotheken verwendet, um PDF Dateien zur Verarbeitung in Bild- oder Textdateien umzuwandeln. Ist der Vorgang beendet, wird die Bild- beziehungsweise Bilddatei wieder in eine PDF transformiert. Da es für die Konvertierung einer Bild- in eine PDF-Datei keine besonders guten Bibliotheken für C++ gibt, wird ein Python Paket verwendet, welches im C++ Code eingebettet ist. Somit werden auch PDF-Dateien vom 'Printed Document Linking System' unterstützt.

*Tabelle 5.11: Ergebnis der Anforderung R16*

#### 5.1.2.12 Anforderung R17 - Unterstützung von Dokumenten in gängigen Bildformaten

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Durch die Nutzung der Bibliothek OpenCV zu Bildverarbeitung, werden alle gängigen Bildformate ('TIF / TIFF', 'BMP', 'JPEG / JPG', 'PNG') vom 'Printed Document Linking System' unterstützt.

*Tabelle 5.12: Ergebnis der Anforderung R17*

#### 5.1.2.13 Anforderung R18 - Unterstützung von Dokumenten im XSL.FO Format

<b>Erfüllt</b>	Ja
<b>Lösungsansatz</b>	Mit Hilfe der pugiXML Bibliothek werden alle Arten von (validen) XML Formaten vom 'Printed Document Linking System' unterstützt.

*Tabelle 5.13: Ergebnis der Anforderung R18*

### 5.1.3 Anforderung des Typs 'Design'

Wie bereits in Abschnitt 4.5 erwähnt, waren die Haupteinflussfaktoren des Software-Designs der 'Proof-Of-Concept'-Implementierung die 'GRASP'-Prinzipien (siehe [47]) und somit auch die Modularität und Erweiterbarkeit der Architektur. Außerdem wurde an vielen Stellen in der Architektur des implementierten Systems, 'Gang of Four'-Entwurfsmuster aus dem Buch 'Design Patterns. Elements of Reusable Object-Oriented Software' [48] umgesetzt. Die am häufigsten angewandten Muster sind das 'Singleton'-, 'Facade'- und 'Strategy'-Pattern. 'Facade' ist ein sogenanntes 'Structural-Pattern', das heißt es ist zur Strukturierung eines Systems gut geeignet, was sich durch ein modulares Design ausdrückt. Ebenfalls der Modularität zugute kommt das 'Singleton' Muster, indem es garantiert, dass bestimmte Schlüsselklassen des Systems nur einmal instantiiert werden können. Mit dem 'Strategy'-Pattern aus der Kategorie 'Behavioural-Patterns', wird die Erweiterbarkeit eines Systems durch das Anbieten vieler Schnittstellen unterstützt. Wurde ein Algorithmus implementiert, welcher eine bestimmte Software-Bibliothek verwendet, wurde immer ein Interface bereitgestellt, das unabhängig von einer Bibliothek ist. Folgende Anforderungen gelten somit als erfüllt:

- Anforderung R19 - Modulare und übersichtliche Software
- Anforderung R20 - Erweiterbarkeit der Software
- Anforderung R21 - Einfache Möglichkeit zur Auswechslung der verwendeten Bibliotheken

Die Anforderung 'R22 - Implementierung des Proof-Of-Concept als Webdienst oder Handy-App' wurde nicht erfüllt, da sich im Verlauf der Arbeit für eine 'CLI'- beziehungsweise Kommandozeilen-Applikation entschieden wurde. Dies liegt vor allem daran, dass sich C++ Bibliotheken zur Entwicklung eines Kommandozeilen-Programms besser eignen und weil sich eine 'CLI-Application' besser testen lässt und Bugs einfacher behoben werden können.

Jedoch kann der Programmcode, da er in C++ geschrieben ist, auf alle gängigen Smartphonebetriebssysteme portiert werden.

Ein 'SOAP/WSDL'-basierter Webservice würde sich gut mit Hilfe des 'gSOAP' Frameworks (siehe [70]) erstellen lassen. Soll ein 'REST'-Service angeboten werden, könnte das 'C++ REST SDK' von Microsoft (siehe [71]) verwendet werden.

### 5.1.4 Übersicht der Anforderungsergebnisse

Wie in Tabelle 5.14 dargestellt, wurden alle Anforderungen bis auf R22 erfüllt. Der, in Abschnitt 4.1.2 ermittelte, maximal erreichbare Gesamtwert aller Anforderungen liegt bei 106. Der tatsächlich erreichte Wert liegt bei  $13 \cdot 7 + 4 \cdot 3 + 3 \cdot 1 = 104$ . Somit wurden  $104/106 \cong 0.9811 \Leftrightarrow 98.11\%$  der Anforderungen erfüllt beziehungsweise beantwortet.

Titel	ID	Priorität	Erfüllt?
Wie stabil ist die OCR Erkennung?	R1	Hoch	Ja
Wie kann ein Dokument in einer DHT oder Blockchain referenziert werden?	R2	Hoch	Ja
Können Kopien von Dokumenten über die gleiche ID referenziert werden?	R3	Mittel	Ja
Kann ein Dokument über seinen Content-Hash referenziert werden?	R4	Mittel	Ja
Wie kann eine Abbildung vom Content-Hash auf die Binär-Hashes eines Dokuments verwaltet werden?	R5	Optional	Nein
Extraktion des Inhaltes eines Dokuments	R6	Hoch	Ja
Normalisierung des Inhaltes eines Dokuments	R7	Hoch	Ja
Erzeugung des Content-Hash eines Dokuments	R8	Hoch	Ja
Erzeugen einer Digitalen Signatur eines Dokuments	R9	Optional	Ja
Verlinkung eines Dokuments in einer 'Distributed-Hash-Table' oder Blockchain	R10	Hoch	Ja
Injektion eines Links in ein Dokument	R11	Mittel	Ja
Extraktion eines Links aus dem Inhalt eines Dokuments	R12	Hoch	Ja
Auflösen eines Links und laden der Verlinkungsdaten eines Dokuments	R13	Hoch	Ja
Überprüfung des Inhalts eines Dokuments	R14	Hoch	Ja
Überprüfung der Signatur eines Dokuments	R15	Optional	Ja
Unterstützung von Dokumenten im PDF Format	R16	Hoch	Ja
Unterstützung von Dokumenten in gängigen Bild Formaten	R17	Optional	Ja
Unterstützung von Dokumenten im XSL.FO Format	R18	Mittel	Ja
Modulare und übersichtliche Software	R19	Hoch	Ja
Erweiterbarkeit der Software	R20	Hoch	Ja
Einfache Möglichkeit zur Auswechslung der verwendeten Bibliotheken	R21	Hoch	Ja
Implementierung des Proof-Of-Concept als Webdienst oder Handy-App	R22	Optional	Nein

*Tabelle 5.14: Zusammenfassung der Anforderungsergebnisse*

## 5.2 Beispielhafte Ausführungen des 'Printed Document Linking System'

In diesem Abschnitt werden Beispielszenarien der Anwendungsfälle aus Abschnitt 4.1.1 unter Anwendung der 'Proof-Of-Concept'-Implementierung vorgestellt. Zur Illustration dienen Screenshots, auf denen die Eingaben des Users und Ausgaben des Systems zusehen sind.

Dieses Vorgehen soll die Funktionalität der 'Proof-Of-Concept'-Implementierung unter Beweis stellen.

### Use-Case #1: Verlinkung eines gedruckten Notenspiegel Testdokument

In diesem Abschnitt wird der Anwendungsfall #1 aus Abschnitt 4.1.1.1 ausgeführt. Mit dem Aufruf aus Abbildung 5.3 wird das System gestartet. Durch die Option '-l' wird dem Programm gesagt, dass eine Verlinkung stattfinden soll. Mit '-d' wird das Eingabedokument, mit '-c' wird das Zertifikat, mit '-k' der private Schlüssel und mit '-o' das Ausgabedokument angegeben. Das Eingabedokument ist ein Scan eines Notenspiegel-Testdokuments.

Nach einer kurzen Zeit muss der User das Passwort des privaten Schlüssels eingeben. Diese Aufforderung des Systems ist in Abbildung 5.4 zu sehen.

