NORDAKADEMIE

Masterthesis

im Studiengang
Angewandte Informatik / Software Engineering (M.Sc.)

Konzept eines universellen, dezentralen Software Repositorys und dessen Adaption im NPM-Frontend-Ökosystem

Fan Xu

Matrikelnr.: 10638

Gutachter
Prof. Dr. Jan Himmelspach

Co-Gutachter
Prof. Dr. José Baltasar Trancón Widemann

Abgabetermin
17. Februar 2022

Inhaltsverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface
AUR Arch User Repository

BTFS BitTorrent File System

CDN Content Delivery Network
CLI Command Line Interface
CORS Cross-Origin Resource Sharing
CRUD Create, Read, Update and Delete
CSS Cascading Style Sheets

DAO Decentralized Autonomous Organization
dApp Decentralized Application
DeFi Decentralized Finance
DMCA Digital Millennium Copyright Act
dRepo Decentralized Repository
dVPN Decentralized Virtual Private Network

ESM ECMAScript Modules
ethPM Ethereum Package Manager
EVM Ethereum Virtual Machine

GPG GNU Privacy Guard

HTTP HyperText Transfer Protocol
HTTPS HyperText Transfer Protocol Secure

I2P Invisible Internet Project
IPFS InterPlanetary File System
ISP Internet Service Provider
IT Informationstechnik

JSON-RPC JavaScript Object Notation Remote Procedure Call
JVM Java Virtual Machine

NFT Non-Fungible Token
NPM Node Package Manager

OCI Open Container Initiative

P2P Peer-to-Peer
PyPI Python Package Index

RIAA Recording Industry Association of America

SASS Syntactically Awesome Style Sheets
SBOM Software Bill of Materials
SemVer Semantic Versioning
SSH Secure Shell

URL Uniform Resource Locator

VCS Version Control System
VPN Virtual Private Network

Abbildungsverzeichnis

Quellcodeverzeichnis

1. Einführung

1.1. Problemstellung

Software Repositories wie Maven Central, Docker Hub, GitHub oder die NPM Registry stellen Entwicklern und Anwendern Bibliotheken und Applikationen zur Verfügung. Sie fördern die Wiederverwendung von erprobter Software und erleichtern so den Austausch zwischen Entwicklern. Sie sind das zentrale Einfallstor ihrer jeweiligen Community, um zuverlässige Lösungen zu finden, auf denen Applikationen aufgebaut werden können. Als essentielles Bindeglied in der Software Supply Chain baut die Softwareentwicklung fundamental auf Software Repositories auf. Es wird fast blind auf ihre Verfügbarkeit, Authentizität und Ehrlichkeit vertraut. Ausfälle und andere Disruptionen können weitreichende Auswirkungen haben oder gar die Entwicklung teilweise lahmlegen. Es ist eine untrennbare Abhängigkeit entstanden.

Bei Software Repositories wie der NPM Registry oder GitHub handelt es sich um zentralisierte Systeme, die durch ggf. kommerzielle Entitäten oder Organisationen betrieben werden. Sie müssen daher wirtschaftlich agieren und sich innerhalb eines gesetzlichen Rahmenwerks bewegen[1]. Die Organisationen sind dementsprechend verantwortlich für die Inhalte der jeweiligen Repositories und müssen sich für diese gegenüber Dritten verantworten. Sie müssen sich an lokale Gesetze halten und ethische Grundsätze verfolgen. Dies hat zur Folge, dass die Betreiber letztlich bestimmen und beschränken müssen, welche Inhalte in ihren verwalteten Repositories publiziert werden dürfen. Dies verleiht den Betreibern eine besondere Machtposition, insbesondere wenn das jeweilige Repository eine Monopolstellung einnimmt.

Nutzer der Repositories vertrauen den frei zugänglichen Softwarepaketen. Dies beinhaltet, dass sie davon ausgehen, dass das Repository die Programmbibliotheken unverändert, wie vom Autor geschaffen, an sie verteilt. Gleichzeitig bauen sie darauf, dass die Autoren einer Software keine schädigenden Absichten verfolgen[2].

[1] Vgl. Tsitoara 2019, S. 95, S. 104.
[2] Vgl. Bisht et al. 2017, S. 7-8, S. 27.

In der jüngsten Vergangenheit kam es jedoch zu Vorfällen, die das Vertrauen von Nutzern in Software Repositories und die dort bereitgestellten Inhalte auf die Probe stellte.

Im Oktober 2020 wurde das Code Repository des youtube-dl[3] Projekts sowie dessen Forks durch GitHub gesperrt. Dies wurde durch eine DMCA Takedown Notice seitens der Recording Industry Association of America (RIAA) ausgelöst. Der Code durfte einige Wochen nicht auf GitHub publiziert werden und wurde letztlich nach genauer Prüfung der Takedown-Anfrage wieder zugelassen[4].

Im Januar 2022 wurde durch den Entwickler zweier populärer Programmbibliotheken, faker.js und colors.js, absichtlich Schadcode in diese eingefügt. Diese Änderungen wurden durch neue Versionen auf GitHub und in die NPM Registry publiziert. Dies hatte zur Folge, dass die Benutzerkonten des Entwicklers gesperrt und die kompromittierten Releases seitens der NPM Registry entfernt wurden. Weiterhin löschte der Entwickler danach den Code eines der Projekte auf GitHub[5].

Diese und ähnliche Ereignisse gefährden die globale Software Supply Chain. Wenn Softwarepakete und Quellcode plötzlich nicht mehr frei zur Verfügung stehen oder gar existieren, sind die darauf aufbauenden Programmbibliotheken und Applikationen ebenfalls betroffen[6]. Auch die nachträgliche Änderung von veröffentlichten Softwareversionen kann zu Konflikten und Sicherheitsproblemen führen.

Da Software Repositories eine zentrale Rolle in der Softwareentwicklung einnehmen, entsteht ein Single Point of Failure. Es stehen zwar alternative Bezugsmöglichkeiten zur Verfügung, jedoch weichen sie vom Mainstream ab und finden daher in der Masse wenig Verwendung.

Neben der technischen Verfügbarkeit eines zentralen Repositories stellen vor allem nicht-technische Faktoren eine Bedrohung für dessen Inhalte dar. Jede Zensur durch Politik oder lokale Gesetze beeinflusst Nutzer global. Manipulation der angebotenen Softwarepakete durch beispielsweise die Autoren oder die Betreiber des Software Repositorys können Schadcode in eine größere Menge an abhängigen Applikationen einschleusen. Dies könnte die globale Infrastruktur gefährden.

[3] youtube-dl ist ein Open-Source-Werkzeug, dass es Nutzern erlaubt, Videos und Audioinhalte von YouTube und anderen Medienplattformen herunterzuladen und damit offline verfügbar zu machen.
[4] Vgl. Vollmer 2020.
[5] Vgl. Sharma 2022.
[6] Vgl. Soto-Valero et al. 2019, S. 341-342.

Softwareindustrie und Regierungen ist mittlerweile die Bedeutung der Software Supply Chain bewusst[7]. Beispielsweise werden Sicherheitslücken in enger Kooperation aller Stakeholder verfolgt und geschlossen. Jedoch liegen große, essentielle Repositories wie die NPM Registry oder Docker Hub in den Händen einzelner Unternehmen, die ihre eigene Agenda verfolgen und daher nicht immer im Sinne der globalen Community handeln müssen.

Es ist die Meinung der Verfasserin, dass Software Repositories eine so wichtige Position einnehmen, dass sie für das Allgemeinwohl unabhängig und sicher gestaltet sein müssen. Sie müssen hochverfügbar, manipulations- und zensursicher sowie für jeden Menschen frei nutzbar sein.

Zentralisierte Systeme sind nicht in der Lage, diese Anforderungen zu erfüllen. Dezentrale Systeme zeichnen sich hingegen gerade durch diese Eigenschaften aus. Software Repositories, die im Sinne dezentraler Software erstellt wurden, könnten eine zuverlässige Alternative zu existierenden Systemen darstellen und diese schließlich ersetzen[8].

In den letzten Jahren haben dezentrale Bewegungen wie DeFi eine Vielzahl an Nutzern und Kapital angezogen. Sie bieten transparente Systeme, die für jeden nutzbar sind. Gleichzeitig werden neue Konzepte und Ideen entwickelt und umgesetzt, um nicht nur dezentrale Konstrukte effizienter und sicherer zu gestalten[9]. Öffentliche Software Repositories können von diesen Entwicklungen profitieren und die Software Supply Chain dadurch stärken, dass sie zuverlässig und im Sinne des Allgemeinwohls Software verteilen.

So wie Kunst und Kultur ist auch Software im Endeffekt ein intellektuelles Gemeingut der Menschheit, das geschützt und bewahrt werden muss. Dezentrale Repositories könnten ein Weg sein, dies zu erreichen.

1.2. Zielsetzung und Forschungsfragen

Im Rahmen dieser Masterthesis wird ein Konzept für ein dezentrales Software Repository und sein umgebendes Ökosystem erstellt. Das Repository soll allgemein anwendbar sein, sodass es für die meisten Software-Ökosysteme ohne oder mit minimalen Anpassungen verwendet werden kann.

[7] Vgl. Biden 2021.
[8] Vgl. Raval 2016, S. 3-4.
[9] Vgl. Oreizy 1999, S. 730-731.

Das Konzept soll demonstrieren, dass bestehende Software Repositories verschiedener großer Software-Ökosysteme dezentralisiert und gleichzeitig in eine generalisierte Form überführt werden können. Zudem werden notwendige weitere Komponenten und Konzepte identifiziert, die das ausgearbeitete Repository unterstützten, den Anforderungen der Dezentralisierung Genüge zu tun. Gleichzeitig ist zu beachten, wie existierende Infrastrukturbestandteile unter möglicher Anpassung integriert und wiederverwendet werden können.

Die Basis des Konzepts wird durch eine Betrachtung des Begriffs Dezentralisierung und dezentralisierter Systeme sowie durch eine detaillierte Untersuchung einer Reihe verschiedener, bestehender Software Repositories gebildet. Es werden zentrale und dezentrale Systeme gegenübergestellt und auf diese Weise die Anforderungen im Sinne der Dezentralisierung an ein potentielles Repository identifiziert. Die Analyse existierender Software Repositories erlaubt die Extraktion gemeinsamer Eigenschaften sowie von Alleinstellungsmerkmalen. Mithilfe dieses Katalogs lässt sich eine Generalisierung durchführen, die die Grundlage weiterer Anforderungen bildet. Aus der Gesamtheit der Anforderungen kann im Anschluss ein Konzept für den Kern eines dezentralen Repositorys abgeleitet werden.

Dieses Kern-Repository umfasst allerdings nur ein minimales Feature-Set. Um das System praktikabler zu gestalten und gleichzeitig das Potential der Dezentralisierung zu nutzen, werden notwendige Erweiterungen im Umfeld des Kern-Repositories diskutiert. Sie konzentrieren sich auf die weitere Absicherung der Repository-Inhalte im Sinne der Authentizität und des Vertrauens in die Software selbst. Diese Komponenten stellen zusammen die Gesamtheit eines dezentralen Repositorys dar.

Anschließend wird eine potentielle Einführung des dezentralen Repositorys in das NPM-Frontend-Ökosystem skizziert. Migration, Entwicklungsanpassungen und Einflüsse auf den Betrieb von Webanwendungen werden erörtert. Etwaige Probleme und Benefits für Entwickler und Anwender werden identifiziert und ansatzweise diskutiert.

Basierend auf dieser Zielsetzung lässt sich folgende Forschungsfragestellung beantworten:

> *Kann ein dezentrales, universelles Software Repository konzipiert werden, welches existierende Systeme schließlich ersetzen könnte? Welche Features und Komponenten sind hinreichend? Und welche Probleme und Vorteile können durch eine modellhafte Adaption des konzipierten Repositorys anhand des NPM-Frontend-Ökosystems erkannt werden?*

1.3. Aufbau der Arbeit

Um die Forschungsfrage beantworten und das Konzept eines dezentralen Repositories erstellen zu können, werden zum Anfang dieser Arbeit in Kap. 2 die Begriffe Zentralisierung und Dezentralisierung gegenübergestellt. Es werden zunächst beispielhaft zentrale Systeme in Wirtschaft, Politik und Softwareentwicklung dargereicht. Anschließend werden entsprechende Exemplare der Dezentralisierung in Kap. 2.2 aufgezählt. Schließlich werden die Eigenschaften dieser miteinander verglichen.

Im Anschluss werden in Kap. 3 existierende Software Repositories betrachtet. Dies beginnt mit der Erklärung von Grundbegriffen sowie von Prozessen und Workflows in diesem Rahmen. Danach wird die Vergleichsanalyse existierender Repositories betrachtet. Hier werden die gewählten Exemplare kurz in ihren Ökosystemen und Eigenschaften beschrieben und die Fokuspunkte der Analyse ergründet. In Kap. 3.3 sind die Ergebnisse der Analyse zu finden. Sie werden in Gemeinsamkeiten und Unterschiede unterteilt.

Aufbauend auf diesen Grundlagen wird in Kap. 4 das Konzept des universellen, dezentralen Repositorys entwickelt. Hierzu werden zunächst in Kap. 4.1 Requirements an das neue Repository erfasst. Diese sind unterteilt nach Anforderungen, die aus den Erkenntnissen der Analyse existierender Repository-Systeme stammen, und Anforderungen, die aus den Eigenschaften der Dezentralisierung hergeleitet wurden. Datensicherheit spielt eine essentielle Rolle in dem Konzept, die in Kap. 4.2 tiefgehender besprochen wird. Diese Grundlagen erlauben die Darlegung des Konzepts in Kap. 4.3. Zu Anfang wird das Kern-Repository beschrieben. Nach einer kurzen Übersicht wird auf die Hauptkomponenten Index und Storage im Detail eingegangen. Abschließend folgt an dieser Stelle eine Begründung für die Wahl der Technologien und die Aufteilung des Systems. Kap. 4.3.2 schließt an diesem Punkt an und erläutert Systemerweiterungen, die zu einem ganzheitlichen Konzept führen. Es werden die Kernkomponenten Index und Storage erneut besprochen und die Grundideen eines Build Verification Networks sowie eines Review Networks vorgestellt. Nach der Beschreibung des Konzepts wird in Kap. 4.4 abschließend die Erfüllung der Anforderungen überprüft.

In Kap. 5 wird das erstellte Konzept modellhaft auf das NPM-Frontend-Ökosystem übertragen. Hierzu wird zunächst der Aufbau und die Zusammensetzung von Frontends moderner Webanwendungen erläutert. Darauf folgend wird in Kap. 5.2 die mögliche Migration der Inhalte der NPM Registry in das dezentrale Repository skizziert.

Anschließend werden mögliche Anpassungen an Entwicklungsprozessen mit dem neuen Repository beleuchtet. Eine Skizzierung der Datenverteilung und Sicherheitsvorteile aus Entwicklersicht folgen danach. In Kap. 5.4 werden mögliche Anwendungskonsequenzen aus Endanwendersicht besprochen. Zunächst wird die Notwendigkeit einer Software Bill of Materials verdeutlicht, wenn das dezentrale Repository in Applikationen bis zum Anwender genutzt wird. Hieraus ergeben sich Netzwerkeffekte und Vorteile bei der Verifikation von Anwendungen sowie die Möglichkeit, diese zu anzupassen. Abschließend werden notwendige Applikations- und Browseranpassungen angedeutet.

Ein kritischer Ausblick auf das Thema erfolgt in Kap. 6. Es wird zunächst der Reifegrad der eingesetzten Technologien besprochen und welchen Einfluss diese und neue Mechanismen auf die Akzeptanz der Anwender haben. Weiterhin wird die Sicherheit des Repositorys und dessen Inhalte thematisiert. Mögliche rechtliche Folgen und Probleme durch die Nutzung eines dezentralen Repositorys folgen im Anschluss. Weiterhin stellt die Datenverfügbarkeit eine Komplikation dar, die abschließend besprochen wird.

2. Zentralisierung und Dezentralisierung

2.1. Zentralisierte Systeme

Generell handelt es sich bei einem zentralisierten System um eine Ordnung, in der eine Entität die gesamte Kontrolle über die Funktionalität des Systems besitzt[10]. Die Entscheidungen, um das Systemziel zu erreichen, werden mittels eines zentralen Mechanismus getroffen bzw. ermittelt.

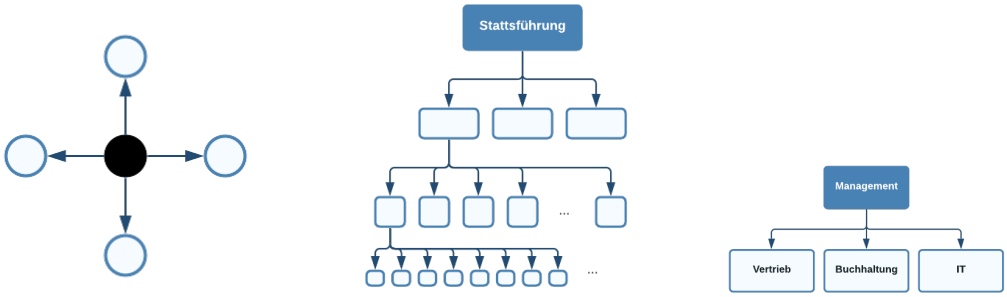
Der Begriff der Zentralisierung kommt häufig in verschiedenen Bereichen von Politik und Wirtschaft zur Geltung. Die zentrale Staatsverwaltung wird beispielsweise in autoritären Staaten besonders deutlich. Bei diesen liegt die gesamte Macht bei einer oder nur wenigen staatlichen Stellen. Moderne Diktaturen, das Nazi-Regime in Deutschland oder auch absolute Monarchien, wie sie in der Vergangenheit in Europa existieren, sind die bekanntesten Beispiele. Allerdings sind Zentralisierungsansätze auch in demokratischen Staatsformen zu finden. Sie werden beispielsweise durch Hierarchien in Staatsorganen ausgedrückt, die mit höherer Einstufung eine höhere Autorität und damit mehr Einfluss besitzen. Föderalistische Staaten arbeiten teilweise nach diesem Prinzip. Der Grad dieser Konzentration von Macht bleibt jedoch zu diskutieren[11].

Zentralisierung ist ebenfalls innerhalb von Unternehmen aufzufinden. Diese ist bereits in den Organisationsformen mitunter definiert, da bestimmte Organe und ihre Aufgaben gesetzlich verankert sind[12]. Die Organisationsformen regeln unter anderem die Verantwortlichkeit der Organe im Unternehmen, gegenüber Staat und anderen Marktteilnehmern und erzeugen homogene Strukturen in der Wirtschaft. Weiterhin bilden sich auch am Markt implizite, zentrale Ordnungen durch Monopole oder Oligarchien, die Preise und andere Marktkonditionen bestimmen und Konkurrenz verdrängen können.

[10] Vgl. Khare 2002, S. 5.
[11] Vgl. Benz 2011, S. 13-15, S. 47.
[12] Vgl. Berndt et al. 2016, S. 555-559.

Es ist anzumerken, dass hoch zentralisierte Systeme, besonders in Politik und Wirtschaft, in der Gesellschaft als negativ und geradezu dystopisch angesehen werden[13]. Jedoch ist Zentralisierung im alltäglichen Leben überall vorzufinden, und stellt eine der einfachsten und effizientesten Organisationsstrukturen dar.

Abb. 1 zeigt zentralisierte Konstrukte schematisch. 1a illustriert modellhaft Zentralisierung zwischen Gleichberechtigten. Der zentrale Knoten in Schwarz besitzt die Entscheidungsgewalt über die anderen Knoten. Eine diktaturähnliche Staatsführung ist in 1b abgebildet. Das Staatsoberhaupt übt seine Macht über seine Untergebenen aus, welche hierarchisch aufgestellt sind. In 1c ist dasselbe Prinzip auf einen Unternehmenskontext bezogen worden. Das Management bestimmt über die einzelnen Abteilungen eines Unternehmens.



(a) Allgemeine Zentralisierung

(b) Zentralisierung im Politik

(c) Zentralisierung im Unternehmen

Abbildung 1: Beispiele für Zentralisierungsmodelle

In der IT sind ebenfalls zentral organisierte Strukturen anzutreffen. Sie werden beispielsweise auf Systemarchitekturebene zur Kontrollflusssteuerung, in der Datenhaltung oder auch in Design Patterns, wie dem Singleton-[14] oder Mediator-Pattern[15], angewendet. Eine einzelne Datenbankinstanz auf einem einzelnen Server ist eines der simpelsten Beispiele für zentralisierte Datenhaltung. Alle Daten werden an einem einzigen Ort gespeichert und gepflegt. Lesende und schreibende Zugriffe können zwar von außerhalb erfolgen, dennoch entscheidet das Datenbanksystem letztlich, welche Inhalte tatsächlich gespeichert oder gelesen werden dürfen.

Auf ähnliche Weise stellen auch CRUD-Webanwendungen zentralisierte Systeme dar. Das Web-Frontend wird zwar in den Browsern einer Menge von Anwendern ausgeführt, dennoch stellt das Backend die Kontrollinstanz dar. Auch wenn das Backend

[13] Vgl. Benz 2011, S. 19-13.
[14] Vgl. Gamma et al. 1995, S. 144-145.
[15] Vgl. Gamma et al. 1995, S. 305-310.

aus einer Vielzahl horizontal skalierender Server besteht, welche die Anwendung verteilt betreiben, arbeiten sie als eine Einheit gegenüber den Clients.