Am Ende der Ausführung liefert das System eine wie im Screenshot aus Abbildung 5.5 den Pfad des verlinkten Dokuments.

Abbildung 5.6 zeigt einen Screenshot des Ausgabedokuments in einem PDF-Viewer. Der QR-Code in der rechten unteren Ecke beinhaltet den Link des Dokuments.

```
./PrintedDocumentLinkingSystem -l -d ../data/Arial_10_Strong_scan.pdf  
-k ../data/privateKey.pem -c ../data/cert.pem -o ../data/Arial_10_Stron  
g_scan_linked.pdf
```

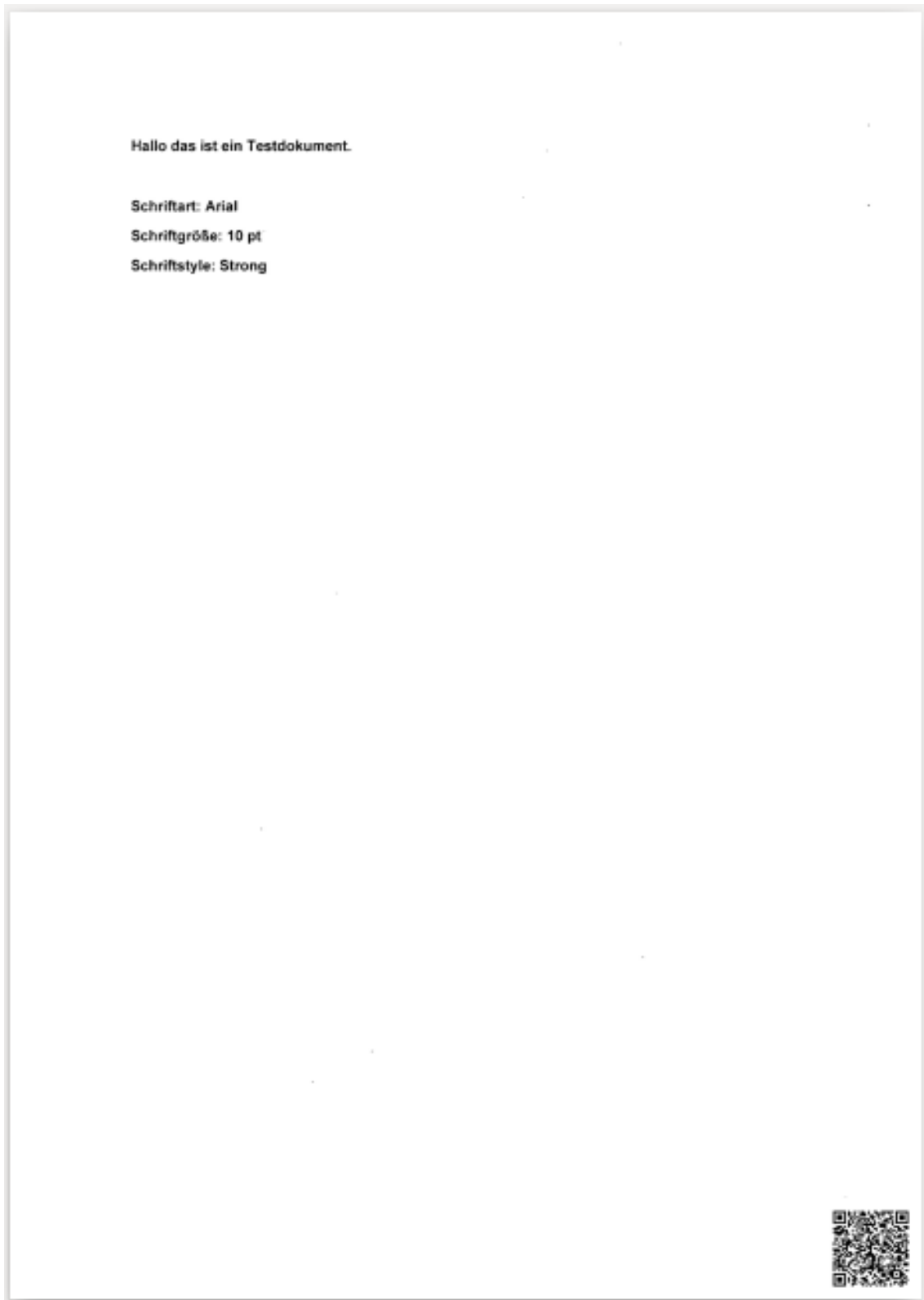
Abbildung 5.3: Beispiel Ausführung UseCase #1: Aufruf des 'PDLS' um ein Dokument zu Verlinken.

```
Enter PEM pass phrase:
```

Abbildung 5.4: Beispiel Ausführung UseCase #1: Aufforderung zur Eingabe des Passworts für den privaten Schlüssel

```
Successfully linked document and created verifiable document:  
"../data/Arial_10_Strong_scan_linked.pdf"
```

Abbildung 5.5: Beispiel Ausführung UseCase #1: Erfolgsmeldung bei Abschluss der Ausführung



*Abbildung 5.6: Beispiel Ausführung UseCase #1: Screenshot des Resultats*

## Use-Case #2: Verlinkung eines neu erstellten Notenspiegel Testdokument

In diesem Abschnitt wird der Anwendungsfall #2 aus Abschnitt 4.1.1.2 mit Hilfe der 'Proof-Of-Concept'-Implementierung ausgeführt.

Diese Ausführung unterscheidet sich bis auf das Eingabe- und Ausgabedokument nicht von der Ausführung des Anwendungsfalls UC#1 im vorherigen Abschnitt. Das Eingabedokument ist eine Beispieldatei im XSL.FO Format, wie sie in der Hochschule Ulm für Notenspiegel genutzt wird.

Abbildung 5.10 zeigt einen Screenshot des Ausgabedokuments im Terminal. Da es sich um eine XML-Datei handelt, wurde der Link nicht in Form eines QR-Codes, sondern als Text in das Dokument eingefügt.

```
./PrintedDocumentLinkingSystem -l -d ../data/notenspiegel.xsl.fo -c ../data/cert.pem -k ../data/privateKey.pem -o ../data/notenspiegel_linked.xsl.fo
```

Abbildung 5.7: Beispiel Ausführung UseCase #2: Aufruf des 'PDLS' um ein Dokument zu Verlinken.

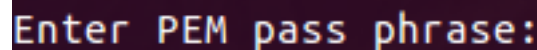


Abbildung 5.8: Beispiel Ausführung UseCase #2: Aufforderung zu Eingabe des Passworts für den privaten Schlüssel

```
Successfully linked document and created verifiable document: "../data/notenspiegel_linked.xsl.fo"
```

Abbildung 5.9: Beispiel Ausführung UseCase #2: Erfolgsmeldung bei Abschluss der Ausführung

```
--| head -20 ../data/notenspiegel_linked.xsl.fo
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:fox="http://xml.apache.org/fop/extensions">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="pagemaster1" page-height="845.0pt" page-width="598.0pt" margin-
    <fo:region-body margin-left="46.0pt" margin-top="28.0pt" margin-bottom="74.0pt" margin-right="46.0pt" background-position-vertical="top" />
    <fo:region-before extent="28.0pt" precedence="true" background-image="url()" background-repeat="no-repeat" />
    <fo:region-after extent="74.0pt" precedence="true" background-image="url()" background-repeat="no-repeat" />
    <fo:region-start extent="46.0pt" background-image="url()" background-repeat="no-repeat" />
    <fo:region-end extent="28.0pt" background-image="url()" background-repeat="no-repeat" />
  </fo:simple-page-master>
</fo:layout-master-set>
<fo:bookmark-tree />
<fo:page-sequence master-reference="pagemaster1" force-page-count="no-force">
  <fo:static-content flow-name="xsl-region-before">
    <fo:block />
    <fo:block />
    <fo:inline>###ipfs/QmbYTk2yUxpMwXsRPECbBkPCbQ8nv7mpnvUpNFCmiqNTqL###</fo:inline>
  </fo:static-content>
</fo:page-sequence>
<fo:static-content flow-name="xsl-region-after">
```

Abbildung 5.10: Beispiel Ausführung UseCase #2: Screenshot des Resultats

### Use-Case #3: Verifizierung eines gedruckten Notenspiegel Testdokument

In diesem Abschnitt wird der Anwendungsfall #3 aus Abschnitt 4.1.1.3 mit Hilfe der 'Proof-Of-Concept'-Implementierung ausgeführt.

In Abbildung 5.11 ist der Aufruf des Systems zu sehen. Mit der Option '-v' wird das System angewiesen, eine Verifikation des Eingabedokuments durchzuführen. Das Eingabedokument, das dem System mit der Option '-d' übergeben wird, ist eine gedruckte und dann wieder gescannte Version des Ausgabedokuments aus Abschnitt 5.2.

In Abbildung 5.12 ist das Resultat der Ausführung, also das Ergebnis der Verifikation zu sehen. 'CONTENT\_VALID\_CERT\_VALID\_SIG\_VALID' bedeutet, dass der Inhalt ('CONTENT'), das Zertifikat ('CERT') als auch die Signatur ('SIG') des Dokuments gültig ('VALID') sind.

```
./PrintedDocumentLinkingSystem -v -d ../data/Arial_10_Strong_scan_linked.pdf
```

Abbildung 5.11: Use-Case Diagramm des Proof-Of-Concept

```
Validationstate of Document '../data/Arial_10_Strong_scan_linked.pdf' is :  
CONTENT_VALID_CERT_VALID_SIG_VALID
```

Abbildung 5.12: Use-Case Diagramm des Proof-Of-Concept



## 5.3 Schwachstellen der Implementierung

### 5.3.1 OCR

Wie bereits in Abschnitt 5.1.1.1 dokumentiert, sind die Ergebnisse der OCR-Analysen sehr instabil. Dies stellt ein ernstes Problem dar, weil die Verlinkung eines Dokuments als auch die Verifizierung eines Dokuments auf dem Content-Hash des Dokuments basiert und der Content-Hash aus dem OCR-Ergebnis des Dokuments berechnet wird. Wird auch nur ein einziges Zeichen bei der OCR-Erkennung falsch erkannt, ändert sich der gesamte Content-Hash und ein Dokument, welches eigentlich gültig wäre, würde als ungültig gekennzeichnet werden.

Bei zukünftigen Arbeiten im selben Themenbereich dieser Arbeit, sollte oberste Priorität sein, die OCR-Ergebnisse zu verbessern. Ideen, wie dies bewerkstelligt werden könnte, sind in Abschnitt 6.2.1 zu finden.

### 5.3.2 PDF-Dokumente

Da PDF-Dateien aufgrund ihrer schlechten Handhabbarkeit im 'Printed Document Linking System' konvertiert werden müssen, um mit ihnen arbeiten zu können, kann es (gerade bei großen Dateien) zu längeren Wartezeiten bei der Konvertierung kommen. Vorschläge, wie das System auch mit PDFs direkt, also ohne Konvertierung arbeiten kann, sowie allgemeine Verbesserungsvorschläge für die Performance befinden sich in Abschnitt 6.2.2.

Ein weiteres Problem, welches bei der Verarbeitung von PDF-Dokumenten in der 'Proof-Of-Concept'-Implementierung auftritt, ist die Limitierung der PDFs auf eine Seite. Dies hängt vor allem mit der Konvertierung der PDFs zusammen, da immer nur die erste Seite umgewandelt wird. Das Problem kann einfach behoben werden, indem der Programmcode der Konverter erweitert wird, wurde aber in dieser Arbeit ignoriert, da es sich nur um eine 'Proof-Of-Concept'-Implementierung handelt.

## 5.4 Sicherheitskritische Schwachstellen des Systems

Da das System viel auf anderen Systemen basiert, erbt es alle Schwachstellen dieser Systeme. Diese Schwachstellen sind nicht beeinflussbar, aber auch nicht besonders kritisch, da es sich um gängige und lang erprobte Systeme handelt.

Zum Beispiel basiert die 'Proof-Of-Concept'-Implementierung auf der Blockchain beziehungsweise dem Netzwerk von IPFS und auf dem OpenSSL-System. Diese Systeme gelten als sicher und es ist sehr unwahrscheinlich, dass es in absehbarer Zeit gelingen sollte, eines der Systeme lahm zulegen.

'Man in the middle' Attacken sind ebenfalls extrem unwahrscheinlich, da die Verbindung zum IPFS-Netzwerk über eine HTTPS Verbindung aufgebaut wird.

Was jedoch ein Problem darstellen könnte, ist die Verlinkung von gefälschten Dokumenten. Nehmen wir an, dass ein/e Student\*in mit seinem Zeugnis unzufrieden ist und Zeugnis so fälscht, das die Note 2.0 mit der Note 1.0 ersetzt wird. Hat der/die Student\*in aus irgendeinem Grund Zugriff auf das 'Printed Document Linking System', kann er das gefälschte Dokument verlinken und Kopien davon weiter geben. Will nun zum Beispiel ein Arbeitgeber das Zeugnis mit Hilfe des Systems überprüfen, wird es als gültig angesehen. Dieses Problem lässt sich nur verhindern, indem darauf geachtet wird, welche Personen Zugriff auf das System bekommen und welche nicht.

## Kapitel 6

### Zusammenfassung und Ausblick

## 6.1 Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde ein Konzept entwickelt, wie gedruckte beziehungsweise gescannte Dokumente sowie rein digitale Dokumente im PDF-, Bild-, XML- oder reinen Textformat in einer Blockchain verlinkt werden können. Durch diese Verlinkung ist es möglich, Kopien der verlinkten Dokumente zu verifizieren. Um die Funktionalität des Konzepts unter Beweise zu stellen, wurde eine umfangreiche 'Proof-Of-Concept'-Implementierung entwickelt.

Bei der Verlinkung eines Dokuments wird dessen Inhalt extrahiert und daraus der sogenannte Content-Hash des Dokuments berechnet. Die Extraktion des Inhalts basiert bei Dokumenten im PDF- und Bildformat auf einer OCR-Analyse; bei Text- oder XML-Dokumenten fällt dieses Vorgehen weg. Der erzeugte Content-Hash wird mit dem privaten Schlüssel des Unterzeichners signiert. Die daraus entstandene digitale Signatur wird zusammen mit dem Zertifikat des Unterzeichners und dem Content-Hash des Dokuments gebündelt und in der IPFS-Blockchain verlinkt. Der Link zu diesen Daten wird anschließend in eine Kopie des Dokuments eingefügt. Handelt es sich beim Dokument um ein PDF- oder Bilddokument, wird der Link in Form eines QR-Codes eingebettet; bei Text- oder XML-Dokumenten wird ein Link in Textform eingebunden.

Bei der Verifizierung (einer Kopie) eines verlinkten Dokuments wird der Link aus dem Dokument extrahiert. Mit dem Link werden nun die Daten aus dem IPFS-Netzwerk heruntergeladen. Außerdem wird erneut der Content-Hash des Dokuments (wie bei der Verlinkung) berechnet. Um den Verifikationsstatus zu ermitteln, wird der neu berechnete Content-Hash und der heruntergeladene Content-Hash verglichen, das Zertifikat kontrolliert und die Signatur überprüft. Sind alle drei Zwischenergebnisse positiv, so gilt das Dokument als verifiziert.

Des Weiteren wurde die Stabilität des OCR-Algorithmus, welcher verwendet wird, um den textuellen Inhalt eines PDF- oder Bilddokuments zu extrahieren, getestet und evaluiert. Dieser Stabilitätstest ist in vier Testphasen unterteilt. In der ersten Testphase wurde getestet, welche Schrifteigenschaften zu guten OCR-Ergebnissen führen. Im Rahmen der zweiten und dritten Testphase wurde getestet, ob sich Tabellen oder Tabulatoren besser zur OCR-Analyse eignen. In der letzten Testphase wurde überprüft, wie stark sich die Ergebnisse von gedruckten Dokumenten (Scans) im Vergleich zu rein digitalen Dokumenten unterscheiden. Da der Fokus der Arbeit nicht auf der OCR-Erkennung liegt, wurden alle Tests mit der Standardkonfiguration des OCR-Algorithmus durchgeführt. Dies hatte auch den Zweck, zu testen, ob diese Einstellungen bereits genügen, oder ob mehr Arbeitsaufwand in die Konfiguration des OCR-Algorithmus investiert werden muss. Die Testergebnisse sind nicht zufriedenstellend und es stellte sich heraus, dass die OCR-Analyse mit erheblich mehr Aufwand betrieben und getestet werden muss.