Master-Slave-Architekturen sind ebenfalls als zentralisiert anzusehen. Ein Master-Knoten kontrolliert eine Menge an Slave-Knoten, welche Dienste für den Master ausführen[16]. Im Datenbankkontext werden die Slaves auch als Read-only-Replica bezeichnet. Sie kopieren im simpelsten Fall den gesamten Datenstand vom Master und können die Daten den Clients zur Abfrage anbieten. Datenänderungen können nur auf dem Master erfolgen. Clients müssen schreibende Aktionen daher gegen diesen durchführen. Die Slaves gleichen anschließend den Datenstand mit dem Master erneut ab. Der Master besitzt in diesem Szenario die absolute Kontrolle über die Inhalte des Datenbanksystems sowie alle Zustandsänderungen.

2.2. Dezentralisierte Systeme

Das Gegenstück zur Zentralisierung ist die Dezentralisierung. Grundsätzlich unterscheiden sich diese beiden dadurch, dass die Verantwortung und Entscheidungsgewalt an eine Vielzahl von Entitäten verteilt wird. Es wird versucht, die Macht von einer oder mehreren zentralen Stellen weiterzugeben. Die Teilnehmer, die die Verantwortung für einen (Teil-)Bereich oder das gesamte System tragen, sind in der Regel gleichberechtigt und müssen gemeinsam zu einem Konsens bei jeder Entscheidungsfindung kommen. Wie ein solcher Konsens aussieht, muss durch das Systemdesign geregelt sein, beispielsweise kommen hier Einheits- oder Mehrheitsentscheide zum Einsatz[17].

In einem dezentralisierten Unternehmen könnten die Zuständigkeiten und Verantwortung mit den Mitarbeitern geteilt werden. Jede Entscheidung, die typischerweise das Management trifft, würde durch alle Mitarbeiter gemeinsam beschlossen werden[18]. Zentrale Führungsrollen müssten in diesem Falle nicht mehr existieren bzw. hätten keine Bedeutung mehr.

Auf politischer Ebene findet Dezentralisierung ebenfalls Anwendung, beispielsweise in demokratischen Staaten. Die Basisdemokratie stellt eine hoch dezentralisierte Form der Demokratie dar, da alle Entscheidungen durch das Volk direkt getroffen werden. Die repräsentative Demokratie nutzt Elemente der Zentralisierung, beispielsweise Parlamente mit Abgeordneten, um Entscheidungsfindungen praktikabler

[16] Vgl. Starke 2015, S. 109-110.
[17] Vgl. Khare 2002, S. 1-2.
[18] Vgl. Berndt et al. 2016, S. 555-559.

zu gestalten. Dies bedeutet, dass jeder Bürger in einer repräsentativen Demokratie prinzipiell nur indirekt an der Entscheidungsfindung beteiligt ist[19,20]. Weiterhin können in einem föderativen Staat viele Entscheidungen weitgehend unabhängig von der zentralen Regierung von selbst verwalteten Städten und Gemeinden getroffen werden. Dies dient in diesem Fall dazu, einerseits Entscheide eher bürgernah zu treffen und andererseits die zentrale Verwaltung nicht mit für sie eher irrelevanten Themen zu belasten.

Zentralisierung und Dezentralisierung lassen sich in Gerade unterscheiden. Ein Alleinherrscher, der alle Entscheidungen trifft, existiert in einem maximal zentralisierten System. Hingegen ist eine Organisation maximal dezentralisiert, wenn alle Entscheidungen von allen teilnehmenden Parteien zusammen getroffen werden. Beide Varianten wären in der Wirklichkeit wenig praktikabel und ineffizient. Daher besitzen reale Systeme in der Regel sowohl zentral- als auch dezentralisierte Komponenten bzw. Eigenschaften. Ein Markt, in dem eine Oligarchie mehrerer Unternehmen existiert, besitzt einen höheren Grad der Zentralisierung als ein föderalistischer Staat, der eher als dezentral organisiert angesehen wird, da der Staat zentrale Organe für seine Verwaltung einsetzt.

Dezentralisierung findet auch im IT-Kontext Anwendung. Eines der typischen Beispiele sind Peer-to-Peer-Netzwerke wie BitTorrent oder das InterPlanetary File System (IPFS). In diesen existieren direkte Verbindungen zwischen gleichberechtigten bzw. gleichbehandelten Teilnehmern. Diese Art der Kommunikation wird auch als Querkommunikation bezeichnet, wodurch die Zusammenarbeit der Knoten im Netz effizienter werden kann. Jeder Knoten speichert die Daten, die ihn interessieren und bietet diese zum Verteilen an (Seeding)[21,22]. Mithilfe von Distributed Hash Tables können gewünschte Daten und ihre Lokation im Netzwerk gefunden und mit den entsprechenden Nodes ausgetauscht werden. Auf diese Weise entsteht ein Datennetzwerk ohne zentrale Autorität, dessen Gesamtzustand durch die Gesamtheit der geteilten Daten aller teilnehmenden Nodes definiert ist[23].

Grundsätzlich zeichnet sich ein dezentrales IT-System dadurch aus, dass keine bestimmende Autorität existiert. Stattdessen sind Protokolle und Systemregeln definiert, nach denen Teilnehmer agieren müssen. Da nur ein Rahmenwerk definiert ist[24], erlaubt dies die Nutzung eigener Implementationen, solange sie diesen Bestim-

[19] Vgl. Rauch 2009, S. 8-9.
[20] Vgl. Benz 2011, S. 162-163.
[21] Vgl. Raval 2016, S. 17-19.
[22] Vgl. Bharambe et al. 2005, S. 2-3.
[23] Vgl. Benet 2014, S. 1-2.
[24] Vgl. Starke 2015, S. 300.

mungen nachkommen. Die Implementationen kann ein Anwender daher frei nach seinen Präferenzen wählen. Sofern sich ein System auf einen Gesamtzustand einigen muss, können auch Konsensverfahren eine Rolle spielen.

2.3. Zentralisierung und Dezentralisierung in der IT

2.3.1. Gegenüberstellung der Systeme in der IT

Zentrale und dezentrale Systeme unterscheiden sich nicht nur in ihrer Organisation. Nicht-funktionale Eigenschaften und entwicklungsseitige Konsequenzen sind ebenfalls zu berücksichtigen.

Der bedeutendste Unterschied zwischen den beiden Systemorganisationen liegt in der Implementatierung von Kontrolle. Bei zentralen Systemen geht die Kontrolle von einer einzigen, exklusiven Autorität aus. Diese Autorität kann Änderungen am Zustand des Systems vornehmen und ggf. andere Maßnahmen ergreifen. Alle anderen Knoten in dem Gefüge müssen der Autorität Folge leisten. Es handelt sich hierbei um eine unidirektionale Kontrollstruktur, in welcher die Kontrolleinheit über Ressourcen und Prozesse im Gesamtsystem entscheidet. Sie stellt damit einen Single Point of Truth dar[25].

In einem dezentralen System existiert hingegen keine einzelne Autoritätseinheit, die über den gesamten Zustand entscheidet. Stattdessen sind zumindest mehrere Kontrolleinheiten anzufinden, die unabhängig voneinander ggf. ihre eigenen Ziele verfolgen. Gemeinsam müssen sie allerdings über den Systemzustand entscheiden. Etwaige Meinungsverschiedenheiten müssen entsprechend über Konsensverfahren gelöst werden, sodass eine gemeinsame Entscheidung getroffen werden kann. Dieser Zustand hat anschließend für alle Systemteilnehmer Geltung. Dementsprechend existiert kein Single Point of Truth. Der Zustand des Systems ist zumindest über alle Kontrolleinheiten verteilt, wenn nicht sogar über das gesamte System[26].

Diese Konstellation von Kontrolle führt dazu, dass in zentralisierten Systemen ein Single Point of Failure entsteht. Fällt die Kontrolleinheit aus, ist das gesamte System lahmgelegt. Dieses Problem kann jedoch mithilfe von Backups und Mirroring mitigiert werden[27]. So wird eine höhere Verfügbarkeit hergestellt. In dezentralen Systemen geht die Kontrolle jedoch von einer größeren Menge an Autoritätsknoten aus.

[25] Vgl. Aggarwal und Kumar 2021, S. 8-9.
[26] Vgl. Aggarwal und Kumar 2021, S. 11-14.
[27] Vgl. Federal Aviation Administration 1988, S. 2-3.

Wenn ein oder mehrere dieser Knoten nicht mehr verfügbar sind, können die übrigen das System noch weiterhin verwalten. Dies kann jedoch zu einer geringeren Servicequalität führen. Erst wenn alle Kontrollknoten ausfallen, ist das gesamte System nicht mehr betriebsfähig. Daher ist durch das Konzept von Natur aus eine höhere Verfügbarkeit implementiert und es ist auch toleranter gegenüber Fehlern[28].

Der Single Point of Failure in zentralen Systemen erweist sich auch gleichzeitig als Sicherheitsproblem. Diese Konzentration von Kontrolle stellt ein geeignetes Angriffsziel dar, um das gesamte System zu beeinflussen. Denial-of-Service-Attacken können zum Ausfall der Autoritätseinheit und folgend des gesamten Systems führen[29]. Kompromittierte Kontrollelemente können verwendet werden, um den Zustand des Systems zu manipulieren. Dezentrale Systeme besitzen diese Verwundbarkeit gewöhnlich nicht. Hingegen können möglicherweise Datenschutzprobleme auftauchen, da die Nodes in der Regel offen miteinander kommunizieren und es sich bei Knoten ggf. um böswillige Teilnehmer handelt. Ebenso sind 51%-Attacken möglich, wenn die Mehrheit der Autoritätsknoten durch eine Partei kontrolliert wird[30]. Diese kann so ihre Agenda im System durchsetzen.

Weiterhin könnte Skalierung ein Problem für zentrale Systeme darstellen, wenn die Autoritätsknoten nicht entsprechend mitskalieren können. Mit steigender Systemgröße wird dessen Verwaltung ressourcenintensiver. Dadurch kann die Gesamtgröße des Systems eingeschränkt sein, wenn die Abhängigkeit von der zentralen Kontrolle zu hoch ist oder die Servicequalität sinkt, indem Anfragen langsamer abgearbeitet werden. Dezentrale Konstrukte können von der wachsenden Systemgröße profitieren, indem beispielsweise mehr Ressourcen zur Verfügung stehen und entsprechend mehr Endpunkte genutzt werden können[31]. Muss allerdings ein Gesamtzustand explizit beschlossen werden, kann dies mit einer größeren Menge an Kontrollknoten zu einem komplexeren Problem werden, was zu höheren Latenzen führen kann[32].

Entwicklungsseitig ist die Erstellung einer zentralisierten Applikation zumindest anfangs weniger aufwendig als in einem dezentralisierten System. Kommunikation zwischen Komponenten folgt in der Regel einem bestimmten Fluss und kann mit geringerem Umfang an Infrastruktur umgesetzt werden, wenn Redundanzen zunächst nicht betrachtet werden. Da weiterhin alle Komponenten durch die Entwicklung verantwortet werden, können Anpassungen und Aktualisierungen kontrolliert durchgeführt werden.

[28] Vgl. L. Chen et al. 2021, S. 7-11.
[29] Vgl. Eckert 2018, S. 121-122.
[30] Vgl. Eckert 2018, S. 839-841.
[31] Vgl. Bharambe et al. 2005, S. 2-3.
[32] Vgl. Abadi und Brunnermeier 2018, S. 1-3.

Dezentrale Systeme besitzen initial bei einer Neuentwicklung einen erhöhten Entwicklungsaufwand. Das Kommunikationsprotokoll und mögliche Schnittstellen müssen frühzeitig fest definiert werden, wenn Knotenimplementationen von Dritten im System erlaubt sein sollen. In diesem Zuge sind auch mögliche Vertrauens- und Konsenskonzepte zu erstellen, die für die Sicherheit der Anwendung notwendig sind. Da die Anwendung nicht nur verteilt, sondern auch möglicherweise mit einer Vielzahl von dritten Parteien betrieben wird, erhöhen sich Aufwendungen für Wartung sowie Weiterentwicklung und Anpassung des Systems. Diese benötigen entsprechenden Vorlauf zur Kommunikation und Abstimmung, um den Zustand des Konstrukts zur Laufzeit nicht zu gefährden[33].

Die Verwendung eines zentralisierten Systems eignet sich in Umgebungen und Szenarien, in denen eine Autoritätsentität die komplette Kontrolle über ein System besitzen soll. Unternehmenswebseiten und -Anwendungen, ERP-Systeme oder Kubernetes-Cluster wären Beispiele hierfür. Soll hingegen die Kontrolle zwischen mehreren Parteien geteilt werden bzw. sie sich gegenseitig überprüfen, eignen sich dezentrale Systeme. In Peer-to-Peer-Filesharing-Netzwerken ist die Kontrolle gewissermaßen über alle verbundenen Knoten verteilt, indem jeder zum Netzwerk seine Daten beiträgt. Permissioned Blockchains, die zwischen mehreren Unternehmen zum Informationsaustausch verwendet werden, spalten die Kontrolle über alle Teilnehmer auf. Sie überprüfen sich gegenseitig, sodass keine absolute Vertrauensbeziehung zueinander notwendig ist[34].

2.3.2. Differenzierung von Verteilung

Bei verteilten Systemen handelt es sich allgemein um Anwendungen, die auf einer Mehrzahl an Computern betrieben werden und sich gleichzeitig dem Anwender gegenüber als ein einzelnes System präsentieren. Dies bedeutet, dass sowohl zentralisierte als auch dezentralisierte Systeme verteilt sein können. Zentralisierte Systeme können jedoch auch sinnvoll durch eine einzelne Instanz ausgeführt werden, beispielsweise eine einzelne Datenbankinstanz. Dies ist bei dezentralen Systemen nicht der Fall. Sie müssen verteilt betrieben werden, um den Eigenschaften der Dezentralisierung Genüge zu tun, beispielsweise der verteilten Kontrolle[35].

[33] Vgl. Raval 2016, S. 40-59.
[34] Vgl. Gatteschi et al. 2018, S. 68-69.
[35] Vgl. Aggarwal und Kumar 2021, S. 9.

Ein System kann auch nur rein verteilt sein, wenn es keine Kontrollkomponente oder gemeinsamen Zustand zur Erfüllung einer Aufgabe benötigt. Skalierte, zustandslose Funktionen sind ein Beispiel hierfür.

2.3.3. Abgeleitete Eigenschaften dezentralisierter Anwendungen

Dezentrale Systeme zeichnen sich dadurch aus, dass keine einzelne Kontrollinstanz existiert. Stattdessen ist die Kontrolle im Netzwerk über eine Mehrzahl an gleichberechtigten Knoten verteilt. Weiterhin können Knoten in einem offenen System durch verschiedenste, unabhängige Teilnehmer betrieben werden, welche nicht nur ihre eigenen Interessen verfolgen, sondern auch eigene Implementationen der Systemsoftware verwenden können, wodurch die *Integrationsmöglichkeiten* des Systems steigen. Dies wird als *permissionless* bezeichnet, wenn beliebige Teilnehmer dem System beitreten bzw. es nutzen können. Es steht im Gegensatz zu *permissioned*, wenn nur gewisse Teilnehmer unter bestimmten Bedingungen zugelassen sind[36].

Generell dürfen sich Knoten im System nicht gegenseitig vertrauen. Stattdessen müssen sie jede Kommunikation untereinander überprüfen. Erst wenn genügend Konsens unter den Knoten besteht oder die Wahrheit einer Aussage bewiesen werden konnte, darf dies als Fakt akzeptiert werden. Diese Eigenschaft wird als *trustless* bezeichnet, wenn ein Faktum nicht durch eine Autorität vorgeschrieben wird[37]. Innerhalb eines geschlossenen, *permissioned* Systems kann dieser Umstand aufgeweicht sein.

Um die trustless-Eigenschaft herstellen zu können, ist es notwendig, dass eine Information, welche als Faktum akzeptiert wird, *verifizierbar* ist. Die bedeutet, dass sie validiert, sie als authentisch nachgewiesen oder nachvollzogen werden kann. Handelt es sich um ein System, das sich auf einen Gesamtzustand einigen muss, kann *Immutability* die trustless-Eigenschaft unterstützen. Wenn Zustandsänderungen immutable bzw. *append-only* sind, kann der aktuelle Zustand des Systems stets verifiziert werden[38].

Ist ein System genügend dezentral aufgestellt, kann es ebenfalls *manipulations- und zensurresistent* werden[39]. Wenn beispielsweise eine große Anzahl an Nodes existiert und der Zustand des Systems append-only geführt wird, lassen sich Manipulationen nur unter erschwerten Bedingungen durchführen, da der Zustand durch eine Vielzahl

[36] Vgl. Helliar et al. 2020, S. 3-4.
[37] Vgl. De Filippi et al. 2020, S. 12-15.
[38] Vgl. Puthal et al. 2018, S. 18-19.
[39] Vgl. Eckert 2018, S. 840.

an Nodes stets geprüft werden kann. Gleichzeitig kann das System auch hochverfügbar werden, da ein simultaner Ausfall aller unabhängigen Knoten unwahrscheinlicher wird.

3. Software Repository Analyse

3.1. Grundbegriffe

Grundlegend ist ein Software Repository ein digitales System, in dem Entwickler ihre programmierte Software speichern und der Öffentlichkeit oder einer definierten Gruppe zur Verfügung stellen können. Sie unterstützen bei der sicheren Verteilung von Bibliotheken und ganzen Applikationen und bilden oftmals eine essentielle Basis in Software-Ökosystemen und der Software Supply Chain. Auf dieser Grundlage können Entwickler mithilfe von Build- und Dependency-Tools standardisiert und effizienter Software schreiben, indem sie bereits publizierte Programmbibliotheken wiederverwenden[40,41].

In der Praxis haben sich eine Reihe umfangreicher Repository-Systeme innerhalb verschiedener Ökosysteme etabliert. So ist in größeren Software Communities wie den JVM-, Rust- und Python-Ökosystemen zumindest einer dieser Dienste anzutreffen[42].

Im Rahmen dieser Arbeit werden vornehmlich öffentliche Repositories betrachtet, die Open-Source-Software kostenfrei bereitstellen. Ausschließlich geschlossene, kommerzielle Systeme werden nicht diskutiert.

Abbildung 2 skizziert den generell Fluss von Artefakten in und aus einem Software Repository. Ein Entwickler oder auch *Autor* einer Software erzeugt eine neue Version dieser, in der Abbildung Version 1.1.0. Diese möchte er dem Software Repository hinzufügen, sodass andere Entwickler, *Konsumenten*, diese verwenden können. Dazu muss er jene in eine vom Repository definierte Form bringen und mit entsprechenden Metadaten anreichern. Bei diesen Daten kann es sich um Verweise auf Abhängigkeiten, nämlich andere Softwarepakete im Repository, Verifikationsinformationen oder auch Dokumentation handeln[43].

[40] Vgl. Habermann und Leymann 2020, S. 11-12, S. 15.
[41] Vgl. Ma 2018, S. 458-459.
[42] Vgl. D'mello und Gonzalez-Velez 2019, S. 132-133.
[43] Vgl. Catuogno et al. 2020, S. 379-380.

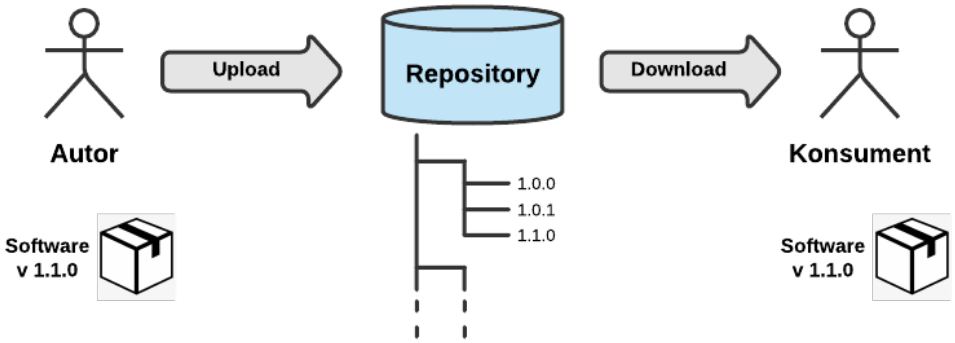


Abbildung 2: Flow von Artefakten in einem Software Repository

Im Repository hatte er bereits Vorgängerversionen seiner Software unter einem Package veröffentlicht. Durch das Hochladen seiner Artefakte erzeugt er ein neues Release seines Packages. Dieses wird daraufhin für Konsumenten im Repository sichtbar. Sie können nun die neue Version des Packages herunterladen und in ihren eigenen Anwendungen nutzen.

Die Implementationen von Software Repositories können sich in der Praxis unterscheiden, sodass sich auch entsprechend insbesondere der Upload-Prozess des Workflows unterscheiden kann. Der generelle Austauschgedanke bleibt jedoch erhalten.

3.2. Analyse existierender Software Repositories

Um ein neues dezentrales Software Repository konzipieren zu können, wurden aktuelle Implementationen gesichtet und untersucht. Ziel war es, die eingesetzten Technologien, Prozesse und Funktionsweisen zu analysieren und zu vergleichen. Auf dieser Basis können Features identifiziert werden, die im Anschluss im dezentralen Repository berücksichtigt werden müssen.

Hauptaugenmerk lag auf öffentlichen Software Repositories aus verschiedenen, großen Communities und Bereichen der Softwareindustrie. Dies umfängt die Softwareentwicklung mit unterschiedlichen Programmiersprachen, Betriebssysteme und Anwendungen. Weiterhin wurden auch Content-verteilende Systeme betrachtet, die sich nicht nur auf Software spezialisieren, um ebenfalls dezentral organisierte Konstrukte einbeziehen zu können. Folgende Repositories und Systeme wurden untersucht:

Maven Central Maven Central ist das Standard-Repository des JVM-Ökosystems und findet dadurch in Communities verschiedener Sprachen Anwendung.