## 6.2 Ausblick

### 6.2.1 Vorschläge zur Verbesserung der OCR-Ergebnisse

In diesem Abschnitt werden Ideen und Vorschläge zur Verbesserung der OCR-Ergebnisse beschrieben. Da die OCR-Erkennung ein sehr umfangreiches Thema ist, und die Optimierung der Ergebnisse mit viel Arbeitsaufwand zusammenhängt, wird vorgeschlagen, eine Bachelorarbeit in Auftrag zu geben, welche sich ausschließlich mit der OCR-Erkennung auseinandersetzt.

#### 6.2.1.1 Vorschläge zur Optimierung von TesseractOCR

Die Fehler aus den OCR-Testphasen (siehe Abschnitt 5.1.1.1) lassen sich eventuell verhindern, indem man Tesseract mit einem benutzerdefinierten Wörterbuch initialisiert, welches die zu erwartenden Wörter enthält. Dadurch kann Tesseract die gefundenen Wörter mit den Wörtern des Wörterbuchs vergleichen und gegebenenfalls den Fehler ausbessern.

Eine weitere Möglichkeit wäre, Tesseract auf bestimmte Schriftarten zu trainieren. Je mehr Schriftarten Tesseract kennt, desto besser kann der Algorithmus Buchstaben und damit auch Wörter erkennen. **TesseractTraining**

Die Einstellungsmöglichkeiten zu den verschiedenen Segmentierungsmethoden des OCR-Algorithmus sowie die verschiedenen 'OCR-Engines' von Tesseract sollten ebenfalls getestet werden. Im Allgemeinen sollte der ganze Abschnitt zur Verbesserung des Ergebnisses von Tesseract in dessen Dokumentation (siehe [72]) durchgearbeitet und evaluiert werden.

Was auch in Betracht gezogen werden sollte, ist ein Upgrade der verwendeten Tesseract Version. Die in dieser Arbeit verwendete Version ist Tesseract 3.05. Dieser Version ist die offizielle 'latest stable release'-Version. Seit Kurzem ist jedoch eine Alpha-Version Tesseract 4.0 [73] verfügbar, welche auf einem neuronalen Netzwerk basiert. Laut Dokumentation sollen die Ergebnisse signifikant besser sein. Der Trade-Off dabei ist die benötigte Rechenleistung.

Um noch bessere Ergebnisse mit der Tesseract API zu erlangen, ist es ein Muss, sich die Möglichkeiten der 'Advanced API' (siehe [74]) anzuschauen und von ihnen Gebrauch zu machen.

#### 6.2.1.2 'Image-Preprocessing'

Gerade um die richtige Reihenfolge der erkannten Wörter bei der OCR-Analyse zu garantieren, sollte man eine Vorverarbeitung der Bilder implementieren.

Zum Beispiel könnte man nach dem 'Divide and Conquer'-Prinzip vorgehen und einzelne Textblöcke aus dem Bild extrahieren und diese Textblöcke dann einzeln vom OCR-Algorithmus analysieren lassen. In welcher Reihenfolge die Textblöcke analysiert werden, könnte anhand deren Position festgelegt werden.

Eine Vorverarbeitung die die Qualität der Bilder verbessert ist wahrscheinlich nicht nötig, da es sich um Scans handelt, die schon von sich aus eine sehr gute Qualität haben. Dennoch könnte man eine Binarisierung und Rauschunterdrückung oder Skalierung der Bilder in Erwägung ziehen. Die OpenCV-Bibliothek bietet hierfür Funktionalitäten.

### 6.2.1.3 Nutzung von professionellen OCR-Bibliotheken

Sollte es nicht gelingen, befriedigende Ergebnisse mit der Tesseract Bibliothek zu erlangen, sollte in Betracht gezogen werden, professionelle OCR-SDKs beziehungsweise OCR-Bibliotheken zu nutzen. Diese professionellen Bibliotheken haben den Vorteil, dass sie schon ausreichend getestet und optimiert wurden, sodass eine sehr hohe allgemeine OCR-Erkennungsrate garantiert wird. Die meisten großen Unternehmen wie zum Beispiel Microsoft, Samsung oder Kodak nutzen diese Tools [75] [76], was die Stärke und Fortschrittlichkeit dieser professionellen Bibliotheken beziehungsweise SDKs weiter untermauert.

Die folgende Auflistung enthält eine Auswahl solcher SDKs, deren Nachteil die kostenpflichtige Nutzung ist. Um die genauen Eigenschaften und Spezifikationen eines SDKs beziehungsweise einer Bibliothek zu erhalten, sind die entsprechenden Literaturhinweise zu beachten.

- Adobe Acrobat [77]
- ABBYY Finereader [78]
- Leadtools OCR [79]
- Nuance Omnipage [80]
- Microsoft Cognitive Services [81]

### 6.2.1.4 Sonstige Vorschläge zur Verbesserung der OCR-Ergebnisse

Als weitere Lösungsmöglichkeit um die OCR-Ergebnisse zu verbessern, könnte man die Levenstein-Distanz [82] und ein benutzerdefiniertes Wörterbuch nutzen, um fehlerhafte Wörter zu korrigieren.

Außerdem könnte man versuchen, eine eigene OCR-Engine zu implementieren. Bei einer Recherche wurde ein sehr interessantes Projekt beziehungsweise Tutorial (siehe [83]) gefunden, wie man dies bewerkstelligen kann. In diesem Tutorial werden erst die Grundlagen erläutert und danach Schritt für Schritt die Implementierung erläutert. Der Quellcode des Projekts ist in C# und kann umsonst genutzt und heruntergeladen werden.

## 6.2.2 Allgemeine Verbesserungs- und Erweiterungsvorschläge

In diesem Abschnitt werden allgemeine Verbesserungs- und Erweiterungsvorschläge der 'Proof-Of-Concept'-Implementierung aus Abschnitt 4.5 beschrieben, welche aus zeitlichen oder thematischen Gründen nicht umgesetzt wurden.

### 6.2.2.1 Testen der Ergebnisse der Scanner-internen OCR-Algorithmen

Die entwickelte 'Proof-Of-Concept'-Implementierung bietet die Funktion den Inhalt einer durchsuchbaren PDF (siehe Abschnitt 4.5.5.1) zu extrahieren, indem die PDF in ein Textdokument konvertiert wird. Den Inhalt einer Textdatei auszulesen ist triviale Aufgabe, und es können keine Fehler beim Extrahieren des Inhaltes passieren. Implementiert wurde diese Funktionalität im 'Content Extraction Module' aus Abschnitt 4.5.5.1. Da die zur Konvertierung verwendete Bibliothek 'GhostScript' keine Funktionalität bietet um festzustellen, ob eine PDF durchsuchbar ist oder nicht, muss der/die Anwender\*in diese Information beim Programmstart liefern (siehe Abbildung 4.20). Da rein digitale PDFs, welche durch ein Textverarbeitungsprogramm

erstellt wurden, (fast) immer durchsuchbar sind, eignen sie sich hervorragend für diese Funktionalität.

Um einen Scan eines Dokuments als durchsuchbare PDF zu speichern, kann die Scanner-Software 'Kodak Capture Pro' [84] verwendet werden. Diese Software wurde zusammen mit dem während der Arbeit genutzten Scanner 'Kodak ScanMate i1150WN' geliefert. Um damit eine durchsuchbare PDF zu erzeugen, wird eine OCR-Analyse auf dem Scan durchgeführt.

Da dies eine offizielle Software ist, wird angenommen, dass die internen Algorithmen zur OCR-Erkennung bereits ausreichend getestet und optimiert wurden.

In einer neuen OCR-Testphase sollte geprüft werden, wie stabil die OCR-Ergebnisse der Scanner-Software tatsächlich sind und ob die Funktionalität sich für das Vorhaben dieser Arbeit eignet.

#### **6.2.2.2 Direkte Verarbeitung von PDF-Dokumenten**

Die Bearbeitung der PDF-Dokumente funktioniert in der aktuellen Implementierung nur, indem die PDF- in Bilddateien konvertiert werden. Dieses Vorgehen ist nicht sehr performant, könnte aber eventuell verhindert werden, in dem man die in dieser Arbeit verwendete Bibliothek GhostScript auswechselt. Folgende kostenlose Open-Source Bibliotheken zur Verarbeitung von PDFs würden sich eventuell für das direkte Arbeiten (ohne Konvertierung) mit PDFs eignen. Dies wurde allerdings noch nicht getestet und genau untersucht.

- libHaru [85]
- PoDoFo [86]
- JagPDF [87]

Falls diese Bibliotheken nicht allen Anforderungen gerecht werden, könnte eventuell das kostenpflichtige Adobe Acrobat SDK [77] Hilfe bieten. Die Wahrscheinlichkeit, dass dieses SDK den Anforderungen entspricht, ist sehr hoch, da das PDF Format von Adobe entwickelt wurde.

#### **6.2.2.3 Verbesserung der Performance**

Da das 'Printed Document Linking System' aus mehreren Modulen besteht, die man auch als Verarbeitungsstufen betrachten kann, bietet es sich an, das System um die Funktionalität einer 'Multithreaded-Pipeline' zu erweitern. Jedes Modul würde dann eine Pipeline-Stufe darstellen. Dies kann, dank des C++11 'Concurrency'-Features [88] ohne großen Implementierungsaufwand bewerkstelligt werden. Zum Beispiel könnte man die Module zu Funktoren erweitern, indem der 'operator()' überladen wird. In der Hauptklasse, welches die Module verwaltet, könnte ein Thread (beziehungsweise 'std::async') pro Modul gestartet werden. Der Vorteil einer Realisation als Pipeline wäre die performante Stapelverarbeitung, wenn viele Dokumente verlinkt werden sollen. Soll nur ein einziges Dokument verlinkt werden, bringt die Pipeline nichts.

#### **6.2.2.4 Funktionale Erweiterungsmöglichkeiten**

Eine weitere Möglichkeit zur Erweiterung des Systems wäre, wenn bei Nichtübereinstimmung des Inhalts eines verlinkten Dokuments, ein visueller Abgleich mit dem Inhalt des Originals durch einen Menschen ermöglicht wird. Bei diesem Abgleich

könnten, ähnlich wie bei einem 'diff-Tool', Nichtübereinstimmungen hervorgehoben werden. Somit könnten mögliche Fehler der OCR-Erkennung des verlinkten Dokuments ignoriert werden. Um diese Funktionalität zu realisieren, müsste man den Originalinhalt zu den Verlinkungsdaten (siehe Abschnitt 4.5.2) eines Dokuments hinzufügen. Schlägt der Vergleich der Content-Hashes fehlt, könnte man mit der 'diff-match-patch'-Bibliothek [89] von Google den Vergleich der Inhalte machen und die Bereiche der Nichtübereinstimmung mit der OpenCV-Bibliothek hervorheben und anzeigen.

Da es im System viele Einstellungsmöglichkeiten gibt, wäre es sinnvoll, eine Möglichkeit zu bieten, die gewünschten Einstellungen in einer Config-Datei zu speichern und die Datei beim Aufruf dem System mitzugeben. Für diese Config-Datei würde sich das JSON-Format anbieten, da das System ohnehin schon mit JSON arbeitet.

#### **6.2.2.5 Anbindung an die Master-Thesis von Emmanuel Schwartz**

Während dieser Arbeit hat Emmanuel Schwartz parallel an einer Master-Thesis mit verwandtem Thema gearbeitet. Das Thema der Arbeit dreht sich um eine dezentrale Dokumentenverwaltung auf Basis der Ethereum Plattform und trägt den Namen "Decentralized Document Management". Ziel dieser Arbeit ist es Links von Dokumenten, die bei verschiedenen Anbietern hochgeladen wurden, in der Ethereum Blockchain zu speichern. Jeder Nutzer erhält ein Dokumenten-'Wallet', das alle Dokumente des Nutzers auf Basis der Links aus der Blockchain auflistet und abrufen kann.

Um diese beiden Arbeiten zu verbinden, könnte man das 'Printed Document Linking System' aus Abschnitt 4.5 so erweitern, dass die verlinkten Ausgabedokumente direkt vom System in das IPFS-Netzwerk hochgeladen und deren Links in der Ethereum Blockchain verlinkt werden. Dafür sollte ein neues Modul in das System eingeführt werden, das diese Aufgabe übernimmt. Außerdem muss das Provider-Interface für Ethereum realisiert werden (siehe Abschnitt 4.5.2). Da für die Anbindung noch zusätzliche Funktionen von Ethereum benötigt werden, sollte der dafür vorgesehene offizielle Client für C++ verwendet werden (siehe [90]).

Um den Link des verlinkten Dokuments in der Ethereum Blockchain zu speichern, damit das Dokument im Dokumenten-'Wallet' erscheint, muss der dafür entwickelte Smart-Contract [37] ausgeführt werden. Dieser Smart-Contract wird im Rahmen der Masterarbeit entwickelt und wird bei Bedarf zusammen mit den nötigen Informationen zu dessen Anwendung zur Verfügung gestellt. Da der Abgabetermin der Masterarbeit nach dem Abgabetermin meiner Bachelorarbeit ist, kann die Masterarbeit noch nicht referenziert werden. Ansprechpartner für nähere Informationen zu dieser Arbeit ist Prof. Dr. Traub.

# Abbildungsverzeichnis

3.1	Vereinfachte schematische Darstellung des Blockchain Prinzips . . . . .	10
4.1	Use-Case Diagramm des Proof-Of-Concept . . . . .	12
4.2	Säulendiagramm der Anzahl und Gesamtwerte der Anforderungen . .	18
4.3	Säulendiagramm der OCR-Erkennungsrate der Scans aus OCR-Testphase 2 mit 300 DPI . . . . .	31
4.4	Säulendiagramm der OCR-Erkennungsrate der Scans aus OCR-Testphase 2 mit 600 DPI . . . . .	31
4.5	Säulendiagramm der OCR-Erkennungsrate der Scans aus OCR-Testphase 3 mit 300 DPI . . . . .	37
4.6	Säulendiagramm der OCR-Erkennungsrate der Scans aus OCR-Testphase 3 mit 600 DPI . . . . .	37
4.7	Ausschnitt des Testdokuments 'Calibri_12_N_600_DPI' aus OCR-Testphase 3 . . . . .	38
4.8	OCR-Ergebnis der Tesseract CLI des Dokumentenausschnitts aus Abbildung 4.7 . . . . .	38
4.9	Säulendiagramm der OCR-Erkennungsrate der Notenspiegel-Testdokumente mit Tabulator aus OCR-Testphase 4 . . . . .	40
4.10	Säulendiagramm der OCR-Erkennungsrate der Notenspiegel-Testdokumente mit Tabelle aus OCR-Testphase 4 . . . . .	40
4.11	Erstellen eines digitalen Dokuments mit Hilfe eines Textverarbeitungsprogramms . . . . .	44
4.12	Erzeugung eines digitalen Dokuments durch Scannen eines gedruckten Dokuments . . . . .	44
4.13	Erzeugung eines gedruckten Dokuments durch Drucken eines digitalen Dokuments . . . . .	44
4.14	Schematische Darstellung der Verlinkung von Dokumenten . . . . .	45
4.15	'High-Level' Aktivitätsdiagramm der Verlinkung von Dokumenten . .	46
4.16	Schematische Darstellung der Dokumenten Verifikation . . . . .	46
4.17	Highlevel Aktivitätsdiagramm der Dokumenten Verifikation . . . . .	47
4.18	Wichtigste Datentypen des 'Printed Document Linking System' und deren Zusammenhänge . . . . .	52
4.19	Module des 'Printed Document Linking System' mit 'Facade-' und 'Singleton-Pattern' Architektur . . . . .	55
4.20	Hilfeanzeige des 'Printed Document Linking System' . . . . .	56
4.21	Aktivitätsdiagramm der 'High-Level'-Algorithmen in der Hauptklasse des 'Printed Document Linking System' . . . . .	57
4.22	Klassendiagramm des 'Content-Extraction-Module' . . . . .	59
4.23	Ebenen-Aufbau einer PDF-Datei (Quelle: [64]) . . . . .	60
4.24	Aktivitätsdiagramm des des 'Content-Extraction-Module' . . . . .	61
4.25	Klassendiagramm des 'Content-Normalization-Module' . . . . .	62