NPM Registry Die NPM Registry ist nicht nur das größte Repository für JavaScript Software, sondern beherbergt unter anderem auch CSS- und Asset-Bibliotheken sowie Softwarebibliotheken anderer Programmiersprachen.

PyPI Der Python Package Index ist das zentrale Standard-Repository der Python Community, die in den letzten Jahren wieder deutlich populärer geworden war[44]. Es erlaubt die Veröffentlichung von Python Bibliotheken in Quellcodeformat sowie in plattformspezifischen, teilvorkompilierten Wheel-Paketen, welche den Entwicklungsprozess vereinfachen[45].

Docker Hub Docker Hub ist das öffentliche, de facto Standard-Repository für Container Images. Diese Images werden nicht nur für den Softwarebetrieb, sondern auch in der Entwicklung eingesetzt. Dadurch sind viele Anwendungen und Werkzeuge als Images verfügbar und finden an fast allen Punkten der Software Supply Chain Anwendung. Docker Hub erlaubt neben dem Hosting von Docker Images auch die Verwendung von OCI Images und ist damit auch mit Werkzeugen außerhalb des Docker eigenen Ökosystems kompatibel[46,47].

Debian Repository Debian ist eine der am häufigsten eingesetzten Linux Distributionen und bildet gleichzeitig den Grundstein für weitere beliebte Distributionen wie Ubuntu und Pop!_OS. Das dazugehörige Repository wird verwendet, um Software Updates und generell Softwarepakete an die Nutzer zu verteilen. Es wird durch das Team ein zentrales Repository gepflegt und betrieben, das als Master für eine Vielzahl an Mirrors dient, die durch verschiedene Organisationen kostenfrei angeboten werden. Diese sind über die gesamte Welt verteilt und sorgen dadurch für Ausfallsicherheit und optimierte Zugriffszeiten[48].

Arch User Repository (AUR) Arch Linux ist eine Linux Distribution, die versucht, die aktuellsten Software-Versionen zu nutzen. Das AUR ist eines der Repositorys, das durch das Betriebssystem verwendet wird. Es wird durch die Community gepflegt und ergänzt die offiziellen Repositories um weitere Software, die

[44] Vgl. TIOBE 2022.
[45] Vgl. Holth 2012.
[46] Vgl. Docker 2022a.
[47] Vgl. Estes und Brown 2018.
[48] Vgl. Debian 2022a.

nicht durch das Kern-Team verantwortet wird. Im AUR werden zentral Git Repositories bereitgestellt, die Build-Anweisungen enthalten, um die gewünschte Software beziehen zu können[49].

GitHub GitHub ist das de facto Zentrum der Open-Source-Softwareentwicklung. Eine Vielzahl an Open-Source-Projekten wird auf GitHub entwickelt oder bieten zumindest einen Mirror ihres VCS dort an. Gleichzeitig kann es auch als Quelle für Software Dependencies verwendet werden. Es handelt sich um eine zentrale Plattform, die das Hosting von Git Repositories sowie dazugehörige Entwicklungswerkzeuge anbietet[50]. Dazu gehören ein eigenes Continuous Integration System, Projektmanagement-Tools sowie diverse Paket-Hosting-Lösungen, die als alternative Software Repositories zu Maven Central, NPM, Docker Hub, usw. dienen können. Diese sind allerdings nicht Fokus der Untersuchung.

ethPM Das Ethereum Package Management ist ein dezentrales Software Repository für EVM Smart Contracts und ermöglicht nicht nur die Nutzung von Quellcode, sondern auch deployed Contract-Instanzen als Dependency[51,52]. Eine Vielzahl an ERC1319[53]-kompatiblen Registries, vornehmlich auf dem Ethereum Mainnet, können den Index eines Repositorys bilden. Sie verweisen jeweils auf Metadaten sowie Softwareartefakte in IPFS[54].

WAREZ Scene In der WAREZ Scene werden Software und andere Inhalte geteilt. Da diese teilweise urheberrechtlich geschützt sind, werden Mittel und Wege genutzt, privat und anonym operieren zu können[55]. Dezentralisierung ist ein Teil hiervon. Da es sich um eine weitreichende Community handelt, wurden drei der größten Verteilungsmethoden näher untersucht. Dazu gehören die Distribution über private Sites bzw. File Dumps, die Nutzung des Peer-to-Peer-Protokolls BitTorrent und die Veröffentlichung im Usenet[56,57].

Die genannten Repositories wurden analysiert und unter bestimmten Gesichtspunkten miteinander verglichen. Ziel war es, Gemeinsamkeiten und Unterschiede zu identifizieren sowie diese unter dem Blickwinkel eines dezentralen Systems zu untersuchen.

[49] Vgl. Arch Linux 2022a.
[50] Vgl. Tsitoara 2019, S. 9-10, S. 95-104.
[51] Vgl. ethPM 2021a.
[52] Vgl. Merriam et al. 2018.
[53] ERC1319 definiert eine Standardschnittstelle für Smart Contract Package Registries.
[54] Vgl. ethPM 2021b.
[55] Vgl. Goldman 2003, S.1-4.
[56] Vgl. Huizing und van der Wal 2014, S. 1.
[57] Vgl. BitTorrent.org 2015.

Die Strukturierung der Inhalte der Repositories ist ein grundlegender Baustein. Diese können sich mitunter unterscheiden. Es muss ein gemeinsamer Nenner identifiziert werden, der in einem neuen, universellen Repository durch Nutzer erwartet werden würde. Gleichzeitig müssen entscheidende Unterschiede erkannt werden, sodass sie mit entsprechenden Lösungen adressiert werden können. Metainformationen sollten gleichermaßen betrachtet werden, um diese ggf. effizienter im Repository einsetzen zu können.

Einen weiteren, wichtigen Bestandteil bilden die Sicherheit und Integrität der Inhalte eines Software Repositories. Es muss sichergestellt werden, dass Software nicht verfälscht oder korrumpiert werden kann. Hierfür stehen verschiedene Methoden bereit, die Vor- und Nachteile besitzen sowie unterschiedliche Grade an Konfidenz bieten.

Im Sinne der Sicherheit sind ebenso Zugriffskontrollen von Bedeutung. Es muss gewährleistet sein, dass Sicherheit und Integrität dadurch nicht beeinflusst werden und genügend Flexibilität zum Management der Inhalte besteht.

In diesem Zuge sind auch Verfügbarkeit und Ausfallsicherheit zu betrachten, da ein universelles Repository an dieser Stelle erhöhte Anforderungen aufweist, wenn es ökosystemübergreifend eingesetzt werden würde.

Schließlich bilden auch Richtlinien zu den Repository-Inhalten einen Untersuchungsaspekt. Diese bestimmen, welche Inhalte erlaubt sind und wie auf unerwünschte reagiert wird. Entsprechende Prozesse könnten ebenfalls für das Konzept von Bedeutung sein.

Der Fokus des Vergleichs liegt auf Software Repositories. Die Veröffentlichung in der Filesharing Szene unterliegt teilweise anderen Regeln, die in der Analyse entsprechend weniger Beachtung finden werden.

3.3. Analyseerkenntnisse

3.3.1. Gemeinsamkeiten

Durch die Untersuchung der einzelnen Repositories und der anhängenden Prozesse konnten insbesondere zwischen den Software Repositories eine Vielzahl an Gemeinsamkeiten erkannt werden. Diese Erkenntnisse sind für das Konzept und etwaige die Kompatibilität eines neuen dezentralen Repositorys äußerst bedeutend.

Eine umfangreiche Übersichtstabelle, die im Laufe der Untersuchung erstellt wurde, ist in Anhang A zu finden. Sie diente als Quelle für den Vergleich von Merkmalen.

Komfortabler Zugriff

Software Repositories, wie Maven Central und die NPM Registry, stellen jeweils das Standard Repository für ihr jeweiliges Ökosystem dar. Dependency Tools und Ähnliches können diese ohne zusätzliche Konfiguration direkt nutzen, um fast jedes gewünschte Softwarepaket zu beziehen. Dies vereinfacht das Dependency Management für Konsumenten und gibt den jeweiligen Repositories eine de facto Monopolstellung. Die Nutzung alternativer Repositories ist häufig mit erweiterter Konfiguration verbunden. Ihre Nutzung ist daher neben dem Mainstream-Repository eher eine Randerscheinung. Für Container Images gestaltet sich die Verwendung anderer Quellen durch das eingesetzte Adressierungsschema hingegen nahtlos.

Koordinaten von Paketen

In allen untersuchten Software Repositories hat sich eine ähnliche Art der Referenzierung von Versionen von Softwarepaketen etabliert. Es wird vornehmlich eine Adressierung aus *Groupname - Packagename - Version* verwendet, wobei Gruppierungen in einigen Systemen wie beispielsweise NPM optional sind. Ebenso können zum Beispiel in Maven Central weitere Elemente zur Adressierungen hinzugefügt wurden. Die Version eines Releases wird in SemVer oder kompatiblen Formaten ausgedrückt[58]. Hierdurch ist es möglich, repository- und damit ökosystemübergreifende Releases ohne Anpassungen und Mappings zu erzeugen.

Verschlüsselte Übertragung mittels HTTP

Die meisten der untersuchten Repositories setzen HTTPS als Standardübermittlungsprotokoll ein. Die Übertragung von Informationen ist daher verschlüsselt. Entsprechend sind die Repository-Inhalte für Anwender auf dieselbe Art und Weise erreichbar wie Webseiten im Internet. Weitere verwendete Kommunikationsprotokolle, wie SSH oder BitTorrent, sind aufgrund von Firewalls und anderen Netzwerkkonstrukten für Nutzer nicht immer verwendbar.

Open Source

Open-Source-Projekte können in allen untersuchten Repositories ihre Daten kostenlos publizieren. Dies fördert die Verbreitung und Nutzung von Open-Source-

[58] Vgl. Raemaekers et al. 2014, S. 3.

Software. Closed-Source-Bibliotheken und Anwendungen dürfen in einigen Repositories explizit nicht veröffentlicht werden, während andere entsprechende kommerzielle Dienste hierfür anbieten.

Verifikation

Es findet durch fast alle untersuchten Repositories bzw. die dahinter steckenden Systeme keine Verifikation auf Authentizität der publizierten Softwareartefakte statt. Stattdessen bieten sie die Möglichkeit für Konsumenten, die Artefakte anhand von Prüfsummen auf ihre Integrität zu kontrollieren. Hierzu werden größtenteils mehrere Standard-Hashfunktionen verwendet. Die Prüfsummen werden entweder durch die Autoren oder durch das Repository erzeugt.

Weiterhin erlauben die meisten der Repositories die Option, GPG-Signaturen zu den Artefakten zu veröffentlichen. Nur wenige erklären dies allerdings zu einer essentiellen Anforderung. Aufgrund der erhöhten Komplexität des GPG-Prozesses ist die Kontrolle der Signaturen oftmals nicht in den Standard-Workflow der entsprechenden Werkzeuge integriert.

Erweiterte Daten und Metadaten

Alle untersuchten Software Repositories speichern primär Build-Artefakte. Daneben werden teilweise auch der entsprechende Quellcode und Dokumentation angeboten. Weitere Metadaten sind im Regelfall in den Build-Artefakten selbst enthalten[59]. Neuere Repositories oder Anwendungen um die entsprechenden Repositories herum werten diese Daten aus und erzeugen beispielsweise Abhängigkeitsgraphen, die unter anderem für die Auswertung von Lizenzkonflikten genutzt werden. Diese Funktionen übersteigen die Kernkompetenzen von Software Repositories, sind jedoch in der Praxis von größter Wichtigkeit.

Trennung von Metadaten und Artefakten

Metadaten, die zur Verwaltung der Pakete benötigt werden, und die eigentlichen Build-Artefakte werden überwiegend voneinander getrennt gespeichert. Dies hat vorwiegend technische Gründe zur Optimierung. So können die Daten entsprechend effizienter gespeichert und an Konsumenten transferiert werden. Debian beispielsweise dedupliziert die eigentlichen Softwarepakete im Dateisystem durch Referenzierung, während andere Systeme schnellere Blockspeicher für Artefakte einsetzen.

[59] Vgl. The Apache Software Foundation 2021a.

Bedingt immutable und nicht zensurresistent

Den Repository-Betreibern und deren Communities ist die Bedeutung der Unveränderlichkeit der Artefakte bewusst, da dies weitreichende Probleme in der Supply Chain verursachen kann. Allerdings ist zumindest den Administratoren der untersuchten Software Repositories immer die Möglichkeit gegeben, unerwünschte Artefakte zu entfernen. Teilweise können auch die Besitzer der Packages selbst diese löschen oder nachträglich bearbeiten. Solche Aktionen stellen die Konsumenten der Artefakte vor unvorhergesehene Probleme und inkonsistentes Verhalten entlang der Supply Chain. Zudem sind die Repositories dadurch nicht zensurresistent, wenn Administratoren oder Autoren, ggf. unter Zwang, Änderungen durchführen können.

Permissioned und geringe Inhaltskontrolle

Die meisten der untersuchten Repositories sind auf ein schnelles Onboarding neuer Autoren ausgelegt. Häufig reicht die Erstellung eines neuen Benutzerkontos aus, um Schreibrechte zu erhalten. Nutzer können anschließend ihre Artefakte ohne weitere Reviewprozesse unverzüglich veröffentlichen. Die Verantwortung für die Inhalte wird in diesem Fall in Richtung der Autoren verschoben[60,61]. Spätestens im Falle von Beschwerden und nach der Erkennung von Sicherheitsproblemen in Bezug auf die Inhalte greifen Administratoren ein.

Zentralisierte Verwaltung

Die meisten der analysierten Repositories werden federführend durch eine einzelne Organisation oder eine kleine Personengruppe betrieben und organisiert. Obwohl Autoren in der Praxis zunächst nicht daran gehindert werden, ihre Inhalte zu publizieren, besitzen die Betreiber generell die Möglichkeit, die Daten in den jeweiligen Systemen zu beeinflussen. Dies geschieht im Normalfall zur Beseitigung von Problemen, zur Entfernung von Malware oder bei Verstoß gegen die Nutzungsbedingungen[62].

Auch wenn die Repositories, wie das Debian Repository, organisatorisch verteilt aufgestellt sind, existiert dennoch ein Master, von dem Mirrors und andere Slaves ihre Daten erhalten[63].

[60] Vgl. npm Docs n. d.(f).
[61] Vgl. GitHub Docs 2021.
[62] Vgl. npm Docs n. d.(e).
[63] Vgl. Debian 2022a.

3.3.2. Unterschiede

Neben den Gemeinsamkeiten, die weitgehend alle Repositories teilen, existieren entscheidende Unterschiede in der Technik, Organisation und Prozessen, durch welche die einzelnen Systeme Schwerpunkte setzen. Entsprechend büßen andere Aspekte hierdurch an Fokussierung ein.

Interaktionsschnittstelle

In Bezug auf die Interaktionsschnittstelle lassen sich die untersuchten Software Repositories in zwei Gruppen unterteilen. Eine Gruppe nutzen APIs und Web Interfaces, um ihre Inhalte zu repräsentieren. Dies versteckt die tatsächliche Funktionsweise des Systems und Struktur der Daten. Die Konsumenten sind auf die korrekte Funktion der teilweise proprietären Systeme angewiesen.

Die zweite Gruppierung verwendet File System-orientierte Strukturen. Dadurch ist es möglich, alternative oder lokale Repositories sowie Mirrors mit geringem Aufwand zu erzeugen.

Separierte Adressierung

Dezentral organisierte Repository Systeme arbeiten hingegen mit separierten Indizes, die entweder auf die Lokation der gewünschten Daten verweisen oder Content Addressing nutzen. Diese logische und auch rechtliche Trennung ist explizit beabsichtigt. Die Daten können auch ohne den Index weiterexistieren oder durch einen anderen Index nahtlos übernommen werden. Dadurch ist das Repository bzw. der verwaltende Index austauschbar, ohne dass die Artefakte erneut publiziert werden müssen. Die Nutzdaten sind nicht vom Index abhängig.

Verantwortung für Inhalte

Insbesondere die Teams hinter dem Debian und Arch-Core Repository übernehmen Verantwortung für die veröffentlichten Inhalte. Sie durchlaufen einen Reviewprozess durch eine kleine Menge vertrauenswürdiger Personen, bevor eine Publikation möglich ist. Hier findet zumindest theoretisch eine strikte Verwaltung und Qualitätskontrolle statt.

Andere Repositories weisen diese Verantwortung weitgehend von sich, indem sie keine solchen Prozesse verfolgen und die Publikationsautorität den Autoren überlassen. Diese können neue Releases in kürzester Zeit veröffentlichen. Nur bei Bekanntwerden

von Verstößen gegen die Nutzungsbedingungen oder beim Auffinden von Schadsoftware werden die Repository-Betreiber tätig.

Prüfsummen und Signaturen

Der Umgang mit Prüfsummen und Signaturen der Artefakte als Beweis für deren Echtheit wird je nach Repository unterschiedlich gehandhabt. Teilweise werden diese durch das Repository selbst erzeugt, was dieses in eine zentrale Vertrauensposition befördert. Andernfalls werden Prüfsummen und Signaturen durch die Autoren erzeugt und importiert. Deren Authentizität wird aber durch die Repositories nicht nachgewiesen. Es liegt in diesem Fall im Aufgabenbereich der Konsumenten, die Verknüpfung von Build-Artefakten, Autor und Quellcode selbst, meist manuell, zu kontrollieren. Dies beinhaltet den Overhead, die passenden öffentlichen Schlüssel der Autoren sicher zu beziehen, da diese in der Regel nicht von den Repositories angeboten werden.

Docker Hub versucht, diesen Prozess der Schlüsselfindung und der Signaturprüfung mittels des Docker Content Trust Konzepts zu automatisieren. Hierbei wird ein Schlüsselserver neben der Container Registry angeboten, der alle erforderlichen öffentlichen Schlüssel enthalten soll. Die Prüfung der Signaturen ist in den Docker Client eingebettet. Allerdings ist auch dieses Verfahren aktuell nur ein optionales Feature[64].

Im Falle von Debian werden die Artefakte und Signaturen durch Debian vertraute Personen erzeugt, was die Echtheit im Rahmen des Debian-Ökosystems beweist. Zudem werden Prüfsummen und Signaturen bei jeder Änderung über den gesamten Repository Inhalt erzeugt, was dessen Integrität als Ganzes sichert[65].

Kostenpflichtige Erweiterungen

Software Repositories, wie die NPM Registry oder Docker Hub, enthalten kommerzielle Produktbestandteile, die erweiterte Features anbieten. Diese haben teilweise auch Auswirkung auf die Persistenz der veröffentlichten Artefakte. Im Falle von Inaktivität könnten beispielsweise Releases gelöscht oder Zugriffe von nicht-zahlenden Konsumenten durch ein Rate Limit begrenzt werden[66].

[64] Vgl. Docker 2021.
[65] Vgl. Debian 2021.
[66] Vgl. npm Docs n. d.(g).

Weiterhin erlauben solche Features meist geschlossene Bereiche, die das Hosting von Closed-Source-Software ermöglichen, sowie umfangreichere Security Features und die erweiterte Auswertung von Metadaten über Dependency Trees.

Zensurresistenz und Ausfallsicherheit

Von den untersuchten Systemen haben sich nur die Nutzung von BitTorrent und das ethPM Konzept unter Verwendung von Blockchains und IPFS als technisch zensurresistent gezeigt. Die zentralisierten Repositories werden zentral administriert, wobei Inhaltskontrollen oft vernachlässigt werden. Dennoch liegt es in der Macht der Betreiber, Inhalte zu entfernen oder gar zu manipulieren. Sie sind ebenso an lokale Gesetze gebunden und müssen diese einhalten, auch wenn diese keine globale Gültigkeit besitzen. BitTorrent und IPFS sind gegen solche Eingriffe resistent, da einerseits die Daten über eine Vielzahl an Klienten redundant verteilt und andererseits die Inhalte eindeutig, immutable adressiert sind. Dadurch können sie nicht durch eine zentrale Instanz auf technische Art und Weise aus den Netzen entfernt werden, ohne das gesamte System lahmzulegen. Auch die Durchsetzung von Gesetzen ist so nicht ohne Kollateralschäden realisierbar. Nicht-technische Wege sind allerdings durch die grundlegende Transparenz von Peer-to-Peer-Methoden weiterhin möglich[67].

Die genannte Verteilung sorgt weiterhin dafür, dass die Daten theoretisch hochverfügbar bereitstehen, solange ausreichend Interesse an den Inhalten besteht. Zentralisierte Repositories nutzen teilweise CDNs, um Verteilung und Hochverfügbarkeit zu erreichen. Dennoch handelt sich hier um zentralisierte Systeme, die durch Gesamtausfälle betroffen sein können. Debians Netzwerk von Repository Mirrors kann dieses Problem umgehen, da sie durch eine Menge freiwilligen, unabhängiger Mirror-Betreiber weltweit bereitgestellt werden[68].

Inhaltsprüfung und Publikationsgeschwindigkeit

Repositories wie die NPM Registry oder Maven Central zeichnen sich unter anderem dadurch aus, dass ständig neue Bibliotheken oder neue Releases von Packages in ihrem Katalog hinzugefügt werden. Hier findet kein explizierter Reviewprozess statt, sodass neue Inhalte direkt durch die Autoren publiziert werden. Eine tiefgründige, manuelle Inhaltskontrolle, wie sie auch nur ansatzweise für Debian Pakete stattfindet, würde diesen Prozess ausbremsen. Aufgrund der fehlenden Inhaltsprüfung müssen die Konsumenten die Verantwortung für die Nutzung von Softwarepaketen übernehmen.

[67] Vgl. Piatek et al. 2008.
[68] Vgl. Cappos et al. 2008, S. 566.

4. Konzeption des universellen, dezentralen Repositorys

4.1. Anforderungserfassung

4.1.1. Allgemeine Anforderungen aus existierenden Repositories

Mithilfe der Analyse der untersuchten Repositories (siehe Kapitel 3) lassen sich Mindestanforderungen für ein neues, dezentrales Software Repository ableiten. Hierbei steht die Verbesserung in Bezug auf die bisherigen Systeme sowie die generelle Kompatibilität im Mittelpunkt. Dies bedeutet, dass bestehende Konzepte und Standards, soweit sinnvoll, beibehalten werden sollten, damit Workflows und Werkzeuge nicht gravierend in ihrer Funktionsweise angepasst werden müssen. Ebenso würden weitreichende, logische Änderungen wahrscheinlich die Akzeptanz bei Autoren und Konsumenten verringern.

Ein neues Repository sollte die nachfolgenden Mindestanforderungen umsetzen. Die Beschriftung **A*** bezeichnet allgemeine Anforderungen.

A1 - Einfacher, zentraler Einstieg für Konsumenten

Beschreibung	Die Nutzung des neuen Repositories sollte über eine zentrale, Standard-Schnittstelle erfolgen, die keine oder nur wenig Konfiguration benötigt. Dies bedeutet, dass Konsumenten über eine Adresse auf alle Inhalte des Repositories Zugriff erhalten sollten. Dies steht beispielsweise im Gegensatz zu der Liste an Registries, die für die Nutzung von ethPM durch jeden Anwender selbst zusammengestellt werden muss[69]. Dies ist unabhängig von der Verwendung alternativer Repositories.

[69] Vgl. ethPM 2019.

A2 - Koordinaten	
Beschreibung	Die Adressierung von Softwarepaketen und Releases im Format *Groupname - Packagename - Version* sollte beibehalten werden, um aktuelle Strukturen übernehmen zu können. Die Nutzung von SemVer für die Versionsdefinition ist zu empfehlen, sollte jedoch nicht forciert werden. Die detaillierte Auswertung der Versionsnummer sollte in entsprechenden Dependency Tools geschehen.

A3 - Hosting einsatzfertiger Softwarepakete	
Beschreibung	Soweit es möglich und sinnvoll ist, sollte das Hosting von Build-Artefakten priorisiert werden, die in der Entwicklung ohne erneutes Kompilieren eingesetzt werden können. Andernfalls sollte auf den Quellcode zurückgefallen werden. Weiterhin muss beachtet werden, dass mehrere Varianten eines Artefakts bereitgestellt werden können, die für beispielsweise verschiedene Plattformen vorkompiliert wurden.

A4 - Schnelles Onboarding und Schreibrechte	
Beschreibung	Wie in den untersuchten Repositories verbreitet, sollte die Registrierung eines neuen Benutzerkontos ausreichend sein, um Schreibrechte für das System zu erhalten[70, 71]. Diese sollten auf eigene Gruppen und Packages sowie deren initiale Erzeugung begrenzt sein.

A5 - Leserechte	
Beschreibung	Konsumenten sollten Leserechte auf alle öffentlichen Inhalte des Repositories besitzen. Das gesamte Repository ist öffentlich. Die Erstellung eines Benutzerkontos ist hierfür nicht notwendig.

A6 - Alternative Repositories	
Beschreibung	Die Existenz von alternativen Repositories sollte nicht verhindert werden.

A7 - Öffentliche Strukturen	
Beschreibung	Ein Mindestmaß an Navigierbarkeit des Repositories muss entlang des Koordinatensystems gegeben sein. Dies bedeutet, dass Packages innerhalb von Gruppen sowie Versionen innerhalb von Packages auffindbar sein müssen.

A8 - Validation von Artefakten	
Beschreibung	Die Integrität von Artefakten sollte über die Verwendung von Prüfsummen und Signaturen geprüft werden können. Dabei sollen jeweils mehrere Prüfsummen über verschiedene, starke Hash-Funktionen erzeugt werden. Die Nutzung von Signaturen, die beispielsweise über GPG erzeugt werden, ist optional, wird jedoch empfohlen. Falls solche Signaturen verwendet werden, müssen Verweise auf Schlüssel in Form von Fingerprints oder Links definiert werden.

[71] Vgl. npm Docs n. d.(d).
[72] Vgl. Arch Linux 2022b.

A9 - Historisierung

Beschreibung	Die Veränderungen an den Inhalten des Repositories müssen dokumentiert werden. Gruppen, Packages und Versionen dürfen nicht spurlos verschwinden. Es muss zumindest ein Verweis auf ihre Löschung existieren. Entitäten, die in der Vergangenheit bereits existierten oder derzeit bestehen, dürfen nicht mit alternativen Inhalten überschrieben oder wiederhergestellt werden.

A10 - Kompatibilität mit Web-Infrastruktur

Beschreibung	Das Repository muss weitgehend Kommunikationsprotokolle verwenden, welche die allgemeine Erreichbarkeit des Systems ermöglichen. Ein Einsatz aus Unternehmensnetzen und ähnlichen abgeschotteten Netzwerken muss berücksichtigt werden. Eine verschlüsselte Übertragung wird vorausgesetzt.

A11 - Management existierender Inhalte

Beschreibung	Für den Fall, dass Packages oder Releases gegen die Nutzungsbedingungen des Repositorys verstoßen oder Schadsoftware in diesen bekannt werden, müssen diese Inhalte aus dem System entfernt werden. Dies muss zum Schutz der Konsumenten und Repository-Betreiber geschehen.

Die folgenden Punkte sollten beim Konzept eines neuen Repositorys berücksichtigt werden, sind allerdings keine Muss-Anforderungen. Sie wurden ebenfalls aus der Analyse abgeleitet:

A12 - Geschlossene Bereiche

Beschreibung	Die Nutzung geschlossener Bereiche ist insbesondere für die Verbreitung von proprietärer Software essentiell, sodass unbefugte Dritte keinen Zugriff auf diese Bibliotheken und Applikationen erhalten. Dies erfordert auch die Erstellung und Umsetzung eines Rechte- und Rollenkonzepts innerhalb dieser Bereiche.

A13 - Quellcode und Dokumentation

Beschreibung	Neben dem Hosting von Build-Artefakten sollten auch der entsprechende Quellcode und etwaige Dokumentation publiziert werden können. Dies gibt Konsumenten die Möglichkeit, die Software selbst kompilieren zu können und Dokumentation sowie Quellcode in der identischen Version für tiefgreifende Studien zur Verfügung zu haben.

4.1.2. Spezielle Anforderungen aus Dezentralisierungseigenschaften

Zusätzlich zu den Anforderungen, die sich aus der Analyse zentralisierter Repositories (Kapitel 3.3) ergeben, müssen die Eigenschaften eines dezentralen Systems eingehalten werden. Entsprechend sollte das System trustless, permissionless und immutable bzw. append-only sein. Aus diesen Merkmalen ergeben sich weitere, detaillierte Anforderungen, die umzusetzen sind.

Die Akteure innerhalb eines *trustless* Repository-Systems müssen sich nicht gegenseitig vertrauen, um dieses sicher nutzen zu können. Dementsprechend darf das Software Repository nicht nur durch eine einzige Entität, sei es eine Organisation oder einen kleinen Personenkreis, betrieben werden. Diese könnte zusätzlich besondere Rechte innerhalb des Systems besitzen. Eine solche Machtkonzentration muss vermieden werden. Daher sollte das Repository ausreichend dezentral durch eine Vielzahl von unabhängigen Parteien verteilt betrieben werden.

Konsumenten und Autoren können aufgrund des Systemdesigns sichergehen, dass eine Manipulation seitens des Repositorys selbst ausgeschlossen ist. Dementsprechend muss dieses Design, ausgedrückt in Quelltext, Protokollen und Prozessen, einsehbar und damit verifizierbar sein. Zeitgleich muss ausgeschlossen werden, dass die Logik des Repositorys selbst zur Laufzeit, ggf. ohne ausreichende Vorwarnungen, geändert werden kann. Im besten Fall ist der laufende Programmcode *immutable*.

Weiterhin darf keine Vertrauensbeziehung zwischen Konsument und Autor bestehen müssen. Dies bedeutet, dass ebenfalls Build-Artefakte inhaltlich *verifizierbar* sein sollten, um so ihre Authentizität und ihre Unverfälschtheit beweisen zu können. Ein Bruch der Vertrauenskette, um dieses zu erreichen, muss verhindert werden. Entsprechend muss der Quellcode eines Artefakts ebenfalls publiziert und sämtliche schreibenden Interaktionen des Autors mit dem Repository aufgezeichnet werden. Auf diese Weise kann ebenfalls festgestellt werden, ob alle Informationen unverfälscht in das Repository übertragen wurden[73].

Generell muss jede Interaktion mit dem Software Repository *permissionless* stattfinden. Lese- und Schreiboperationen dürfen nicht seitens des Systems verweigert werden können. Neue Packages oder Releases bedürfen keinem vorherigen Review oder sonstiger Erlaubnis, bevor diese erstellt werden können. Ein Autor kann innerhalb seines Bereiches beliebig neue Elemente erzeugen. Dabei dürfen keine Konflikte mit Daten von Autoren in anderen Bereichen entstehen. Gleichzeitig kann ein Konsument alle Informationen des Repositorys auslesen, ohne aufgrund mangelnder Berechtigungen blockiert zu werden.

Auf Seiten des Autors darf nur er seine hinzugefügten Inhalte beeinflussen, sodass diese zensurresistent im Repository abgelegt sind. Das Repository oder seine Betreiber dürfen technisch nicht in der Lage sein, jene Inhalte beim Schreiben oder Lesen zu manipulieren oder zu blockieren. Dasselbe gilt für dritte Parteien.

[73] Vgl. Reproducible Builds n. d.

Weiterhin sind die hinzugefügten Daten des Autors immutable und können nur erweitert werden, *append-only*. Inhalte wie Packages und Releases können nach der Erzeugung nicht mehr gelöscht oder bearbeitet werden. Dies gilt sowohl für den Autor als auch für das Repository selbst.

Das Repository und seine Inhalte müssen stets verfügbar sein. Es darf kein Single Point of Failure existieren, der das System zum Ausfall bringt oder dessen Funktion unterbricht.

Schließlich muss das Repository offen gestaltet sein, sodass es mit bestehender und zukünftiger Infrastruktur integriert werden kann. Dies beinhaltet, dass die Kernlogik des Systems von außen erweitert werden können sollte.

Die Anforderungen aus dezentraler Sicht sind nachfolgend nochmals kurz aufgelistet. Die Beschriftung **D*** bezeichnet dezentrale Anforderungen.

D1 - Trustless	
Beschreibung	Das System muss durch eine Vielzahl unabhängiger Parteien betrieben werden. Die Repository Logik ist immutable.

D2 - Permissionless	
Beschreibung	Alle Nutzer besitzen allgemeine Leserechte auf alle Inhalte. Autoren können Bereiche erzeugen und innerhalb dieser ungehindert schreiben.

D3 - Immutable	
Beschreibung	Das Repository und seine Inhalte sind unveränderbar und können nur erweitert werden.

D4 - Verfügbarkeit	
Beschreibung	Das verteilte System darf keinen Single Point of Failure besitzen, welcher einen kompletten Systemausfall verursachen könnte.

D5 - Zensurresistent	
Beschreibung	Nur der Autor darf seine Inhalte beeinflussen. Das Lesen von Daten kann nicht blockiert oder manipuliert werden.

D6 - Authentizität und Verifizierbarkeit	
Beschreibung	Das Repository, seine Interaktionen und Inhalte müssen nachvollziehbar und überprüfbar sein.

D7 - Integration	
Beschreibung	Das Repository muss mit bestehender Infrastruktur zusammenarbeiten können.

4.2. Datensicherheitsmechanismen

4.2.1. Verifikation

Verifikation ist ein wichtiger Bestandteil Software Supply Chain. In Bezug auf Software Repositories dreht es sich hier vornehmlich um die Frage:

> *Ist das Artefakt, das aus einem Repository bezogen wird, tatsächlich das Artefakt, das erwartet wird?*

Diese Fragestellung muss entlang der Supply Chain mehrfach betrachtet werden: im Repository, in den Prozessen und den verwendeten Werkzeugen. Um die einzelnen Punkte jeweils überprüfen zu können, müssen sie offengelegt werden. Dies ist beispielsweise durch die ausschließliche Nutzung von Open-Source-Software, die Dokumentation und Aufzeichnung der Prozesse und die unveränderliche Publikation aller entstehenden Daten realisierbar. Anhand dieser ist es theoretisch jeder teilnehmenden Partei möglich, die Historie eines Artefakts nachzuverfolgen und die o.g. Frage zu beantworten.

Bei den untersuchten, zentralisierten Software Repositories liegen diese notwendigen Daten nicht in Gänze vor, da sie nicht für deren Publikation entsprechend designt wurden und es sich teilweise um kommerzielle Produkte handelt.

Stattdessen kommen in allen untersuchten Repositories Prüfsummen und Signaturen auf die ein oder andere Weise zur Anwendung. Dies gibt den Konsumenten die Möglichkeit, eine ansatzweise Verifikation eines Artefakts durchführen zu können.

4.2.2. Prüfsummen

Von den Artefaktdateien werden jeweils Prüfsummen erzeugt und können neben den eigentlichen Artefakten durch die Konsumenten heruntergeladen werden. Es handelt sich hierbei um einen Datensatz, der mittels einer Hash-Funktion vom Inhalt einer Datei generiert wird. Nutzer können dieselbe Funktion auf heruntergeladene Artefakte anwenden und das Ergebnis mit dem Ergebnis, welches durch das Repository bereitgestellt wurde, vergleichen. Stimmen diese überein, gilt die Datei als valide[74].

Die Frage an dieser Stelle ist, welche Bedeutung diese Übereinstimmung besitzt. Im Endeffekt wurde bewiesen, dass der Download des Artefakts erfolgreich war und eine

[74] Vgl. Jockisch 2021.

inhaltsgleiche Datei vom Repository transferiert wurde. Dies ist keine Verifizierung dafür, dass es sich um das erwartete Artefakt handelt oder es die gleiche Datei ist, die durch den Autor erzeugt wurde. Es ist zunächst nicht eindeutig, durch wen die Prüfsumme berechnet wurde. Sie könnte durch das Repository erstellt worden sein und dient dann lediglich zur Download-Verifikation. Stammt das Hash-Ergebnis hingegen vom Autor, kann diese verwendet werden, um zu beweisen, dass die Inhalte vom Autor stammen.

Allerdings bestehen zu diesem Sachverhalt weiterhin Zweifel an der Vertrauenswürdigkeit des Software Repositorys. An diesem Punkt könnte es den Inhalt und die Prüfsumme manipuliert und ersetzt haben. Es liegt hier im Aufgabenbereich des Konsumenten, zu verifizieren, dass dies nicht der Fall ist. Eine Möglichkeit wäre es, die Prüfsumme aus einer anderen, unabhängigen Quelle zusätzlich zu beziehen und abzugleichen.

Die bessere Alternative ist hingegen die Pflege eines unveränderlichen Transparency Logs, in dem alle Prozessschritte der Publikation eines Artefakts festgehalten werden. Sofern der Logging-Prozess ebenfalls transparent stattfindet, kann dieses verwendet werden, um Artefakte entlang der Up- und Download-Kette zu verifizieren.

Eine Prüfung der Artefakte über die gegebenen Prüfsummen hinaus findet in vielen untersuchten Werkzeugen wie `npm` nicht statt.

4.2.3. Signaturen

In den meisten zentralisierten Software Repositories, beispielsweise Maven Central oder PyPI, ist es möglich, Signaturen zu Artefakten abzulegen. GPG ist in den untersuchten Repositories die Standardtechnik. Durch das Public-Key-Verfahren signiert der Autor die Artefakte mit seinem privaten Schlüssel und veröffentlicht eine Signaturdatei neben dem eigentlichen Artefakt, ebenso wie es mit Prüfsummen geschieht. Zur Verifikation benötigt der Konsument den öffentlichen Schlüssel des Autors. Hierüber wird bewiesen, dass eine Datei tatsächlich in dem vom Autor gewollten Zustand erhalten wurde[75]. Eine Signatur erweitert in diesem Sinne die Funktion einer Prüfsumme um die eindeutige Beziehung zum Autor. Eine Manipulation durch das Repository ist damit weitgehend ausgeschlossen.

[75] Vgl. Eckert 2018, S. 381-384.

Bei vielen untersuchten Repositories wird dieser Prozess allerdings nicht automatisiert mit den Standardwerkzeugen genutzt, da die notwendigen öffentlichen Schlüssel nicht im Repository enthalten sind und auch nicht auf diese verwiesen wird. Daher liegt es beim Konsumenten, die entsprechenden Schlüssel von den einzelnen Projekten und Autoren einzusammeln. Daraus ergibt sich wiederum das Risiko, dass ein falscher oder kompromittierter Schlüssel verwendet wird.

In der Praxis werden öffentliche GPG-Schlüssel hauptsächlich über drei Verfahren bezogen:

- zentrale, öffentliche Schlüsselserver, in denen jeder Public Keys veröffentlichen kann,
- Peer-to-Peer-Verfahren, bei denen einzelne Nutzer Schlüssel direkt miteinander austauschen und damit ein Vertrauensnetzwerk aufbauen, und
- als Web Ressource, indem die Schlüssel beispielsweise auf der jeweiligen Projekt-Webseite veröffentlicht werden.

Der Einsatz der genannten Verfahren ist bezüglich ihrer Vertrauenswürdigkeit abzuwägen. Der Austausch von Schlüsseln zwischen vertrauten Nutzern ist dabei am sichersten einzuschätzen, während öffentliche Schlüsselserver unkontrollierte Daten verteilen. Services wie keybase.io versuchen durch die Verknüpfung mit anderen Online-Identitäten den Vertrauensstatus zu erhöhen. Allerdings handelt es sich hierbei wiederum einen dritten, zentralisierten Dienst. Andere Bestrebungen versuchen von Zertifikatsketten, wie es bei der Verschlüsselung von Webseiten der Fall ist, Gebrauch zu machen, um Schlüssel mit Vertrauen zu versehen[76,77].

Am Ende muss der Konsument jedoch selbst entscheiden, wann er einen Schlüssel als vertrauenswürdig einschätzt. Dabei muss ebenfalls die Frage beantwortet werden, ob die signierende Identität auch tatsächlich der Autor der gewollten Software ist oder wie diese zusammenhängen.

Auch durch die Ausweitung der Verifikation in Richtung der Identität des Autors und dessen Verknüpfung mit dem Artefakt kann dennoch nicht eindeutig verifizieren, dass aus dem Repository tatsächlich die erstrebte Software geladen wurde. Der Deployer eines Artefakts könnte beispielsweise vor dem Build den Quellcode manipulieren und dieses ins Repository importieren. Für die Konsumenten wäre dies am Artefakt und der Signatur nicht ersichtlich.

[76] Vgl. Keybase Book n. d.
[77] Vgl. Ramsdell 2004, S. 4.

4.2.4. Reproducible Builds

Um das Artefakt eindeutig mit dem Quellcode zu verknüpfen, können Reproducible Builds eingesetzt werden. Dies bedeutet, dass in einer definierten Umgebung aus dem gleichen Quellcode immer wieder das binär hundertprozentig gleiche Artefakt erzeugt werden kann. Dadurch wird jedem die Möglichkeit gegeben, zu verifizieren, dass eine Datei, die aus einem Repository bezogen wurde, tatsächlich durch die Verarbeitung eines bestimmten Quelltextes erzeugt wurde.

Der Quellcode wird hierdurch zum Single Point of Truth. Dieser stellt gleichzeitig diejenige Wahrheit dar, die durch die Konsumenten einer Software erwartet wird. Zumindest in Open-Source-Systemen ist es ihnen damit möglich, die Funktionsweise, Sicherheit usw. eines Artefakts trustless zu überprüfen. Dies ist mit zentralisierten Repositories sowie ohne Reproducible Builds nur über Vertrauen in die jeweiligen Repositories und die importierenden Autoren möglich[78].

Das Kompilieren von Software ist jedoch nicht immer eine einfache Aufgabe. Hierfür sind bestimmte Werkzeuge und Umgebungen notwendig, die ggf. einer Konfiguration bedürfen. Weiterhin sind diese auch häufig zeit- und ressourcenintensiv, sodass ein Build jeder Dependency einer Software unpraktikabel ist. Konsumenten ohne hohe Sicherheitsanforderungen sollten daher doch weitgehend auf Prüfsummen und Signaturen vertrauen. Um die Konfidenz in diese allerdings zu stärken, sollten sie durch Dritte verifiziert werden, die tatsächlich die Mühe der Kompilierung auf sich nehmen und ihre Ergebnisse veröffentlichen. Dies würde böswillige Akteure identifizieren und Konsumenten auf diesen Umstand aufmerksam machen. Ein solches Build Validator Network ist eine essentielle Erweiterung eines dezentralen Repositories und wird im späteren Verlauf genauer betrachtet werden.

Eine Frage, die sich hier stellt, ist die Notwendigkeit von Signaturen. Für das eben definierte Konzept reichen Prüfsummen aus und besitzen zusätzlich nicht den Umstand, dass die öffentlichen Schlüssel der Autoren sicher akquiriert werden müssen. Zudem ist eine Verknüpfung mit dem Autor nicht mehr notwendig, da eine eindeutige, direkte Beziehung zwischen Artefakt und Quellcode hergestellt wird.