4.26	Aktivitätsdiagramm des 'Content-Normalization-Module'	62
4.27	Klassendiagramm des 'Content-Hash-Generation-Module'	63
4.28	Aktivitätsdiagramm des 'Content-Hash-Generation-Module'	63
4.29	Klassendiagramm des 'Signature-Generation-Module'	64
4.30	Aktivitätsdiagramm des 'Signature-Generation-Module'	65
4.31	Klassendiagramm des 'Document-Linking-Module'	66
4.32	Aktivitätsdiagramm des 'Document-Linking-Module'	67
4.33	Klassendiagramm des 'Link-Injection-Module'	68
4.34	Aktivitätsdiagramm des 'Link-Injection-Module'	69
4.35	Klassendiagramm des 'Link-Extraction-Module'	71
4.36	Aktivitätsdiagramm des 'Link-Extraction-Module'	72
4.37	Klassendiagramm des 'Linking-Data-Loading-Module'	73
4.38	Aktivitätsdiagramm des 'Linking-Data-Loading-Module'	73
4.39	Klassendiagramm des 'Content-Validation-Module'	74
4.40	Klassendiagramm des 'Signature-Verification-Module'	75
4.41	Aktivitätsdiagramm des 'Signature-Verification-Module'	75
5.1	Säulendiagramm der durchschnittlichen OCR-Erkennungsraten aus den verschiedenen OCR-Testphasen	77
5.2	Säulendiagramm der durchschnittlichen OCR-Erkennungsraten aus den OCR-Testphasen 2 bis 4 bezogen auf die Variationsparameter der Testdokumente	78
5.3	Beispiel Ausführung UseCase #1: Aufruf des 'PDLS' um ein Dokument zu Verlinken.	87
5.4	Beispiel Ausführung UseCase #1: Aufforderung zur Eingabe des Passworts für den privaten Schlüssel.	87
5.5	Beispiel Ausführung UseCase #1: Erfolgsmeldung bei Abschluss der Ausführung	87
5.6	Beispiel Ausführung UseCase #1: Screenshot des Resultats	88
5.7	Beispiel Ausführung UseCase #2: Aufruf des 'PDLS' um ein Dokument zu Verlinken.	89
5.8	Beispiel Ausführung UseCase #2: Aufforderung zu Eingabe des Passworts für den privaten Schlüssel.	89
5.9	Beispiel Ausführung UseCase #2: Erfolgsmeldung bei Abschluss der Ausführung	89
5.10	Beispiel Ausführung UseCase #2: Screenshot des Resultats	89
5.11	Use-Case Diagramm des Proof-Of-Concept	90
5.12	Use-Case Diagramm des Proof-Of-Concept	90

# Tabellenverzeichnis

4.1	UseCase: Verlinkung eines gedruckten Testdokument . . . . .	13
4.2	UseCase: Verlinkung eines neu erstellten Testdokument . . . . .	14
4.3	UseCase: Verifizierung eines gedruckten Testdokument . . . . .	15
4.4	Bedeutung der Anforderungstypen . . . . .	16
4.5	Bedeutungen der Anforderungsprioritäten . . . . .	16
4.6	Tabelle der Anforderungen . . . . .	17
4.7	Übersicht über Verlinkungsanbieter Dat Data . . . . .	19
4.8	Übersicht über Verlinkungsanbieter Ethereum . . . . .	20
4.9	Übersicht über Verlinkungsanbieter IPFS . . . . .	20
4.10	Übersicht über Verlinkungsanbieter Storj . . . . .	20
4.11	Variationsparameter der einfachen Testdokumente aus OCR- Testphase 1 . . . . .	23
4.12	Fehler der OCR Erkennung aus OCR Testphase 1 . . . . .	24
4.13	OCR-Ergebnistabelle der Scans aus OCR-Testphase 1 . . . . .	25
4.14	Variationsparameter der Notenspiegel-Testdokumente aus OCR- Testphase 2 und 3 . . . . .	26
4.15	OCR-Ergebnistabelle der Scans mit 300 DPI aus OCR-Testphase 2 .	29
4.16	OCR-Ergebnistabelle der Scans mit 600 DPI aus OCR-Testphase 2 .	30
4.17	OCR-Ergebnistabelle der Scans mit 300 DPI aus OCR-Testphase 3 .	35
4.18	OCR-Ergebnistabelle der Scans mit 600 DPI aus OCR-Testphase 3 .	36
4.19	OCR-Ergebnistabelle der Notenspiegel-Testdokumente mit Tabulator aus OCR-Testphase 4 . . . . .	41
4.20	OCR-Ergebnistabelle der Notenspiegel-Testdokumente mit Tabelle aus OCR-Testphase 4 . . . . .	42
4.21	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Tesseract . . . . .	48
4.22	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Boost . . . . .	48
4.23	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: OpenCV . . . . .	49
4.24	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Ghostscript Interpreter API . . . . .	49
4.25	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: OpenSSL . . . . .	49
4.26	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: PugiXML . . . . .	49
4.27	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: IPFS C++ API . . . . .	50
4.28	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Nlohmann JSON . . . . .	50

4.29	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: img2pdf . . . . .	50
4.30	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: Nayuki QR Code generation library . . . . .	50
4.31	Verwendete Bibliotheken der 'Proof-Of-Concept'-Implementierung: ZXing . . . . .	51
4.32	Namen und Beschreibungen der wichtigsten Datentypen im 'PDLS' .	53
4.33	Namen und Aufgaben der Module des 'Printed Document Linking System' . . . . .	54
5.1	Ergebnis der Anforderung R7 . . . . .	80
5.2	Ergebnis der Anforderung R7 . . . . .	80
5.3	Ergebnis der Anforderung R8 . . . . .	81
5.4	Ergebnis der Anforderung R9 . . . . .	81
5.5	Ergebnis der Anforderung R10 . . . . .	82
5.6	Ergebnis der Anforderung R11 . . . . .	82
5.7	Ergebnis der Anforderung R12 . . . . .	82
5.8	Ergebnis der Anforderung R13 . . . . .	83
5.9	Ergebnis der Anforderung R14 . . . . .	83
5.10	Ergebnis der Anforderung R15 . . . . .	83
5.11	Ergebnis der Anforderung R16 . . . . .	84
5.12	Ergebnis der Anforderung R17 . . . . .	84
5.13	Ergebnis der Anforderung R18 . . . . .	84
5.14	Zusammenfassung der Anforderungsergebnisse . . . . .	86