Das Signieren von Build-Artefakten stellt jedoch einen weiteren Prüfpunkt dar, der den Artefakten weitere Glaubwürdigkeit verleiht. Neben Dateien können Autoren ihre Commits im VCS signieren. Dies würde den Autor direkt mit dem Quellcode verbinden und dessen Authentizität stärken, auch wenn der Code selbst weiterhin das zu validierende Objekt bleibt.

[78] Vgl. Reproducible Builds n. d.

Ein weiteres Problem stellt das Auffinden des Quellcode dar. Dies gestaltet sich ähnlich der Verbreitung von öffentlichen Schlüsseln für Signaturen. Allerdings wird der Quellcode in der Regel aus den Metadaten von zentralisierten Repositories verwiesen oder ist über die üblichen Suchmaschinen wie Google auffindbar. Mittlerweile hat sich ebenfalls GitHub als Standardanlaufstelle etabliert. Es stellt sich nur die Frage, ob es sich hierbei um das offizielle Entwickler-Repository handelt, wobei dies allerdings für die Verifikation wenig relevant ist, solange der Quellcode der gleiche ist.

Die Einführung von Reproducible Builds ist jedoch nicht immer direkt möglich, insbesondere wenn der Build einer Software von externen Faktoren abhängig ist oder sonstige Seiteneffekte besitzt[79]. Container Images und Packages aus dem AUR sind beste Beispiele hierfür, da innerhalb des Build-Prozesses oftmals Anwendungen aus externen Quellen bezogen werden müssen. Diese können sich ohne Vorwarnung ändern und ein Reproducible Build invalidieren. Im Prozess müssten daher Verfahren angewendet werden, die diesen Umstand mitigieren.

4.3. Architektur des dezentralen Repositorys

4.3.1. Kern-Repository

4.3.1.1. Übersicht

Das dezentrale Repository (dRepo) wird in zwei Komponenten unterteilt: Index und Storage. Abbildung 3 stellt diese beiden, die das Kern-Repository bilden, schematisch in Relation dar.

Der Index bildet die Verwaltungskomponente des Systems und wird als Smart Contract auf einer entsprechend permissionless, öffentlichen, Smart-Contract-fähigen Blockchain realisiert. Dieser beinhaltet die Struktur des Repositorys in Form von Gruppen, Packages und den dazugehörigen Versionen[80]. Des Weiteren wird eine minimale Menge an Metadaten hier gespeichert, die zur Adressierung und zur Verifikation der Inhalte notwendig sind. An dieser Stelle werden explizit keine Nutzdaten vorgehalten.

Die Benutzerverwaltung wird ebenfalls innerhalb des Indexes stattfinden. Diese ist auf einen minimalen Funktionsumfang reduziert, um die Komplexität an diesem

[79] Vgl. Reproducible Builds n. d.
[80] Vgl. Gatteschi et al. 2018, S. 63-65.

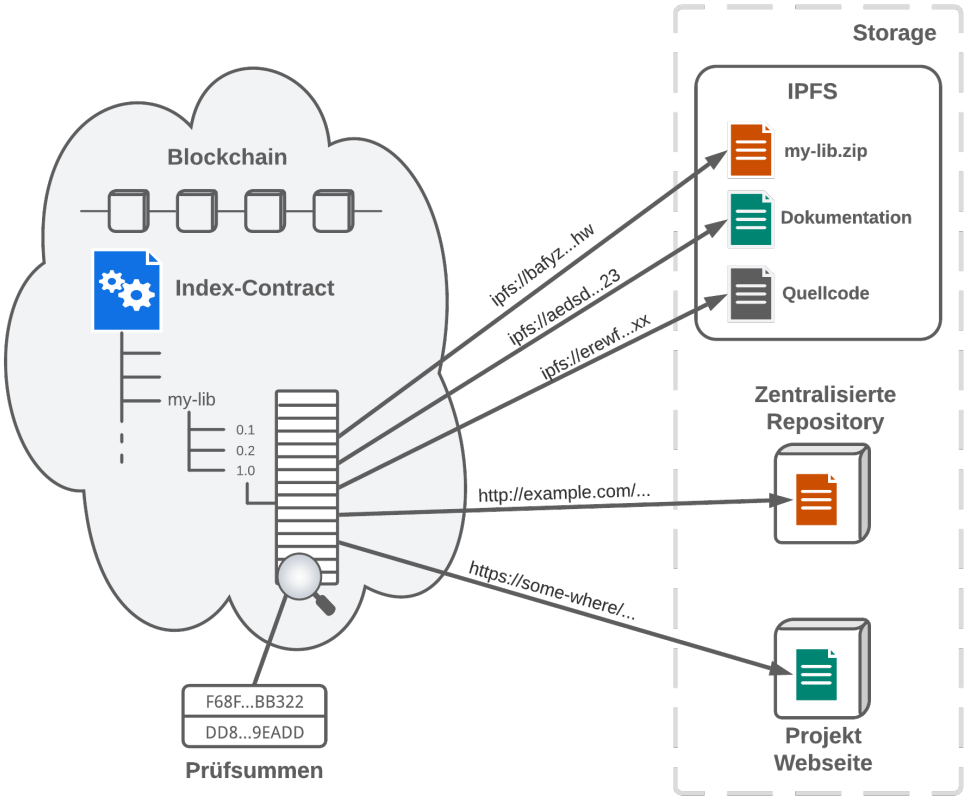


Abbildung 3: dRepo Komponenten

Punkt gering zu halten. Sie ist ausschließlich auf das Management von Schreibrechten der Autoren innerhalb einer Gruppe beschränkt.

Das Kern-Repository besitzt im eigentlichen Sinne keine eigene Storage-Komponente, sondern macht Gebrauch von existierender Infrastruktur. Vornehmlich sollen an dieser Stelle dezentrale Storage-Systeme verwendet werden. Primärer Kandidat, der hier empfohlen und als Standard genutzt werden sollte, ist das P2P-Netz IPFS[81]. Allerdings kann auch jedes andere System, welches Content Addressing unterstützt, genutzt werden.

Da die Verweise auf Inhalte aus dem Index mittels URLs ausgedrückt werden, sind auch Links auf zentral organisierte Speicherlösungen nutzbar und gar gewünscht. Auf diese Weise kann eine breite Menge an möglichen Bezugsquellen den Konsumenten angeboten werden.

[81] Vgl. IPFS Docs 2021.

4.3.1.2. Index

Der Smart Contract, der den Index repräsentiert, verwaltet grundlegend nur die Beziehung zwischen Autoren, Gruppen, Packages und Releases. In der Abbildung 4 ist diese skizziert. Dieser Zusammenhang wurde dem Konstrukt, welches sich in den untersuchten Repositories etabliert hat[82,83], nachempfunden.

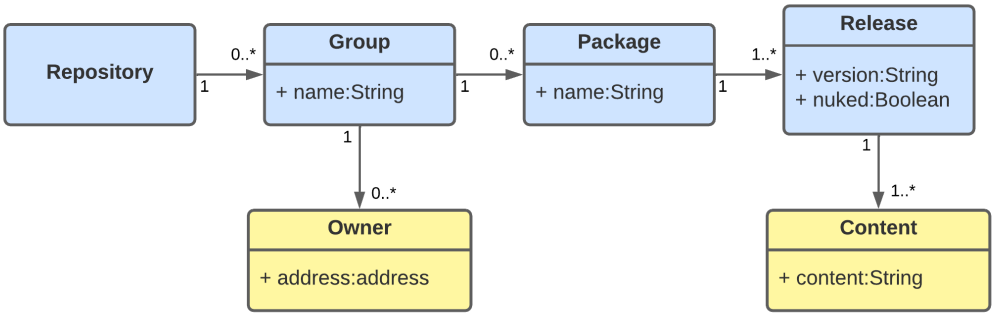


Abbildung 4: Index Klassendiagramm

Der Index beherbergt auf oberster Ebene benannte Gruppen. Diese können durch eine Liste von Autoren, *Owner*, verwaltet werden. Neben dieser Liste von Autoren bündelt eine Gruppe keine oder mehrere ebenfalls benannte Packages in sich. Ein Package stellt ein zu publizierendes Softwarepaket dar, beispielsweise eine Applikation oder eine Programmbibliothek, und muss mindestens ein betiteltes Release enthalten. Die jeweiligen Bezeichnungen von Gruppen, Packages und Releases können innerhalb bestimmter Grenzen, welche durch eine konkrete Implementation definiert werden müssen, frei gewählt werden. Dies ist durch die Nutzung von String-Typen ausgedrückt worden. Dies bedeutet ebenfalls, dass durch das Typsystem kein Versionsschema wie SemVer vorgegeben ist.

Ein Release ist ein logischer Blattknoten in der aufgestellten Baumstruktur und enthält die Nutzdaten, die für Konsumenten interessant sind. Bei diesen handelt es sich um eine Liste von formatierten Content-Strings, die Verweise und Metadaten enthalten. Es muss mindestens ein Eintrag in dieser Liste enthalten sein, um ein Release darstellen zu können. Weiterhin enthält ein Release ein Flag, welches dessen Nuked-Status definiert. Dieser Wert kann nur einmal geändert werden.

[82] Vgl. The Apache Software Foundation 2021b.
[83] Vgl. npm Docs n. d.(b).

Listing 1 gibt eine Übersicht über die Interface-Methoden des Index Contracts in Solidity[84]. Diese definieren die möglichen Interaktionen. Schreibende Methoden werden ausschließlich durch Autoren verwendet, während lesende Funktionen es auch den Konsumenten erlauben, den aktuellen Zustand des Repositorys zu erfahren. Weiterhin handelt es sich hierbei um einen Entwurf, der ggf. aufgrund technischer Bedingungen bzgl. Blocksize, Gas Fees, Storage Optimierung, etc. im Detail angepasst werden muss.

```
interface RepositoryIndex {

  function register(string memory groupName) external;

  function addOwner(string memory groupName, address newOwner) external;

  function removeOwner(string memory groupName, address owner) external;

  function release( string memory groupName,
                    string memory package,
                    string memory version,
                    string[] memory content) external;

  function nuke( string memory groupName,
                    string memory package,
                    string memory version,
                    string[] memory content) external;


  function groupExists(string memory groupName)
                    external view returns (bool);

  function packages(string memory groupName)
                    external view returns (string[] memory);

  function releases(string memory groupName,
                    string memory package)
                    external view returns (string[] memory);

  function getRelease(string memory groupName,
                      string memory package,
                      string memory version
                      ) external view returns ( string[] memory content,
                                                bool nuked);

  function owners(string memory groupName)
                  external view returns (address[] memory);
}
```

Listing 1: Interface des Repository Index

[84]Eine Beispielimplementation ist in Anhang B zu finden.

Bevor ein Autor in der Lage ist, seine Inhalte im dezentralen Repository zu veröffentlichen, muss er eine Gruppe erzeugen (`register`). Diese muss einen eindeutigen Namen besitzen und darf nicht zuvor registriert worden sein. Durch die initiale Erstellung einer Gruppe wird er zum ersten Owner. Als Owner ist es ihm gestattet, diese Rolle weiteren Autoren zu verleihen (`addOwner`). Ebenso kann ein Besitzer anderen Autoren dieses Recht entziehen (`removeOwner`). Dabei ist es explizit gestattet, dass ein Nutzer seine eigene Befugnis widerruft. Dies kann dazu führen, dass Gruppen ohne Owner für immer read-only sind.

Nachdem eine Gruppe angelegt wurde, kann ein Autor als Owner Packages und Releases veröffentlichen. Mithilfe der `release` -Methode muss er in der Gruppe ein Package mit Release und einer Liste von Verweisen definieren. Das Package wird hierbei implizit erzeugt. Weiterhin muss mindestens ein Verweis mit übergeben werden, damit dem Release Nutzdaten beiliegen. Obwohl Releases eigentlich immutable sind, kann die Methode erneut aufgerufen werden, um weitere Verweise hinzuzufügen. Diese Option muss gegeben sein, um mögliche große Transaktionen aufteilen und um neue Quellen und Metadaten im späteren Verlauf anfügen zu können. Existierende Einträge können hierdurch nicht gelöscht oder überschrieben werden.

Mittels der `nuke` -Methode hat der Autor die Möglichkeit, ein existierendes Release als invalide zu markieren. Optional kann er über neue Verweise und Metainformationen einen Grund für diese Entscheidung publizieren. Der Nuke-Status kann nicht revidiert werden.

Die rein lesenden Methoden des Contracts können genutzt werden, um durch den Index zu navigieren und um die Detailinformationen von Releases und Gruppen in Erfahrung zu bringen. Da die Standardschnittstelle vergleichsweise einfach gehalten ist, ist auch kein Suchen, insbesondere innerhalb der Verweise und Metadaten, an dieser Stelle möglich.

4.3.1.3. Storage

Der Content eines Release-Objekts besteht aus einer Liste von formatierten Strings, die unterschiedliche Aufgaben wahrnehmen. Die wichtigste von diesen ist die Adressierung von Softwareartefakten, welche die Konsumenten von einem Package beziehen möchten. Hierbei kommen vornehmlich Content Addressing Systeme zum Einsatz. IPFS ist die empfohlene Lösung und sollte als Mindestanforderung für jede Artefaktdatei genutzt werden. Parallel zum Content Addressing sollten auch URLs, die zentralisierte Lokationen von Dateien beschreiben, angegeben werden. Generell

ist es möglich und empfohlen, auf ein Artefakt auf unterschiedliche Art und Weise mehrfach zu verweisen.

Neben der Adressierung von Artefakten werden an dieser Stelle auch Daten zu deren Verifikation gespeichert. So sollen zu jeder verwiesenen Datei auch mehrere starke Prüfsummen existieren und als `content`-String codiert werden. Listing 2 zeigt einige Beispiele hierfür, wie mehrere Prüfsummen zu einem Artefakt definiert werden könnten. Weiterhin sollten auch Signaturen verwendet werden, um die Konfidenz zu erhöhen. Um diese jedoch sinnvoll nutzbar zu machen, sollten über entsprechende Links die passenden Schlüssel referenziert werden. Da es sich bei Signaturen um vergleichsweise großen Datensätze handelt, müssten diese ggf. auch ausgelagert werden, wenn ihre Speicherung on-chain zu kostspielig ist.

```
1  # Hauptartefakt in IPFS
2  ipfs://bafyz...hwq/my_lib.js
3
4  # Pruefsumme des Hauptartefakts als MD5
5  md5:d41d8cd98f00b204e9800998ecf8427e
6
7  # Quellcode in IPFS
8  source:ipfs://bpobeyz...rhwgr
9
10 # Dokumentation in IPFS
11 docs:ipfs://gvkrlsrw...fbiybr
12
13 # Hauptartefakt auf GitHub
14 https://github.com/.../my_lib-1.0.1.js
15
16 # Hauptartefakt in der NPM Registry
17 https://registry.npmjs.org/.../my_lib-1.0.1.tar.gz/my_lib.js
18
19 # Quellcode in BTFS
20 source:btfs://bpobeyz...rhwgr
```

Listing 2: Beispiele von Verweisschemata

Außerdem zeigt Listing 2 neben der Adressierung der Build Artefakte auch die Referenzierung von Quellcode und Dokumentation inkl. deren Verifikationsdaten. Insgesamt soll der `content` eines Releases Referenzen auf alle notwendigen Informationen enthalten, die ein Konsument benötigt. Es muss für ihn möglich sein, die Artefakte zu beziehen, sowie deren Authentizität nachweisen zu können. Weitere Daten, wie Dokumentation, aktive Deployments oder die Projektwebseite, sind ebenfalls möglich abzulegen, da das System explizit offen belassen wurde. Daher kann die Community beliebige Erweiterungen an dieser Stelle ausdrücken, die auch durchaus

sprachökosystemspezifisch sein können. Die verwendeten Dependency Tools haben am Ende die Aufgabe, die gegebenen Inhalte korrekt zu interpretieren.

4.3.1.4. Technologieauswahl

Blockchain

Für das Hosting der Index-Komponente des dezentralen Repositories bieten existierende, permissionless, public Blockchains beste Voraussetzungen. Das erste Argument für diese Technik ist, dass solche Blockchains im Gegensatz zu anderen Systemen bereits existieren. Eine Anwendung, die beispielsweise auf einer dezentralen Datenbank oder einer Reihe von Transparency Logs aufbaut, müsste nicht nur entwickelt, sondern auch die angemessene Infrastruktur erzeugt werden. Diese müsste anschließend durch ggf. freiwillige, unabhängige Teilnehmer dezentral betrieben werden. Im Falle einer passenden, produktiven Blockchain ist diese Infrastruktur bereits vorhanden, sodass der entsprechende Smart Contract dort prinzipiell nur noch deployed werden müsste.

Da auf ein schon funktionierendes Ökosystem zurückgegriffen wird, müssen für das Hosting des Index nicht neue Anreize geschaffen oder sich auf die Gutmütigkeit der Community verlassen werden. Die verwendete Blockchain wird bereits durch Parteien betrieben, die schon einen Ansporn besitzen, zum Erhalt des Netzwerks beizutragen[85,86,87]. Letztendlich ist dies meist ein monetärer Anreiz, der durch das Finanzsystem der Chain sowie dApps erzeugt wird. Der Index des Repositorys ist in diesem Fall nur eine weitere dApp in diesem Netzwerk.

Durch diesen Umstand genießt der Repository-Index den Schutz des Blockchain-Ökosystems und fügt gleichzeitig noch weiteren Wert hinzu. Da beispielsweise eine Blockchain wie *Ethereum* Werte von mindestens 400 Mrd. US-Dollar[88] verwaltet, besteht seitens beteiligter Stakeholder großes Interesse daran, dass dieses Netzwerk bestehen bleibt und Prozesse dort ungehindert und korrekt ablaufen können. Durch das Deployment des Index auf einer solchen Chain würden sich die Miner ebenfalls implizit für den reibungslosen Betrieb des Repositories interessieren. Sollte es gestört werden, hätte dies Auswirkungen auf die gesamte Chain und damit auch auf andere dApps. Dasselbe Argument gilt auch umgekehrt. Allgemein erhöht dies die Menge

[85] Vgl. Nakamoto 2008, S. 3.
[86] Vgl. Ethereum Wiki 2021.
[87] Vgl. Buterin et al. 2019.
[88] Stand Februar 2022, ETH Market Cap + DeFi Total Value Locked. Mehr Informationen siehe: https://defillama.com/chains und https://www.coingecko.com/en/coins/ethereum

an Stakeholdern, die direkt oder indirekt am Zustand des Repositorys teilhaben. Der Wert des dRepo als Teil der Software Supply Chain ist ebenfalls dem Wert der Chain hinzuzurechnen[89].

Aufgrund dessen, dass das Repository permissionless ist und ein First-come-first-served-Prinzip verfolgt, ist Spamming der schreibenden Funktionen ein Problem. Zentralisierte Repositories können dies durch Benutzerkonten und Rate Limits kompensieren. Komplexere Implementationen des Index können davon teilweise auch Gebrauch machen. Allerdings sind Gas Fees hierfür bereits ein Hinderungsgrund[90]. Dies sollte ebenfalls bei der Wahl einer Blockchain beachtet werden, sodass diese Fees nicht zu gering ausfallen.

Abgesehen von den ökonomischen Faktoren bieten Blockchains in der Regel genügend Rechenleistung für die einfachen CRUD-Funktionen des Indexes. Darüber hinaus sind sie in der Ausführung von Funktionen transaktional[91,92] und im Endeffekt single threaded[93], sodass, abhängig von der endgültigen Implementation, Nebenhäufigkeitsprobleme weitgehend vermieden werden können.

Letztendlich ist ebenfalls die Pseudoanonymität[94,95,96] der entsprechenden Blockchains ein weiteres wichtiges Argument, das für deren Nutzung spricht. Dies bestärkt den Punkt der Zensurresistenz, indem Autoren Inhalte veröffentlichen können, ohne dass auf ihre Identität direkt Rückschlüsse gezogen werden können.

InterPlanetary File System

Bei der Wahl der Storage-Komponente ist ebenfalls der Betrieb ein wichtiger Entscheidungsgrund. In diesem Fall sogar umso gewichtiger, da die Menge an Daten, die verbreitet werden müssen, im Vergleich zum Index deutlich größer ausfällt. Entsprechend ist hier mit höheren Kosten zu rechnen, um ein weltweit skalierendes System zu erhalten, wenn zentralisiertes Cloud Hosting sowie CDNs zum Einsatz kommen würden. Des Weiteren können sie aufgrund ihrer Zentralisierung keine Zensurresistenz bieten[97]. Dasselbe gilt auch für ein Netz aus Mirrors, wie es beispielsweise

[89] Vgl. Buterin et al. 2013.
[90] Vgl. Buterin et al. 2019.
[91] Vgl. Ethereum 2021.
[92] Vgl. Etherscan 2021.
[93] Vgl. Ethereum 2022.
[94] Echte Anonymität bei Nutzung bestimmter Privacy Techniken.
[95] Vgl. Nakamoto 2008, S. 6.
[96] Vgl. Wieczner 2017.
[97] Vgl. Eckert 2018, S. 846-849.

bei Debian zum Einsatz kommt. In diesem Fall wäre das System ebenfalls von der Gutmütigkeit von Sponsoren abhängig.

Während der Untersuchung existierender Repositories in Kapitel 3.2 haben sich P2P-Verfahren als weitgehend zensurresistent gezeigt. Daher sollten solche Systeme eine zentrale Rolle in der Datendistribution spielen, um Inhalte ebenfalls mit höherer Verfügbarkeit bereitzustellen[98]. IPFS ist ein solches Netzwerk, das im Rahmen von Web3 zunehmend an Bedeutung gewinnt. Durch den NFT-Hype Ende 2021 wurde das System zum de facto Speicherort dezentraler Kunst[99] und sichert auch die Frontends einer Vielzahl an dApps. Letztere bieten hier ihre Interfaces[100] parallel zum zentralisierten Hosting an, um manipulationsfrei, anonym und autonom agieren zu können[101].

Weiterhin bauen diverse dezentrale Storage-Dienste wie FileCoin, Arweave und Crust auf Integrationen mit dem Protokoll auf[102,103,104]. Sie ermöglichen eine sichergestellte Verteilung im Netzwerk über bestimmte Zeiträume. Andere Infrastruktur Dienstleister wie Cloudflare[105] bieten Gateways an, die es Nutzern erlauben mittels HTTP auf Inhalte des IPFS Netzwerks zuzugreifen. Gleichzeitig wird das System in Browser eingegliedert, sodass der Umweg über ein Gateway eingespart werden kann[106,107].

Diese Use Cases und Integrationen sind Anzeichen für die breiter werdende Adaption des Protokolls. Alternative Systeme wie BitTorrent sind zwar weit verbreitet, bieten aber derzeit nicht diese Integrationsmöglichkeiten. Das auf BitTorrent basierende BTFS, welches IPFS konzeptionell ähnelt, mangelt es bisher an Adaption und Reife[108].

Daher ist als Empfehlung primär IPFS als Datenspeicher für das dezentrale Repository gewählt worden. Der Repository-Index ist allerdings so gestaltet, dass keine harte Abhängigkeit zu IPFS besteht. Daher können beliebige Protokolle genutzt werden, die sich in Zukunft durchsetzen werden oder durch bestimmte Communities

[98] Vgl. Schoder et al. 2005, S. 2-4.
[99] EIP-721 definiert, dass Metadaten, inkl. Bilder, off-chain als URL referenziert werden. IPFS wird in diesem Zusammenhang aufgrund von Immutablity als sicherer Speicherort gegenüber HTTP Systemen angesehen. Mehr Informationen siehe: https://eips.ethereum.org/EIPS/eip-721.
[100] Web-Frontends von dApps werden im Web3-Zusammenhang als Interfaces bezeichnet.
[101] Vgl. Uniswap 2020.
[102] Vgl. Crust n. d.
[103] Vgl. The Arweave Project 2019.
[104] Vgl. Filecoin Docs 2022.
[105] Vgl. Cloudflare Docs 2022.
[106] Vgl. Bondy 2021.
[107] Vgl. Batt 2021.
[108] Vgl. Y. Chen et al. 2017, S. 2652-2653.

präferiert werden. Die Empfehlung dezentrale, zensurresistente Systeme zu verwenden, bleibt davon unberührt.

4.3.2. Erweitertes Repository

4.3.2.1. Index

Die Daten des Index sind auf der Blockchain (on-chain) für immer gespeichert und aufgrund der Historisierung stets wieder auslesbar[109]. Es besteht aber derzeit ein Problem darin, wie Konsumenten auf diese Informationen zugreifen können.

Um die Daten einer Blockchain wie Ethereum lesen zu können, ist es notwendig einen Node bzw. Client des Netzwerkes zu verwenden, da nur diese direkten Zugang zu den Inhalten besitzen[110]. Um tatsächlich uneingeschränkten Zugriff zu erhalten, ist daher das Hosting eines eigenen Nodes notwendig. Dies erfordert jedoch je nach Setup und Anforderungen ein größeres Maß an Ressourcen.

Um allerdings nur sporadisch Daten auslesen zu wollen, ist ein solcher Aufwand womöglich überdimensioniert. Daher ist die Nutzung Nodes Dritter, beispielsweise Infura oder Alchemy, verbreitet. So existieren sowohl öffentliche als auch kommerzielle Anbieter, welche dann gewisse Servicegarantien leisten. Des Weiteren bieten sie Kommunikationsmöglichkeiten auf Basis üblicher Web-Protokolle wie HTTP oder WebSockets[111].

Die Nutzung fremder Nodes ist jedoch als problematisch anzusehen, da es sich um zentrale Entitäten handelt. Diese könnten Lese- und Schreibzugriffe zensieren oder blockieren. Daher müssen Nutzer dieser Nodes entsprechend großes Vertrauen in dieselben setzen. Dezentrale Datennetze wie The Graph könnten zumindest auf der lesenden Seite Abhilfe schaffen, sofern diese sich tatsächlich als ausreichend dezentral und damit zensurresistent erweisen. Jedoch erwarten auch diese Systeme monetäre Kompensation[112].

[109] Vgl. Brühl 2017, S. 136-137.
[110] Vgl. Schlatt et al. 2016, S. 7-12.
[111] Vgl. Iansiti und Lakhani 2017.
[112] Vgl. The Graph 2022.

4.3.2.2. Storage

Obwohl Peer-to-Peer-Systeme wie IPFS technisch zensurresistent und durch die Verteilung weitgehend ausfallsicher sind, sieht die aktuelle Realität hingegen anders aus. Aufgrund der Datenüberwachung durch ISPs und der Offenheit der Protokolle ist es bestimmten Parteien möglich, die Identität eines Teilnehmers zu erfahren[113]. Entsprechend können jene Parteien anschließend Maßnahmen ergreifen, um die Verteilung von Daten im Netzwerk zu unterbinden. In der BitTorrent-Filesharing-Szene ist daher die Nutzung von VPNs üblich, um das Aufdecken der eigenen Identität zu verhindern. Weiterhin werden auch private Netzwerke wie *I2P* oder *Tor* verwendet, um ähnliche Resultate zu erzielen[114].

Um die Zensurresistenz des Repositorys zu stärken, ist die Nutzung vergleichbarer Techniken notwendig, um die Identitäten der Teilnehmer zu schützen, sodass letztendlich zu zensierende Softwarepakete weiterhin verfügbar bleiben. Dies ist eine essentielle Bedingung, die erfüllt werden muss, um die Integrität des Systems zu sichern.

Weiterhin kann es trotz der Kombination verschiedener Quellen und der Nutzung von P2P-Verfahren zu Datenverlusten kommen, wenn die zentralisierte Quellen geschlossen und in P2P-Netzen die entsprechenden Dateien nicht mehr geteilt werden. Die Nutzung von dauerhaften Hosting-Lösungen ist daher zu empfehlen, sofern dies möglich ist. Systeme wie Arweave versprechen eine permanente Datenbereitstellung, welche eine Antwort sein könnten[115].

4.3.2.3. Build Verification Network

Reproducible Builds sind die einzige Möglichkeit, die Authentizität eines Build-Artefakts eindeutig zu beweisen[116]. Da jedoch das Ausführen von Builds zeit- und ressourcenintensiv ist, ist es für Konsumenten unpraktikabel, jedes Artefakt auf diese Weise zu überprüfen. Daher ist die Einrichtung eines Build Verification Networks eine der wichtigsten Erweiterungen um den Kern des dezentralen Repositorys. Dessen Aufgabe ist es, dezentral jedes neue Release mittels Durchführung seines Builds zu verifizieren.

[113] Vgl. Danielis et al. 2010, S. 1-2.
[114] Vgl. Azmat et al. 2009, S. 170-171.
[115] Vgl. Arweave n. d.
[116] Vgl. Reproducible Builds n. d.

Da keiner einzelnen, zentralen Instanz vertraut werden darf, muss in dem Netzwerk das Build mehrfach durch unabhängige Teilnehmer ausgeführt werden. Wird das Build durch genügend Teilnehmer überprüft, kann mithilfe von Konsensverfahren eine Aussage über die Validität eines Releases getroffen werden. Dieses Ergebnis würde separat den Konsumenten des dezentralen Repositorys zur Verfügung gestellt werden. Sie können dadurch entscheiden, ob sie den Artefakten vertrauen möchten, ohne dieses selbst zu prüfen.

Um ein solches System realisieren zu können, müssen Teilnehmer gefunden werden, die in der Lage sind, solche Verifikationen durchzuführen. Aufgrund des Aufwandes ist ein anreizbasiertes System, in dem valide Teilnahme belohnt wird, wahrscheinlich erforderlich. Weiterhin ist es notwendig, Software Builds auf diese automatisierte Prüfung vorzubereiten. Mittlerweile ist Continuous Integration weit verbreitet und passende Werkzeuge werden durch große Entwicklungsplattformen wie GitHub und GitLab angeboten. Jedoch handelt es sich hier weitgehend um proprietäre Produkte mit eigenen Konfigurationen und Prozessen, sodass diese nicht miteinander kompatibel sind. Daher müssen Teilnehmer entweder all diese Systeme unterstützen oder es muss ein gemeinsamer Standard für die Beschreibung von Build-Prozessen definiert werden.

4.3.2.4. Review Network

Da der Quellcode einer Software der Single Point of Truth ist, ist neben der Verifikation des Builds auch die Untersuchung des Quellcodes eine essentielle Aufgabe. Während Builds automatisiert geprüft werden können, ist dasselbe nicht in Gänze mit dem Quelltext möglich. Neben statischer Analyse zum Auffinden von Code Smells, typischer Bugs usw. ist ein manuelles Review die beste Möglichkeit, Logikfehler und kontextübergreifende Probleme ausfindig zu machen[117].

Es existieren bereits Unternehmen, die sich auf solche Reviews spezialisiert haben. Allerdings ist eine Überprüfung durch den einhergehenden Aufwand stets kostspielig. Weiterhin handelt es sich bei diesen wiederum um zentralisierte Organisationen, die durch ihren Namen und Reputation versuchen eine Vertrauensposition einzunehmen. Ein solches Konstrukt untergräbt die Eigenschaften des dezentralen Repositories und ist daher nur bedingt einsetzbar.

Ähnlich der Build-Verifikation ist auch an dieser Stelle der Aufbau eines dezentralen Netzwerks notwendig, in dem Teilnehmer den Quellcode nachvollziehen und bewer-

[117] Vgl. Rösler et al. 2013, S. 1-5.

ten müssen. Etwaige Fehler und Probleme müssen identifiziert und in der Community diskutiert und beurteilt werden. Konsumenten eines Artefakts können diese Informationen verwenden, um abzuschätzen, ob eine Nutzung der Software mit einem akzeptablen Risiko verbunden ist.

Anders als bei dem Beweis der Authentizität eines Build-Artefakts, besitzt ein manuelles Review stets einen gewissen Anteil an menschlichem Versagen, sodass Fehler übersehen werden können. Mehrfache Reviews können dieses Risiko zwar reduzieren, dennoch bleibt ein Restrisiko bestehen. Daher sind nicht nur wiederkehrende Reviews erforderlich, sondern auch die rasche Verbreitung der Ergebnisse. Es existieren bereits Netzwerke, die solche Informationen in der Entwickler-Community verbreiten, dennoch werden die Daten kaum in Dependency Tools wie Maven aufgegriffen, um Entwickler oder Continuous Integration Ketten auf etwaige Probleme aufmerksam zu machen.

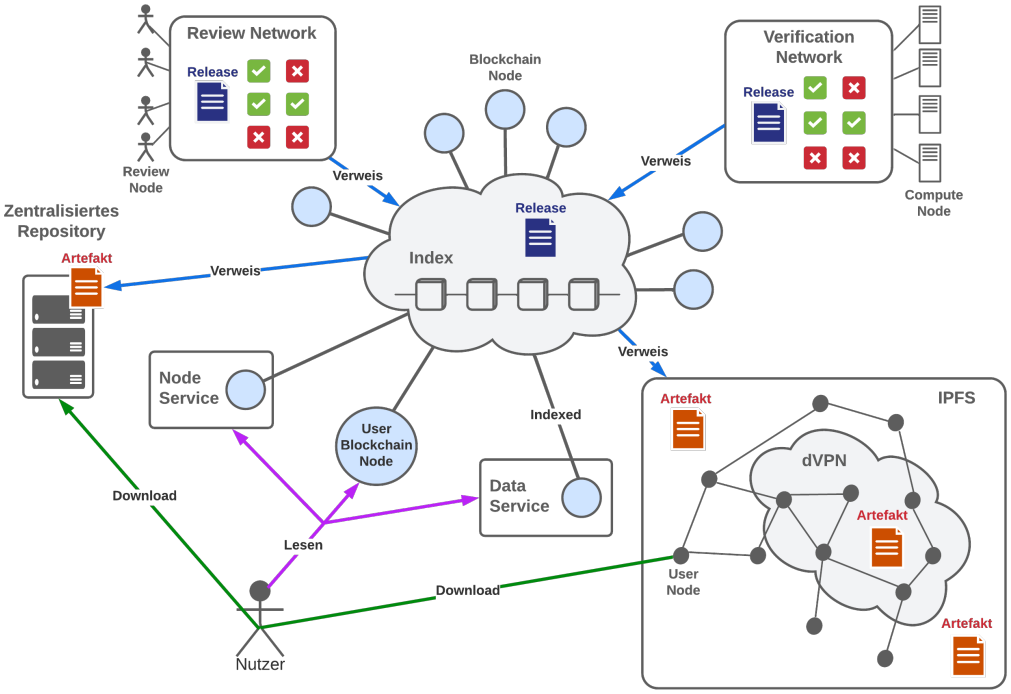


Abbildung 5: Gesamtübersicht des erweiterten, dezentralen Repositorys

In Abb. 5 ist das gesamte erweiterte Repository skizziert. Es veranschaulicht die unterschiedlichen Möglichkeiten eines Nutzers, Indexdaten zu lesen und daraufhin Artefakte herunterzuladen. Weiterhin steht es ihm frei, zusätzliche Informationen aus Review und Verification Network zu beziehen.

4.4. Anforderungsverfolgung

Das Konzept des dezentralen Repositorys wurde auf Grundlage der Anforderungen aus Kapitel 4.1 erstellt. Es wurden Anforderungen aus der Analyse der untersuchten Software Repositories sowie aus den Eigenschaften der Dezentralisierung erhoben. In der folgenden Aufschlüsselung wird die Abdeckung der einzelnen Anforderungspunkte durch das Konzept besprochen.

Allgemeine Anforderungen

A1 - Einfacher, zentraler Einstieg für Konsumenten

Anmerkung	Es steht eine Entscheidung aus, ob ein globales Repository für die gesamte Softwareentwicklung oder eines für jede Software Community eingerichtet werden sollte. Letzteres würde der aktuellen Repository Landschaft entsprechen, indem dezentrale Alternativen zu Maven Central, der NPM Registry, usw. existieren. Unabhängig davon würde in jeder Community ein dezentrales Repository unter einer bestimmten Adresse existieren. Problematisch kann jedoch der Zugriff auf diese sein, da ein Zugang zur Blockchain benötigt wird. Via HTTP und JSON-RPC können Dependency Tools und Ähnliches Anfragen an den Index Contract senden. Hierzu ist allerdings ein entsprechender Blockchain Node notwendig. Öffentliche Nodes stehen nur bedingt zur Verfügung. Daher ist ggf. der Betrieb eines eigenen Nodes notwendig. Alternativ können aggregierende Datendienste Dritter verwendet werden. Insgesamt steht kein allgemeingültiger, kostenfreier Zugangspunkt zur Verfügung. Anwender sind, ähnlich wie bei Web3 dApps, für ihren eigenen Zugangsnode verantwortlich. Dieser Sachverhalt ist allerdings auch abhängig von der letztlich gewählten Blockchain.
erfüllt	✗ Nur bedingt abgedeckt

A2 - Koordinaten

Anmerkung	Es wird ein Koordinatensystem aus *Groupname* - *Packagename* - *Version* im Repository Index verwendet. Das Format der Version eines Releases ist nicht fest definiert und wird durch die API nicht typsicher forciert.
erfüllt	✓

A3 - Hosting einsatzfertiger Softwarepakete

Anmerkung	Software-Community-abhängig sollen primär vorkompilierte bzw. einsatzfertige Softwarepakete publiziert werden. Im Index des Kern-Repositorys kann dies technisch bedingt nicht validiert und forciert werden. Die Beanstandung möglicher Missstände muss durch Komponenten des erweiterten Repositorys stattfinden. Durch das Format der Inhaltsverweise ist das Hosting von Paketvarianten möglich.
erfüllt	✓

A4 - Schnelles Onboarding und Schreibrechte

Anmerkung	Die Erstellung eines Benutzerkontos ist aufgrund der Umgebung nicht notwendig. Autoren erhalten durch die Registrierung einer Gruppe die Schreibrechte zum Erzeugen von Packages und Releases.
erfüllt	✓

A5 - Leserechte

Anmerkung	Aufgrund der technischen Basis einer öffentlichen Blockchain sind alle Daten des Repository Indexes für alle Teilnehmer stets lesbar. Öffentliche Lese-Funktionen im Interface ermöglichen simple Navigation durch die Inhalte.
erfüllt	✓

A6 - Alternative Repositories

Anmerkung	Der Repository Index ist lediglich als Interface definiert. Damit sind alternative Implementationen und Deployments möglich. Durch das offene API-Design wird die Nutzung von Proxy-Repositories zur Erweiterung der Kernfunktionen einer deployed Instanz empfohlen. Auf diese Weise lassen sich komplexere Rollen- und Rechtekonzepte implementieren.
erfüllt	✓

A7 - Öffentliche Strukturen

Anmerkung	Im Repository Index Interface ist ein Mindestmaß an lesenden Funktionen definiert. Diese erlauben die Navigation innerhalb der Elemente von Gruppen und das Auslesen alle Package-Informationen. Erweiterte Funktionen können mithilfe von Proxies oder externer Dienste bereitgestellt werden.
erfüllt	✓

A8 - Validation von Artefakten

Anmerkung	Zu jedem Release-Artefakt sollen mehrere Prüfsummen basierend auf starken Hash-Funktionen beigefügt werden. Sie werden direkt im Index on-chain gespeichert. Die Nutzung von GPG-Signaturen ist optional. Die Signaturen könnten aufgrund ihrer Größe off-chain gespeichert müssen und müssten daher verwiesen werden. Dasselbe gilt für die passenden Public Keys. Da das Verweisformat offen gestaltet ist, können weitere Authentizitätsinformationen und -Systeme nachgereicht werden. Im erweiterten Repository werden diese Informationen verwendet, um die Authentizität verlinkter Artefakte zu beweisen.
erfüllt	✓

A9 - Historisierung

Anmerkung	Alle schreibenden Interaktionen mit dem Repository-Index werden auf der Blockchain festgehalten. Weiterhin können Informationen von existierenden Gruppen, Packages und Releases nicht verändert oder gelöscht werden. Es ist nur möglich, sie zu erweitern. Ein Überschreiben oder Ersetzen mit anderen Inhalten ist nicht möglich. Durch Content Addressing sind Nutzdaten im P2P-Netz IPFS unveränderlich. Lediglich ihre Verfügbarkeit kann aufgrund des Verfahrens variieren.
erfüllt	✓

A10 - Kompatibilität mit Web-Infrastruktur

Anmerkung	Der Repository Index kann mittels öffentlicher und privater Nodes über JSON-RPC per HTTPS angesprochen werden. Dies stellt weitreichende Kompatibilität mit anderen Web-Ressourcen her. Für das Betreiben eigener Blockchain Nodes, um direkten Zugriff zu erlangen, sind allerdings plattformspezifische Protokolle notwendig. Diese erfordern teilweise Anpassung der Netzwerkinfrastruktur. Artefakte sollen primär in IPFS gespeichert werden. Für die direkte Teilnahme ist freier Zugang zum Internet notwendig. Mittels Gateways ist allerdings auch der Zugriff über HTTPS möglich. Parallel sollen Artefakte auch auf zentralen Plattformen zur Verfügung gestellt werden. Diese sind typischerweise auch per HTTPS erreichbar. In beiden Fällen ist Zugriff spätestens über existierende, HTTP-kompatible Infrastruktur möglich.
erfüllt	✓

A11 - Management existierender Inhalte

Anmerkung	Das dezentrale Repository bietet keine Möglichkeit, die Inhalte, die von Autoren publiziert wurden, zu bearbeiten oder zu löschen. Dies ist explizit eine Entscheidung, um Zensurresistenz zu gewährleisten. Inhalte können weder im Index noch in IPFS technisch entfernt werden. Die `nuke`-Funktion bietet Autoren jedoch ein Weg, um Releases als invalide zu markieren. Weiterhin sollen im erweiterten Repository Build-Verifikation- und Reviewhinweise auf problematische Inhalte existieren.
erfüllt	✗

A12 - Geschlossene Bereiche

Anmerkung	Das öffentliche dRepo lässt aufgrund der technischen Implementation keine geschlossenen Bereiche zu. Sie können jedoch off-chain mithilfe von Proxy-Repositories oder Verschlüsselung realisiert werden.
erfüllt	✗

A13 - Quellcode und Dokumentation	
Anmerkung	Es wird empfohlen, dass neben Build-Artefakten auch Quellcode und Dokumentation im dRepo veröffentlicht werden. Die offene Verweislogik des Index schränkt dies nicht ein. Es liegt allerdings in der Verantwortung von Autoren diese Daten zu publizieren.
erfüllt	✓

Dezentrale Anforderungen

D1 - Trustless	
Anmerkung	Für das Deployment des Index muss eine Blockchain ausgewählt werden, die als trustless eingestuft werden kann. Der Smart Contract Code des Index muss entweder immutable sein oder Upgradefähigkeit muss über dezentrale Prozesse gesichert werden. Nutzdaten werden über eine Vielzahl an Quellen angeboten. Speziell in IPFS werden die Daten (pseudo)-anonym geteilt und sind über Content Addressing eindeutig und in ihrer Integrität gesichert.
erfüllt	✓

D2 - Permissionless	
Anmerkung	Aufgrund der Nutzung einer öffentlichen Blockchain sind generell alle Daten auslesbar. Das Interface des Repository Index schränkt den Lesezugriff in keiner Weise ein. Autoren besitzen vollen Schreibzugriff innerhalb ihrer Gruppen. Beim Zugriff auf die Blockchain über Dienste Dritter muss jedoch mit Manipulation gerechnet werden. Daher wird Nutzung eines eigenen Nodes empfohlen. Daten in IPFS sind generell für jeden lesbar.
erfüllt	✓

D3 - Immutable	
Anmerkung	Der Repository Index ist so gestaltet, dass niemand Inhalte überschreiben oder löschen kann. Es ist nur das Hinzufügen weiterer Inhalte möglich. Über Content Addressing sind Dateien in IPFS immutable.
erfüllt	✓

D4 - Verfügbarkeit	
Anmerkung	Der Repository Index wird, abhängig der Blockchain, verteilt von unabhängigen Teilnehmern betrieben. Weiterhin kann ein eigener Blockchain Node verwendet werden. Damit ist zumindest der Lesezugriff höher verfügbar. Schreiboperationen sind abhängig von einem funktionierenden Konsensverfahren. Es soll eine Vielzahl von Quellen für Nutzdaten verwiesen werden. Weiterhin werden Inhalte in IPFS durch verschiedene, unabhängige Teilnehmer angeboten. Diese Punkte sorgen für eine erhöhte Verfügbarkeit der Dateien.
erfüllt	✓

D5 - Zensurresistent	
Anmerkung	Durch die Blockchain und die Implementation des Repository Index sind Inhalte effektiv unveränderbar. Durch direkten Zugriff auf die Blockchain kann ebenfalls bei schreibender Interaktion mit dem Index keine Manipulation stattfinden. Verwiesene Artefakte können durch Prüfsummen und Content Addressing eindeutig validiert werden.
erfüllt	✓

D6 - Authentizität und Verifizierbarkeit	
Anmerkung	Aufgrund der Offenheit der Blockchain können alle Transaktionen mit dem Index nachverfolgt und der Quellcode des Index eingesehen werden. Damit ist eine Prüfung der Inhalte dort möglich. Artefakte sind über Prüfsummen validierbar. Des Weiteren soll der Reproducible-Build-Ansatz verfolgt werden, sodass im erweiterten Repository der Zusammenhang zwischen Quellcode und Build Artefakt eindeutig hergestellt werden kann.
erfüllt	✓

D7 - Integration	
Anmerkung	Das dRepo verwendet in der Datenhaltung von Nutzdaten Verweise auf existierende Infrastrukturen und Systeme. Gleichzeitig bietet der Repository Index über sein Interface eine API an, welches über JSON-RPC programmatisch angesprochen werden kann.
erfüllt	✓

Das Konzept des dRepos erfüllt fast alle aufgestellten Anforderungen. Lediglich das administrative Management von Inhalten (A11) und die Nutzung geschlossener Bereiche (A12) werden nicht erfüllt. Dies waren bewusste Entscheidungen, um das System offen (D2) und zensurresistent (D5) zu gestalten. Die genannten Anforderungen standen in direktem Konflikt zueinander. Mögliche eigene Implementationen eines dezentralen Repositories können diesen Fokus allerdings verlagern.