# Literaturverzeichnis

- [1] P. D. Bharati und P. N. Nitin, „Text watermarking algorithm using structural approach“, in *2012 World Congress on Information and Communication Technologies*, Okt. 2012, S. 629–633. DOI: 10.1109/WICT.2012.6409152.
- [2] Z. Jalil, A. M. Mirza und T. Iqbal, „A zero-watermarking algorithm for text documents based on structural components“, in *2010 International Conference on Information and Emerging Technologies*, Juni 2010, S. 1–5. DOI: 10.1109/ICIET.2010.5625705.
- [3] W. Authors. (2017). Steganography, Adresse: <https://en.wikipedia.org/wiki/Steganography> (besucht am 13.01.2017).
- [4] F. Daraee und S. Mozaffari, „Watermarking in binary document images using fractal codes“, *Pattern Recognition Letters*, Bd. 35, S. 120–129, 2014, Frontiers in Handwriting Processing, ISSN: 0167-8655. DOI: <http://dx.doi.org/10.1016/j.patrec.2013.04.022>. Adresse: <http://www.sciencedirect.com/science/article/pii/S0167865513001803>.
- [5] J. T. Brassil, S. Low, N. F. Maxemchuk und L. O’Gorman, „Electronic marking and identification techniques to discourage document copying“, *IEEE Journal on Selected Areas in Communications*, Bd. 13, Nr. 8, S. 1495–1504, Okt. 1995, ISSN: 0733-8716. DOI: 10.1109/49.464718.
- [6] M. J. Atallah, C. J. McDonough, V. Raskin und S. Nirenburg, „Natural language processing for information assurance and security: An overview and implementations“, in *Proceedings of the 2000 Workshop on New Security Paradigms*, Ser. NSPW ’00, Ballycotton, County Cork, Ireland: ACM, 2000, S. 51–65, ISBN: 1-58113-260-3. DOI: 10.1145/366173.366190. Adresse: <http://doi.acm.org/10.1145/366173.366190>.
- [7] Y. Liu, X. Sun und Y. Wu, „A natural language watermarking based on chinese syntax“, in *Advances in Natural Computation: First International Conference, ICNC 2005, Changsha, China, August 27-29, 2005, Proceedings, Part III*, L. Wang, K. Chen und Y. S. Ong, Hrsg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 958–961, ISBN: 978-3-540-31863-7. DOI: 10.1007/11539902\_119. Adresse: [http://dx.doi.org/10.1007/11539902\\_119](http://dx.doi.org/10.1007/11539902_119).
- [8] H. Wang, X. Sun, Y. Liu und Y. Liu, „Natural language watermarking using chinese syntactic transformations“, *Information Technology Journal*, Bd. 7, Nr. 6, S. 904–910, 2008.
- [9] H. M. Meral, B. Sankur, A. S. Özsoy, T. Güngör und E. Sevinç, „Natural language watermarking via morphosyntactic alterations“, *Computer Speech & Language*, Bd. 23, Nr. 1, S. 107–125, 2009.

- [10] M. J. Atallah, V. Raskin, C. F. Hempelmann, M. Karahan, R. Sion, U. Topkara und K. E. Triezenberg, „Natural language watermarking and tamperproofing“, in *International Workshop on Information Hiding*, Springer, 2002, S. 196–212.
- [11] O. Vybornova und B. Macq, „Natural language watermarking and robust hashing based on presuppositional analysis“, in *2007 IEEE International Conference on Information Reuse and Integration*, IEEE, 2007, S. 177–182.
- [12] Z. Yu und X. Liu, „A new digital watermarking scheme based on text“, in *2009 International Conference on Multimedia Information Networking and Security*, IEEE, Bd. 2, 2009, S. 138–140.
- [13] M.-Y. Kim, „Natural language watermarking by morpheme segmentation“, in *Intelligent Information and Database Systems, 2009. ACIIDS 2009. First Asian Conference on*, IEEE, 2009, S. 144–149.
- [14] P. Houser und J. Adler, *Electronic document verification system and method*, US Patent 5,606,609, Feb. 1997. Adresse: <https://www.google.com/patents/US5606609>.
- [15] A. Espejel-Trujillo, I. Castillo-Camacho, M. Nakano-Miyatake und H. Perez-Meana, „Identity document authentication based on vss and qr codes“, *Procedia Technology*, Bd. 3, S. 241–250, 2012.
- [16] M. H. Eldefrawy, K. Alghathbar und M. K. Khan, „Hardcopy document authentication based on public key encryption and 2d barcodes“, in *Biometrics and Security Technologies (ISBAST), 2012 International Symposium on*, IEEE, 2012, S. 77–81.
- [17] C. Y. Teoh, „Two-dimensional barcodes for hardcopy document integrity verification“, Diss., Universiti Teknologi Malaysia, Faculty of Computer Science und Information System, 2008.
- [18] A. Husain, M. Bakhtiari und A. Zainal, „Printed document integrity verification using barcode“, *Jurnal Teknologi*, Bd. 70, Nr. 1, 2014.
- [19] M. Warasart und P. Kuacharoen, „Paper-based document authentication using digital signature and qr code“, in *4TH International Conference on Computer Engineering and Technology (ICCET 2012)*, 2012.
- [20] G. Rhoads, P. Seder, M. Miller, B. MacIntosh, W. Hein und B. Hannigan, *Method of linking on-line data to printed documents*, US Patent 7,051,086, Mai 2006. Adresse: <https://www.google.com/patents/US7051086>.
- [21] J. Wolosewicz und A. Schmidt, *Document authentication and verification*, US Patent App. 10/057,297, Juli 2003. Adresse: <https://www.google.com/patents/US20030145206>.
- [22] C. Woodford. (2014). Optical character recognition, Adresse: [https://en.wikipedia.org/wiki/Optical\\_character\\_recognition](https://en.wikipedia.org/wiki/Optical_character_recognition) (besucht am 11.01.2017).
- [23] W. Authors. (2017). Ocr (optical character recognition), Adresse: <http://www.explainthatstuff.com/how-ocr-works.html> (besucht am 11.01.2017).
- [24] tutorialspoint Authors. (2017). Optical character recognition, Adresse: [https://www.tutorialspoint.com/dip/optical\\_character\\_recognition.htm](https://www.tutorialspoint.com/dip/optical_character_recognition.htm) (besucht am 11.01.2017).

- [25] G. I. Thomas Breuel. (2016). Python-based tools for document analysis and ocr, Adresse: <https://github.com/tmbdev/ocropy> (besucht am 12.01.2017).
- [26] J. Schulenburg. (2013). Gocr, Adresse: <http://jocr.sourceforge.net/index.html> (besucht am 12.01.2017).
- [27] W. Authors. (2017). Cuneiform, Adresse: <https://de.wikipedia.org/wiki/CuneiForm> (besucht am 12.01.2017).
- [28] I. Free Software Foundation. (2016). Ocrad - the gnu ocr, Adresse: <https://www.gnu.org/software/ocrad/> (besucht am 12.01.2017).
- [29] G. Inc. (2016). Tesseractocr, Adresse: <https://github.com/tesseract-ocr> (besucht am 30.11.2016).
- [30] W. Authors. (2017). Kryptologische hashfunktion, Adresse: [https://de.wikipedia.org/wiki/Kryptologische\\_Hashfunktion](https://de.wikipedia.org/wiki/Kryptologische_Hashfunktion) (besucht am 13.01.2017).
- [31] —, (2017). Rsa, Adresse: <https://en.wikipedia.org/wiki/RSA> (besucht am 13.01.2017).
- [32] —, (2017). Pki, Adresse: [https://en.wikipedia.org/wiki/Public\\_key\\_infrastructure](https://en.wikipedia.org/wiki/Public_key_infrastructure) (besucht am 13.01.2017).
- [33] —, (2017). Digital signature, Adresse: [https://en.wikipedia.org/wiki/Digital\\_signature](https://en.wikipedia.org/wiki/Digital_signature) (besucht am 13.01.2017).
- [34] D. Inc. (2017). Understanding digital signatures, Adresse: <https://www.docusign.com/how-it-works/electronic-signature/digital-signature/digital-signature-faq> (besucht am 13.01.2017).
- [35] W. Authors. (2017). Bitcoin, Adresse: <https://en.wikipedia.org/wiki/Bitcoin> (besucht am 13.01.2017).
- [36] E. Foundation. (2016). Ethereum, Adresse: <https://www.ethereum.org/> (besucht am 30.11.2016).
- [37] W. Authors. (2017). Smart contract, Adresse: [https://en.wikipedia.org/wiki/Smart\\_contract](https://en.wikipedia.org/wiki/Smart_contract) (besucht am 13.01.2017).
- [38] —, (2017). Merkle tree, Adresse: [https://en.wikipedia.org/wiki/Merkle\\_tree](https://en.wikipedia.org/wiki/Merkle_tree) (besucht am 06.01.2017).
- [39] —, (2017). Smart contract, Adresse: [https://en.wikipedia.org/wiki/Blockchain\\_\(database\)](https://en.wikipedia.org/wiki/Blockchain_(database)) (besucht am 13.01.2017).
- [40] dat. (2016). Dat project, Adresse: <https://datproject.org/> (besucht am 30.11.2016).
- [41] W. Authors. (2017). Kademlia dht, Adresse: [https://en.wikipedia.org/wiki/Mainline\\_DHT](https://en.wikipedia.org/wiki/Mainline_DHT) (besucht am 06.01.2017).
- [42] —, (2017). Kryptowährung, Adresse: <https://de.wikipedia.org/wiki/Kryptow%C3%A4hrung> (besucht am 06.01.2017).
- [43] P. Labs. (2016). Ipfs, Adresse: <https://ipfs.io/> (besucht am 30.11.2016).
- [44] S. L. Inc. (2016). Storj, Adresse: <http://www.boost.org/> (besucht am 30.11.2016).
- [45] vasild. (2016). Ipfs++, Adresse: <https://github.com/vasild/cpp-ipfs-api> (besucht am 30.11.2016).

- [46] Kitware.com. (2016). Cmake, Adresse: <http://www.cmake.org/> (besucht am 29.12.2016).
- [47] Wikipedia. (2017). Grasp, Adresse: [https://en.wikipedia.org/wiki/GRASP\\_\(object-oriented\\_design\)](https://en.wikipedia.org/wiki/GRASP_(object-oriented_design)) (besucht am 02.01.2017).
- [48] E. Gamma, R. Helm, R. Johnson und J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995, ISBN: 0-201-63361-2.
- [49] Wikipedia. (2017). Kiss, Adresse: <https://de.wikipedia.org/wiki/KISS-Prinzip> (besucht am 02.01.2017).
- [50] —, (2017). Exception safety, Adresse: [https://en.wikipedia.org/wiki/Exception\\_safety](https://en.wikipedia.org/wiki/Exception_safety) (besucht am 02.01.2017).
- [51] cppreference. (2017). Rule of 3/5/0, Adresse: [http://en.cppreference.com/w/cpp/language/rule\\_of\\_three](http://en.cppreference.com/w/cpp/language/rule_of_three) (besucht am 06.01.2017).
- [52] —, (2017). Pimpl idiom, Adresse: <http://en.cppreference.com/w/cpp/language/pimpl> (besucht am 06.01.2017).
- [53] —, (2017). Raii idiom, Adresse: <http://en.cppreference.com/w/cpp/language/raii> (besucht am 06.01.2017).
- [54] B. Community. (2016). Boost library, Adresse: <http://www.boost.org/> (besucht am 30.11.2016).
- [55] O. Community. (2016). Opencv, Adresse: <http://opencv.org/> (besucht am 30.11.2016).
- [56] A. Software. (2016). Ghostscript, Adresse: <http://www.ghostscript.com/doc/current/Lib.htm> (besucht am 30.11.2016).
- [57] O. S. Foundation. (2016). Openssl, Adresse: <https://www.openssl.org/> (besucht am 30.11.2016).
- [58] A. Kapoulkine. (2016). Pugixml, Adresse: <https://github.com/zeux/pugixml> (besucht am 30.11.2016).
- [59] nlohmann. (2016). Nlohmannjson++, Adresse: <https://github.com/nlohmann/json> (besucht am 30.11.2016).
- [60] J. Schauer. (2017). Img2pdf, Adresse: <https://gitlab.mister-muffin.de/josch/img2pdf> (besucht am 06.01.2017).
- [61] P. Nayuki. (2017). Qr code generator library, Adresse: <https://github.com/nayuki/QR-Code-generator> (besucht am 07.01.2017).
- [62] ZXing. (2017). Zxing, Adresse: <https://github.com/zxing/zxing> (besucht am 07.01.2017).
- [63] W3C. (2016). Xpath language, Adresse: <https://www.w3.org/TR/xpath/> (besucht am 04.01.2017).
- [64] P. Inc. (2010). Searchable pdfs, Adresse: <http://searchable-pdf.com/content.php?lang=en&c=61> (besucht am 02.01.2017).
- [65] Adobe. (2017). In pdf scannen, Adresse: <https://acrobat.adobe.com/de/de/acrobat/how-to/convert-jpeg-tiff-scan-to-pdf.html> (besucht am 09.01.2017).
- [66] Wikipedia. (2017). Regular expression, Adresse: [https://en.wikipedia.org/wiki/Regular\\_expression](https://en.wikipedia.org/wiki/Regular_expression) (besucht am 04.01.2017).

- [67] P. S. Foundation. (2017). Python c api, Adresse: <https://docs.python.org/3/c-api/> (besucht am 06.01.2017).
- [68] Wikipedia. (2017). Self-signed certificate, Adresse: [https://en.wikipedia.org/wiki/Self-signed\\_certificate](https://en.wikipedia.org/wiki/Self-signed_certificate) (besucht am 04.01.2017).
- [69] W. Authors. (2017). Indirection, Adresse: <https://en.wikipedia.org/wiki/Indirection> (besucht am 09.01.2017).
- [70] G. Inc. (2017). Gsoap framework, Adresse: <https://www.genivia.com/dev.html> (besucht am 06.01.2017).
- [71] Microsoft. (2017). Microsoft c++ rest sdk, Adresse: <https://github.com/Microsoft/cpprestsdk> (besucht am 06.01.2017).
- [72] G. Inc. (2017). Tesseract improve quality, Adresse: <https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality> (besucht am 06.01.2017).
- [73] —, (2017). Tesseract 4.0, Adresse: <https://github.com/tesseract-ocr/tesseract/wiki/4.0-Accuracy-and-Performance> (besucht am 06.01.2017).
- [74] —, (2017). Tesseractocr advanced api, Adresse: <https://tesseract-ocr.github.io/a01281.html> (besucht am 06.01.2017).
- [75] ABBYY. (2017). Abbyy finereader engine, Adresse: <https://www.abbyy.com/> (besucht am 09.01.2017).
- [76] Leadtools. (2017). Leadtools ocr, Adresse: <https://www.leadtools.com/> (besucht am 09.01.2017).
- [77] Adobe. (2017). Adobe acrobat sdk, Adresse: <https://www.adobe.com/devnet/acrobat/overview.html> (besucht am 09.01.2017).
- [78] ABBYY. (2017). Abbyy finereader engine, Adresse: <https://www.abbyy.com/de-de/ocr-sdk-linux/ocr-stages/ocr/> (besucht am 09.01.2017).
- [79] Leadtools. (2017). Leadtools ocr, Adresse: <https://www.leadtools.com/sdk/professional-ocr> (besucht am 09.01.2017).
- [80] Omnipage. (2017). Omnipage ocr, Adresse: <http://www.nuance.de/for-business/by-product/omnipage/standard/index.htm> (besucht am 09.01.2017).
- [81] Microsoft. (2017). Microsoft cognitive services, Adresse: <https://www.microsoft.com/cognitive-services/en-us/computer-vision-api> (besucht am 09.01.2017).
- [82] C. Sulzberger. (2016). Levenstein-distance, Adresse: <http://www.levenshtein.de/> (besucht am 21.12.2016).
- [83] D. Admassu. (2017). Unicode optical character recognition, Adresse: <https://www.codeproject.com/kb/recipes/unicodeocr.aspx> (besucht am 09.01.2017).
- [84] K. Alaris. (2014). Userguide capture pro 4 en, Adresse: [http://www.kodakalaris.com/-/media/files/di/uploadedfiles/a61735\\_usersguide\\_capturepro4\\_en.pdf](http://www.kodakalaris.com/-/media/files/di/uploadedfiles/a61735_usersguide_capturepro4_en.pdf) (besucht am 17.01.2017).
- [85] A. D. et al. (2017). Libharu, Adresse: <https://github.com/libharu/libharu> (besucht am 09.01.2017).



- [86] D. S. et al. (2017). Podofo, Adresse: <http://podofo.sourceforge.net/about.html> (besucht am 09.01.2017).
- [87] J. Grešula. (2017). Jagpdf, Adresse: <http://www.jagpdf.org/features.htm> (besucht am 09.01.2017).
- [88] S. C. Foundation. (2017). C++ concurrency, Adresse: <https://isocpp.org/wiki/faq/cpp11-library-concurrency> (besucht am 09.01.2017).
- [89] G. Inc. (2017). Google diff-match-patch bibliothek, Adresse: <https://code.google.com/p/google-diff-match-patch/> (besucht am 06.01.2017).
- [90] E. Foundation. (2016). Ethereum, Adresse: <https://github.com/ethereum/cpp-ethereum> (besucht am 30.11.2016).