5. Adaption des dRepos im NPM-Frontend-Ökosystem

5.1. Aufbau moderner Web Application Frontends

Laut den Umfragen *State of JS* und *State of CSS* von 2020 wurden Frontends moderner Webapplikationen vorwiegend auf Basis von JavaScript-Bibliotheken sowie CSS-Preprozessoren entwickelt. Eine Kombination aus NPM, Webpack, TypeScript, React und SASS ist in 2020 eine der am häufigsten zu findenden gewesen[118]. Dieses Setup zeichnet sich dadurch aus, dass der Quellcode nicht nur weitgehend kompiliert werden muss, sondern dass dieser kompilierte Code anschließend in einer oder einigen wenigen Dateien zusammengefasst wird[119]. Diese Build-Artefakte werden schließlich im Browser der Anwender ausgeführt.

Aus dem genannten Stack liegen insbesondere Dependency Manager wie NPM oder auch *Yarn*, sowie Build Tools wie *Webpack* für die Arbeit mit einem dezentralen Repository im Fokus. Das NPM CLI Werkzeug ist in diesem Fall für das Beziehen aller Dependencies verantwortlich, die in der `package.json` bzw. `package-lock.json` Datei in dem jeweiligen Projekt definiert sind[120]. Nach Standardverhalten werden die entsprechenden Artefakte der Packages aus der NPM Registry heruntergeladen und in einem lokalen Cache gespeichert[121]. Gleichzeitig werden die Artefakte entpackt in das Projektverzeichnis, `node_modules`, kopiert, sodass sie dort weiterverarbeitet werden können. Hierbei kann es sich um eine breite Variation an Inhalten handeln, von JavaScript- und CSS-Bibliotheken bis hin zu Bildern und Schriftarten.

Die Abhängigkeiten werden in der `package-lock.json` Datei automatisiert, exakt definiert. Sie enthält neben den Koordinaten eines Packages auch die konkrete URL,

[118] Mehr Informationen siehe: https://2020.stateofjs.com/en-US/technologies/ und https://2020.stateofcss.com/en-US/technologies/
[119] Vgl. Niranga 2015, S. 20-22.
[120] Vgl. npm Docs n. d.(j).
[121] Vgl. npm Docs n. d.(h).

von der das Artefakt bezogen wurde, sowie dessen Prüfsumme[122]. Diese generierten Informationen sollten in dem jeweiligen VCS gesichert werden, sodass die Continuous Integration Pipeline präzise Informationen für ein Build verwendet und im Falle von Abweichungen das Build fehlschlagen lässt. Diese exakten Daten sind nicht in der `package.json` Datei enthalten, die durch Entwickler gepflegt wird, sondern werden implizit generiert. Andere Dependency Manager neben NPM wie beispielsweise Yarn arbeiten nach ähnlichen Prinzipien[123].

Das Build der Anwendung würde durch Webpack ausgeführt werden. Es handelt sich hierbei um einen Module Bundler, der mithilfe entsprechender Loader Programmcode und andere Inhalte für die Ausführung im Browser aufbereitet. Dabei können verschiedene Transformationen eingesetzt werden, um unter anderem Code zu kompilieren, minifizieren und Tree Shakes durchzuführen oder auch Bilder in andere Formate zu überführen. Es wird versucht all diese Inhalte in möglichst wenigen Dateien zusammenzufassen, damit der Browser des Nutzers die Applikation effizient beziehen kann. Der Download weniger großer Dateien kann schneller durchgeführt werden als das Herunterladen vieler kleiner.

5.2. Migration

Nachdem der Index eines dezentralen Repositorys für eine Software Community deployed wurde, muss dieser mit Inhalten gefüllt werden. Hierbei muss unterschieden werden, ob ein Repository von Grund auf neu aufgebaut wird, oder ob die Daten aus einem zentralisierten Repository migriert werden sollen. Sofern bereits ein zentrales Repository existiert, sollte dieses mit dem neuen dRepo zumindest für einige Zeit parallel existieren und neue Inhalte in beiden gleichzeitig publiziert werden.

Da die initiale Implementation des Index Contracts öffentlich und nach dem First-come-first-served-Prinzip arbeitet, können Gruppen- und Packagenamen prinzipiell von jedem besetzt werden. Dies kann bei existierenden Projekten und Repositories problematisch sein, wenn durch Dritte die entsprechenden Namen zuerst registriert werden. Dies könnte durch koordinierte Registrierungen beim initialen Deployment des Indexes mitigiert werden.

Eine korrekte Überführung der existierenden Namen ist für die konsumentenseitige Migration besonders wichtig, da andernfalls unter denselben Koordinaten aus Gruppenname, Packagename und Version des zentralen Repositorys fremde Artefakte

[122] Vgl. npm Docs n. d.(i).
[123] Vgl. Wittern et al. 2016, S. 351-353.

aufgefunden werden. Es liegt zwar im Aufgabenbereich der Konsumenten, diesen Sachverhalt zu prüfen, dennoch entstehen hierdurch große Risiken für die existierende Supply Chain. Reviews, die aus dem Repository entstehen, sollten solche Risiken global publizieren, um Konsumenten bei der Absicherung ihrer Dependencies zu unterstützen.

Nachdem Projekte ihre entsprechenden Bereiche im Index registriert haben, müssen sie sämtliche existierenden Releases aus dem zentralen Repository überführen. Dazu müssen sie im dRepo jeweils neue Releases erzeugen und mit entsprechenden Verweisen befüllen.

Eine tatsächliche Migration der Nutzdaten ist im Sinne eines Umzugs der Build-Artefakte nicht notwendig, da im Index Verweise auf die bestehende Lokation der Daten gewünscht sind. Das zentrale Repository und das neue dRepo sollten zunächst koexistieren. Konsumenten können daher auch nach der Umstellung des Repositorys die Dateien aus derselben Quelle beziehen. Daher können harte Referenzen, wie sie beispielsweise in der `package-lock.json` von NPM-Projekten existieren, zunächst weiterverwendet werden, ohne dass der Build bricht. Schließlich sollte jedoch ein Umzug auf dezentrale Quellen stattfinden. Die Autoren von Softwarepaketen müssen diese Quellen im Repository mit jedem neuen Release bereitstellen. Dazu müssen alle Release-Artefakte, bestehende und neue, im IPFS-Netzwerk geteilt werden. Dadurch können die entsprechenden Links erzeugt und zusammen mit Prüfsummen und Signaturen in den Repository Index publiziert werden.

5.3. Entwicklung

5.3.1. Workflow

Bei der Verwendung eines dezentralen Repositorys wird der Entwickler keine gravierenden Änderungen an seinem Arbeitsprozess einführen müssen. Das Koordinatensystem, das er zur Adressierung seiner gewünschten Dependency nutzt, bleibt erhalten. Prinzipiell sollten genau die gleichen Artefakte wie bisher bezogen werden können.

Die notwendigen Anpassungen finden für ihn im Hintergrund statt. Sein genutzter Dependency Manager, beispielsweise `npm`, muss in der Lage sein, neben der NPM Registry auch den Index des dRepo auslesen zu können. Mithilfe eines Blockchain Clients könnten die Daten direkt on-chain oder mittels eines Datendienstes gar in

aufbereiteter Form entnommen werden. Anschließend hat der Dependency Manager die Möglichkeit, zwischen verschiedenen Downloadquellen auszuwählen. Alternativ könnte er die Daten des Indexes auch lediglich zur Verifikation der Informationen aus der NPM Registry verwenden.

5.3.2. Caching und Datenverteilung

Der lokale Cache, der derzeit durch das `npm` Werkzeug bei jedem Download einer Dependency angereichert wird, könnte gleichzeitig als Datenquelle für andere Entwickler dienen, welche die Artefakte über beispielsweise IPFS beziehen möchten. Sobald durch `npm` die Dateien aus der NPM Registry heruntergeladen und im Cache abgelegt werden, kann der ggf. separate IPFS Client diese Dateien im Peer-to-Peer-Netzwerk anbieten. Über einen solchen Mechanismus wird transparent eine Brücke zwischen dem zentralisierten Datenspeicher und dem dezentralen aufgebaut. Diese muss durch den Entwickler nicht explizit administriert werden. Dieses Konzept ist in Abb. 6 dargestellt. Der violette Pfad beschreibt das Lesen des Index und der grüne den anschließenden Download aus verschiedenen Quellen.

Weiterhin sind in Organisationen teilweise private Software Repositories wie Nexus und Artifactory anzufinden. Diese werden dafür genutzt, um in der Organisation eigene, nicht öffentliche Packages den internen Entwicklern zur Verfügung zu stellen. Darüber hinaus bieten sie die Funktion eines Proxy-Caches. Ein solcher Cache speichert die Artefakte und Package-Informationen eines Upstream-Repositorys, beispielsweise der NPM Registry, sobald sie intern angefragt werden. Diese Daten werden anschließend im privaten Netzwerk schneller und ggf. ausfallsicherer angeboten. Die direkte Abhängigkeit zur Verfügbarkeit des Upstream Repositorys ist damit gebrochen.

Das private Repository nimmt damit dieselbe Position wie der lokale Cache eines Entwicklers ein. Es kann so die Upstream Dependencies ebenfalls im öffentlichen P2P-Netzwerk verteilen. Dies ist internen Entwicklern in geschützten Unternehmensnetzwerken aufgrund von Sicherheitsbestimmungen ggf. nicht möglich. In Abb. 6 ist der Weg über das Proxy-Repository in violett und grau ausgezeichnet. Intern können Packages weiterhin auf konventionelle Art und Weise bezogen werden.

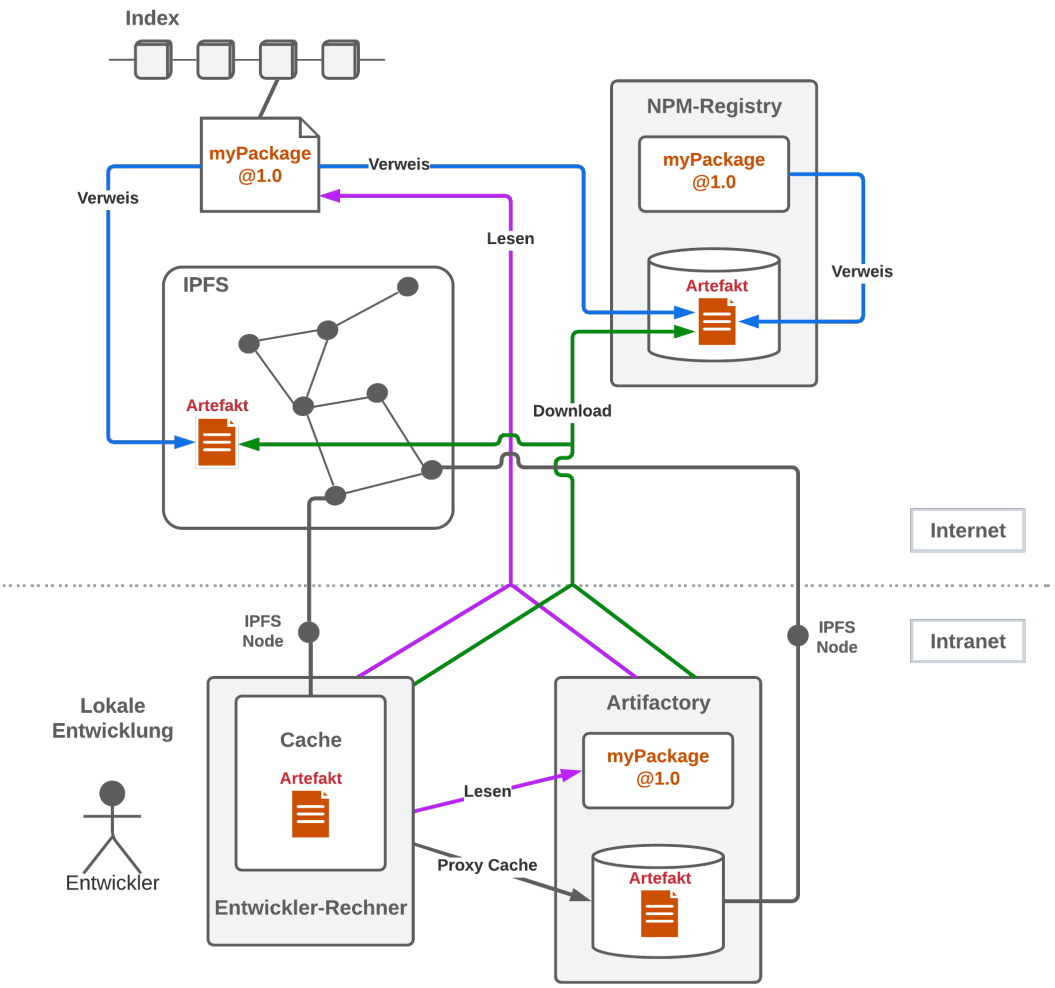


Abbildung 6: Release Lookup und Artefakt-Download direkt durch Entwickler und Proxy-Repository

5.3.3. Sicherheit und Sensibilisierung

Aufgrund der verifizierbaren Daten des dezentralen Indexes gewinnt der Entwickler die Sicherheit, dass die bezogenen Dateien in dem Zustand wie vom Autor beabsichtigt erhalten wurden. Verwendet er weiterhin die Informationen aus dem Build Verification Network, ist damit auch die Authentizität der Artefakte gegeben, ohne dass er sie selbst überprüfen müsste. Continuous Integration Systeme könnten auch so konfiguriert werden, dass sie diese Beweislast von den einzelnen Entwicklern übernehmen. Sie könnten den Index befragen oder selbst die Builds von Dependencies ausführen.

Das `npm` Werkzeug nutzt die GitHub Advisory Database[124], um Entwickler auf Schwachstellen in den verwendeten Dependencies hinzuweisen. Diese Datenbank wird seitens GitHub gepflegt und aggregiert außerdem Meldungen von Schwachstellen aus weiteren Datenbanken[125]. Meldungen über Bugs und weitere Probleme aus einem dezentralen Review Network könnten die gegebene Liste erweitern, da sich Reviews nicht nur auf Sicherheitsmängel beziehen.

5.4. Endanwender

5.4.1. Software Bill of Materials

Ähnlich wie die Nutzung von externen CDN-Plattformen wie cdnjs zum Hosting von Software Dependencies einer Anwendung könnte auch ein dezentrales Software Repository direkt bis zum Endanwender eingesetzt werden. Anstatt Artefakte von den Servern einer Applikation zu beziehen, könnten sie diese auch mithilfe des Repositorys herunterladen. Dadurch profitieren die Nutzer durch erhöhte Sicherheit und ggf. schnellere sowie sicherere Downloads. Auf der anderen Seite gewinnt die Datenhaltung des dRepos weitere Nodes, was ebenfalls die Sicherheit und Verfügbarkeit der Artefakte erhöht.

Moderne Applikationen wie in Kapitel 5.1 beschrieben nutzen verschiedene Techniken zur Optimierung. Es wird versucht, die Codebasis in ihrer Speichergröße zu minimieren und gleichzeitig eine effiziente Datenübertragung zu ermöglichen. Auch unter Verwendung neuer Protokolle wie HTTP2/3 ist die Übermittlung weniger, großer Dateien weiterhin wirtschaftlicher. Während dieser Optimierungsprozesse werden die Dependencies und der Applikationscode transformiert und zusammengefasst. Der Originalquellcode ist in den Build-Artefakten nur schwierig wieder zu identifizieren.

Um das dezentrale Repository ausnutzen zu können, darf eine solche Maskierung der Dependencies nicht stattfinden. Die Applikation muss dem Browser des Endanwenders präzise mitteilen, welche Artefakte verwendet werden. Daraufhin kann der Nutzer die Dateien aus beliebigen Quellen herunterladen. Dies bedeutet, dass das

[124] Mehr Informationen siehe: https://github.com/advisories und https://docs.github.com/en/code-security/supply-chain-security/managing-vulnerabilities-in-your-projects-dependencies/browsing-security-vulnerabilities-in-the-github-advisory-database.
[125] Vgl. Thomson 2021.

Build und die Optimierungsprozesse einer Applikation so angepasst werden müssen, dass Dependencies in ihrer Originalform, wie sie in Repositories zu finden sind, erhalten bleiben und verwendet werden können.

Eine Art Software Bill of Materials (SBOM) könnte während des Builds erstellt werden[126]. Sie enthält eine detaillierte Übersicht über alle Abhängigkeiten einer Applikation und wird dem Browser an definierter Stelle zur Verfügung gestellt. Dies gibt dem Browser die Möglichkeit, alle notwendigen Dependencies so früh wie möglich laden zu können. Ohne eine solche Liste müsste der Browser während des Parsings bzw. der Programmausführung bei Bedarf jede einzelne Abhängigkeit herunterladen. Dies würde die Ausführung der Anwendung deutlich verlangsamen.

In Abb. 7 wird der Unterschied zwischen dem Kompilieren mehrere Dependencies zu einem Applikationsartefakt (oben) und der Erzeugung einer SBOM (unten) verdeutlicht. Die SBOM verweist auf dieselben Artefakte, die der Entwickler einer Anwendung verwendet. Mithilfe dieser Verweise kann ein Endnutzer anschließend die Artefakte aus dem dezentralen Repository beziehen.

Das genaue Format einer SBOM bleibt zu definieren. Sie muss mindestens die Koordinaten einer Abhängigkeit im dRepo bezeichnen, sodass der Browser des Anwenders die möglichen Lokationen der Dateien erfahren kann. Die Angabe zusätzlicher Quellen durch Applikationsentwickler, beispielsweise auf den Servern der Applikation, ist ggf. sinnvoll, um den Ladeprozess in einigen Fällen zu beschleunigen. An dieser Stelle können moderne Protokolle wie HTTP2/3 ausgenutzt werden, um eine Menge an Dateien in einer Verbindung zu transportieren. Der Einsatz des Server-Push-Verfahrens ist ebenfalls denkbar[127].

Es bleibt in der Verantwortung des Browsers, die bezogenen Artefakte unabhängig von der Quelle auf ihre Authentizität zu prüfen. Die notwendigen Daten dafür müssen im Endeffekt aus dem dRepo stammen.

5.4.2. Netzwerkeffekte

Da Endanwender auf die Ressourcen des dezentralen Repositorys zugreifen, sind sie auch Teil dessen Netzwerks. Dies bezieht sich auf die Verfügbarkeit der Nutzdaten sowie die Verifikation von Index-Inhalten.

[126] Vgl. National Telecommunications and Information Administration 2020.
[127] Vgl. Belshe et al. 2015, S. 60.

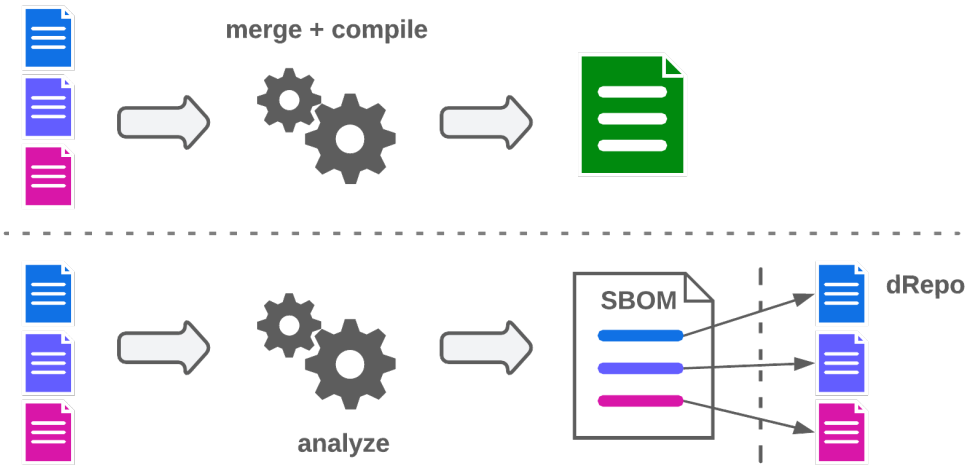


Abbildung 7: Artefakt-Bundling und SBOM-Verweise

Wenn der Browser des Endanwenders die Artefakte, die in der SBOM einer Applikation definiert sind, herunterlädt, muss er deren Authentizität prüfen. Der Browser ist nur soweit dazu in der Lage, dass er die verfügbaren Prüfsummen aus dem Repository mit dem Artefakt, das er bezogen hat, vergleicht. Stimmen diese nicht überein, muss das Artefakt aus einer anderen Quelle geladen werden. Allerdings muss diese Information, dass eine invalide Quelle existiert, in das Netzwerk das Repositorys zurückgemeldet werden. Entsprechende Infrastrukturen müssten definiert werden.

Die bevorzugte Quelle von Nutzdaten des dRepos sind dezentrale Datenspeicher, vornehmlich das P2P-Netzwerk IPFS. Aktuelle Browser können spätestens über Extensions als Node an dem Netzwerk teilhaben. Dateien, die der Browser mittels IPFS bezieht, werden automatisch weiter im Netzwerk verteilt. Die Dependencies einer Applikation stehen damit ausfallsicherer zur Verfügung, da eine größere Menge an Teilnehmern die entsprechenden Dateien teilen.

Weiterhin nutzen viele Applikationen dieselben Abhängigkeiten wie beispielsweise Frameworks und Bibliotheken wie Angular oder React. In derzeitigen Setups müssen Endanwender während der Nutzung unterschiedlicher Anwendungen implizit dieselben Artefakte mehrfach herunterladen. Dadurch, dass diese Dependencies mithilfe des dRepo und SBOMs eindeutig adressierbar werden, können bereits cached Artefakte über mehrere Applikationen hinweg wiederverwendet werden. Dies kann dem Nutzer Vorteile bei Ladezeiten und Speicherauslastung bringen.

Durch die Nutzung unterschiedlicher Anwendungen entsteht beim Endanwender eine relativ große Ansammlung an identifizierbaren Artefakten. Diese können vor allem mit benachbarten Nodes geteilt werden, sodass sie mit geringerer Latenz bezogen werden können. In größeren Netzwerken, innerhalb von Organisation oder ISP-Netzen, sind ggf. »Mirror«-Nodes sinnvoll, welche konfiguriert sind, um eine größere Anzahl an Artefakten des dRepo zu beherbergen. Auch hier könnten Anwender von geringen Latenzen profitieren.

Die Dependencies einer Applikation können aus unterschiedlichen Quellen bezogen werden. Teilweise ist der Download aus zentralisierten Quellen schneller. Auch wenn die Dateien nicht über IPFS bezogen wurden, sollten dennoch alle Artefakte, die in einer SBOM angefordert wurden, über den IPFS Node geteilt werden. Dies hat den Effekt, dass jeder Browser als Brücke zwischen dem P2P-Netzwerk und zentraliersten Datenspeichern fungiert. Dies stärkt die Datenverfügbarkeit im Netzwerk, insbesondere für selten verwendete Artefakte.

Die genannten Effekte sind in Abb. 8 in einem Szenario veranschaulicht. Hier möchte Bob die Anwendungen App1 und App2 verwenden. In seinem Cache befindet sich bereits eines der Artefakte, das für beide Apps benötigt wird. Sein Browser entscheidet, das violette direkt von der Anwendung zu beziehen. Die magenta Abhängigkeit wird dagegen von Alice via IPFS heruntergeladen, die das Artefakt zuvor von App2 bezogen hatte.

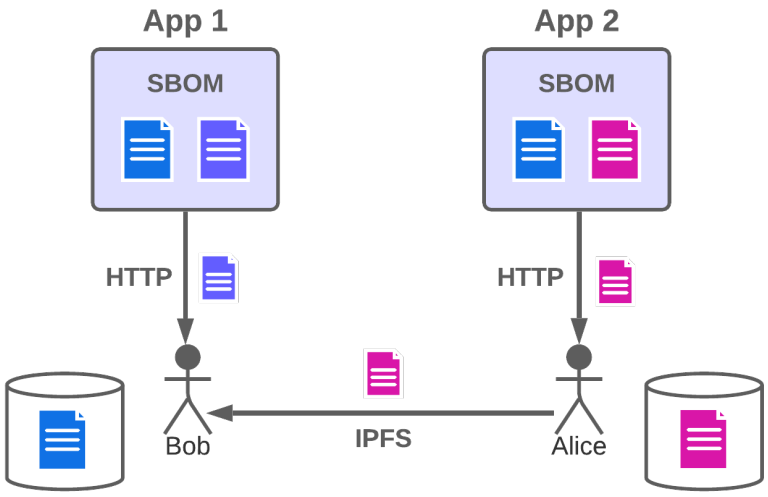


Abbildung 8: Netzwerkeffekte beim Download von Artefakten

5.4.3. Verifikation

Der Browser eines Anwenders muss alle Dependencies, die er mittels SBOM und dRepo bezieht, auf ihre Authentizität überprüfen. Dazu werden Prüfsummen verwendet. Gleichzeitig bietet das erweiterte dRepo Informationen über den Zustand des Builds sowie des Quellcodes. Diese Erkenntnisse sollten dem Endanwender ebenfalls präsentiert werden. Sie sind ein Hinweis auf die Qualität und Sicherheit einer Applikation. Etwaige Probleme sollten bemängelt werden, ähnlich wie die Nutzung unsicherer Verbindungen im Browser. Entsprechend würde es im Interesse von Applikationsentwicklern liegen, dass die genutzten Dependencies als sicher und vertrauenswürdig eingestuft werden.

Im weiteren Sinne sollte der Applikationscode einer Anwendung ebenfalls wie Dependencies im dRepo veröffentlicht werden. Abhängigkeiten und Code, der nicht in der SBOM und im dRepo, definiert sind, sind tendenziell als unsicher einzustufen. Sie können während der Übermittlung manipuliert werden oder generell Schadcode enthalten. Solche Sachverhalte sind für Endanwender nicht ersichtlich, da sie die Authentizität und Validität des Codes nicht prüfen können, so wie es derzeit der Fall ist. Der Code ist nicht öffentlich reviewed worden und eine korrekte Übertragung kann nicht nachgewiesen werden, da der Code unbekannt ist. Ebenso würde die Applikation von der Ausfallsicherheit des dRepo profitieren, wenn der gesamte Code dezentral geteilt wird.

5.4.4. Customization

Eine komplette Aufschlüsselung der Applikation mittels SBOM gibt dem Endanwender eine Übersicht über alle verwendeten Softwarekomponenten. An dieser Stelle könnte der Nutzer die Entscheidung treffen, bestimmte Programmteile nicht ausführen zu lassen. Dies kann durch die Bearbeitung der Abhängigkeiten geschehen. Die Inhalte der SBOM könnten mit eigenen Definitionen überladen werden. Eine unerwünschte Dependency könnte gegen eine Dummy- oder angepasste Implementation ausgetauscht werden. Auf diese Weise bleibt die Applikation weiterhin ausführbar, während einige Programmteile funktionslos werden. Beispielsweise könnten Tracking-Komponenten blockiert werden, bevor sie Daten sammeln können. Dies besitzt gegenüber derzeitigen Adblockern den entscheidenden Vorteil, dass solcher Code definitiv nicht aufgeführt wird und auch für die Zukunft blockiert bleibt. Heu-

tige Adblocker und deren Blockierungsdefinitionen müssen stets an neue Konstrukte angepasst werden, um diese sperren zu können[128].

Neben dem Blockieren von Code ist auch die Nutzung alternativer Implementationen auf demselben Weg denkbar. So könnten Erweiterungen durch Dritte direkt in den Applikationscode integriert und müssten nicht als Extensions im Browser hinzugefügt werden. Ebenso könnten auch ältere Implementationen eines Frontends weiterhin genutzt werden.

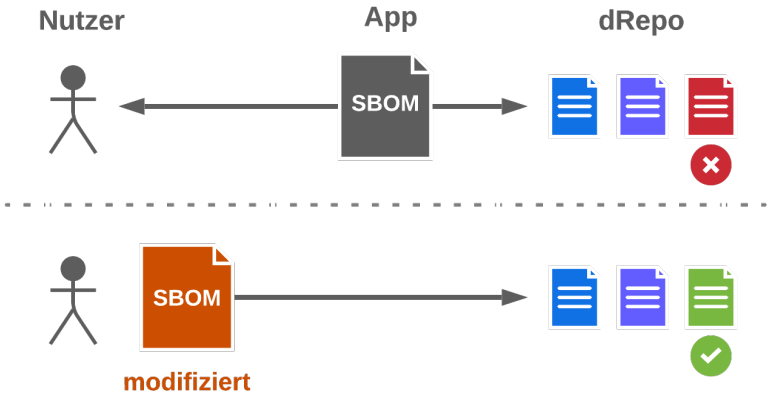


Abbildung 9: Artefakte blockieren und ersetzen

Abb. 9 veranschaulicht dieses Konzept nochmals. Eine Applikation liefert dem Anwender ihre SBOM, welche er analysiert und anschließend modifiziert, um eine Abhängigkeit zu ersetzen. Mit der angepassten SBOM kann er die Applikation zusammensetzen und ausführen.

5.4.5. Applikations- und Browseranpassungen

Ein Problem stellen unterschiedliche Modul- und Import-Systeme in JavaScript Bibliotheken dar. So kann zum Beispiel die CommonJS `require()`-Funktion, die häufig in Node.js Applikationen zum Einsatz kommt, von Browsern zunächst nicht nativ ausgewertet werden[129]. Diese würden während des Builds einer Anwendung mithilfe von Webpack und Transformatoren wie Babel Browser-kompatibel umgewandelt werden[130]. Da die Artefakte einer Dependency allerdings nicht verändert werden sollen, stellt die applikationsentwicklerseitige Transformation den letzten

[128] Vgl. Storey et al. 2017, S. 1-4, S. 8-12.
[129] Vgl. Elliott 2014, S. 82-83, S. 87.
[130] Vgl. Webpack 2020.

Ausweg dar. Sie wäre dadurch inhaltlich nicht mehr durch das dRepo abgesichert. Um dies zu vermeiden, müssen Wege gefunden werden, Kompatibilität zwischen den unterschiedlichen JavaScript Plattformen und Modulsystemen herzustellen. Ein Weg wäre die Bereitstellung der Build-Artefakte von Bibliotheken in unterschiedlichen Modulsystemen. Diese könnten ebenfalls dem dRepo hinzugefügt werden.

Da mittlerweile alle gängigen, modernen Browser ECMAScript Modules (ESM) nativ unterstützten, ist es ggf. sinnvoll sich auf dieses System für die Frontend-Entwicklung zu konzentrieren[131]. Dies bringt allerdings das Problem mit sich, dass solche Modulimporte stets über URLs und Pfade definiert werden. Typischerweise werden in der Entwicklung hingegen Bare Imports[132] verwendet, die sich im Endeffekt auf die Namen von Packages, beispielsweise aus der NPM Registry, beziehen[133]. Dies ermöglicht die einheitliche Adressierung von Modulen. Derzeit werden zur Build Time diese Import-Definitionen jeweils in festgelegte Zielformate übersetzt.

Eine Möglichkeit, um diese Transformationen zu vermeiden, ist die Definition vom *Import Maps*. Sie sind allerdings noch nicht als Standard verabschiedet worden und sind daher nur teilweise in Browsern verfügbar. Durch Import Maps können Bare Imports zur Laufzeit im Browser auf definierte Pfade abgebildet werden. Dadurch kann eine einheitliche Benennung von Softwarekomponenten genutzt werden. Diese Funktion könnte um notwendige SBOM Eigenschaften erweitert werden, sodass multiple Pfade oder Repository-Verweise genutzt werden könnten. Dies ist insbesondere dann relevant, wenn auf externe URLs verwiesen wird[134,135].

Insgesamt müssen solche Features mit dem Gedanken einer SBOM und einem dezentralen Software Repository abgestimmt werden, sodass notwendige Informationen nicht mehrfach oder verstreut in einem Projekt abgelegt werden müssen. Dies würde den Gedanken einer SBOM, alle Softwareabhängigkeiten an einem Ort zu definieren, untergraben.

Schließlich muss der Browser in die Lage versetzt werden, die Dependencies, die eine Applikation verlangt, mithilfe des dRepos und zusätzlichen Verweisen seitens der Anwendung schnellstmöglich zu beziehen. Hierfür müssen Konventionen und Prozesse definiert werden, wie und wann der Browser diese Informationen verarbeitet und wie sich diese in den Life Cycle einer Applikation einfügen. Weiterhin müssen

[131] Vgl. Rauschmayer 2015, S. 36-38.
[132] Bare Imports beschreiben Import-Ausdrücke, die anstatt einer URL den Namen eines Packages verwenden. Problematisch an dieser Art der Adressierung ist, dass diese Namen aufgelöst werden müssen, um die eigentlichen Artefakte finden zu können.
[133] Vgl. Node.js 2022.
[134] Vgl. Denicola 2021.
[135] Vgl. Import Maps 2021.

die Artefakte im Anschluss der Applikation zugereicht werden. Dies könnte problematisch sein, da die bezogenen Dateien nicht in demselben Dateisystem wie ggf. die restliche Applikation liegen. Bei HTTP Ressourcen könnte an dieser Stelle CORS ein Problem sein. Allerdings unterliegen Artefakte aus einem dezentralen Repository anderen Sicherheitseigenschaften.

6. Kritischer Ausblick

6.1. Technischer Reifegrad und Akzeptanz

Der Kern des dezentralen Repositorys basiert auf Technologien, die seit bereits einigen Jahren im produktiven Einsatz sind und den Wert digitaler Assets sichern. Im Rahmen von Non-Fungible Tokens wird die Kombination aus Smart Contract und IPFS verwendet, um Metadaten sowie Payloads zu speichern, die zu groß und damit zu teuer sind, um sie on-chain zu sichern. Aufgrund von Content Addressing und der daraus folgenden Immutability bietet sich IPFS als sicherer, ergänzender Speicher zur Blockchain an.

Das Konzept des dezentralen Repository baut auf demselben Prinzip auf und muss daher auch die entsprechenden Best Practices umsetzen. Hier ist vornehmlich die Unveränderlichkeit der Verknüpfung der beiden Systeme zu nennen. Dies bedeutet, dass ein Objekt des Repository Indexes nicht den Verweis auf ein bestimmtes Objekt in IPFS entfernen oder ändern kann. Dies ist durch die append-only Eigenschaft der Index-Implementation gegeben.

Es handelt sich derzeit jedoch um ein Nischenkonstrukt, welches neben NFTs auch Verwendung in DAO Governance findet. Der inverse Aufbau wird beim Hosting dezentraler dApp Interfaces angewendet.

Blockchain- und Kryptothemen werden allerdings immer wieder polarisiert diskutiert. Energieverbrauch, Bereicherung und Betrugsfälle sind Beispiele solcher Themen. Gleichzeitig werden Dezentralisierungsprobleme und der Bubble-Zustand des NFT-Markts kritisiert. Durch die Verwendung von Krypto-Technologien könnte die Einführung des dRepo ebenfalls auf Kritik und geringe Akzeptanz bei Entwicklern und Endanwendern stoßen. Dies steht dem wachsenden Bewusstsein für die Bedeutung der Software Supply Chain entgegen.

6.2. Sicherheit

Um das dezentrale Repository sicher zu gestalten, wurde es im Kern simpel konzipiert und enthält keine komplexen Features. Auch die Beispielimplementation, siehe Anhang B, zeichnet sich durch Einfachheit aus. Dennoch muss der Contract, bevor er deployed werden kann, durch Experten reviewed werden, da eine Änderung im Nachhinein nicht möglich ist. Weiterhin muss die Weiterentwicklung des Systems bedacht werden. Bevor ein Repository tatsächlich deployed werden kann, sollte eine Community Governance eingerichtet werden. Diese sollte letztlich über Upgradepfade bzgl. Immutability oder Upgradability in Hinsicht auf die Ziele des Systems entscheiden. Ebenso wäre sie für die Verabschiedung von Konventionen verantwortlich, welche auch das Verweissystem bestimmen. Allerdings darf die Community Governance nicht mit administrativen Rechten ausgestattet werden, sondern sollte schließlich eine unterstützende Rolle einnehmen.

Das Verweissystem und insgesamt das gesamte Repository müssen auf dem aktuellsten Technologiestand gehalten werden, um potentielle Sicherheitslücken zu vermeiden. Diese können insbesondere durch die Nutzung schwacher Hash-Funktionen auftreten, wenn die Computertechnik voranschreitet. Daher sollten auch alte Artefakte durch aktuelle Prüfsummen aufgefrischt werden. Weiterhin muss die Sicherheit und Dezentralisierung der zugrundeliegenden Blockchain beachtet werden. Es ist die Hoffnung, dass durch die Teilnahme an einem größeren Ökosystem der Repository Index Netzwerkeffekte ausnutzen kann. Sollte dies nicht zutreffen und der Index gefährdet werden, müssen Maßnahmen ergriffen werden. Die Schaffung expliziter Anreize wäre eine Möglichkeit.

Neben der Gefährdung der Chain selbst, sind auch die avisierten Zugangspunkte zum Repository Index durch die Community Governance zu überwachen. Diese sind derzeit die praktikabelsten Verbindungen zu den Indexinhalten für die meisten Anwender und stellen damit geeignete Angriffspunkte dar. Mit der wachsenden Adaption von Blockchain-Applikationen sollten im Sinne der Dezentralisierung andere Zugangsmethoden präferiert werden.

6.3. Datenverfügbarkeit

Um die Verfügbarkeit der Nutzdaten des dezentralen Repositorys sicherzustellen, sind IPFS, aktuelle Peer-to-Peer-Netzwerke und auch die Kombination verschiedener Quellen, unter anderem zentralisierte Datenspeicher, nicht ausreichend. Storage

Dienste wie Filecoin oder Arweave versprechen dauerhafte Verfügbarkeit. Allerdings können nicht beliebige Daten dort gesichert werden[136,137,138].

Im Rahmen der Zensurresistenz sind nur P2P-Netzwerke anwendbar. Diese kommunizieren jedoch einerseits offen, wodurch die Identität von Teilnehmern sichtbar werden könnte, und andererseits existieren keine Garantien, dass Daten nicht verloren gehen können, indem keine Seeds mehr existieren.

Die Nutzung dezentraler VPNs könnte das Problem der Anonymität und damit der Zensurresistenz adressieren. Anders als bei zentralen VPN-Anbietern müssen Anwender diesen nicht vertrauen. Weiterhin existiert keine zentrale Entität, die durch Dritte identifiziert und manipuliert werden könnte. Daten und Verbindungen werden durch Teilnehmer des dVPN-Netzwerks geleitet.

Ähnliche Verschleierungen der Kommunikationswege könnte auch auf Protokoll- und Applikationsebene erreicht werden. Systeme wie IPFS und BitTorrent verteilen nur die Daten, die durch den Nutzer zuvor angefragt wurden. Um die Zensurresistenz und gleichzeitig die globale Datensicherheit zu verstärken, ist ein automatisches Weiterleiten und Verteilen von Inhalten im Netzwerk vorstellbar. So könnten Daten Chunk-weise gleichmäßig über die Teilnehmer verteilt werden, während die Inhalte teilweise auch zufällig nur durch Clients durchgeleitet werden. Damit ist intransparent, wo Daten tatsächlich vorgehalten werden. Zwischengespeicherte Daten könnten ebenfalls für den Nutzer verschlüsselt werden, sodass im Sinne der Verantwortung ein Zustand *glaubhafter Abstreitbarkeit (Plausible Deniability)*[139] entsteht. Durch die zufällige Distribution von Datenteilen kann deren Redundanz erhöht werden, ohne dass einzelne Teilnehmer mit großen Datenmengen überfordert werden.

Solche Konzepte stellen Anwender jedoch vor eine größere Einstiegsschwelle, da sie sich einerseits mit bisherigen Nischentechnologien beschäftigen müssen und das System andererseits einen gewissen Darknet-Charakter besitzt, was ebenso die Akzeptanz verringern kann.

6.4. Erweiterung des Repositorys

Das Kern-Repository stellt nur die Basisfunktionalitäten zur Verfügung. Um das System komfortabel nutzen zu können, sind Erweiterungen notwendig. Ein Review-

[136] Vgl. S. A. Williams et al. 2019, S. 13.
[137] Vgl. S. Williams und Jones 2018, S. 7.
[138] Vgl. Filecoin Spec 2021.
[139] Vgl. Office of the Historian, Department of State 1948.

und ein Build-Verifikationssystem werden als essentielle Expansionen des Repositorys angesehen. Diese und andere erweiternde Systeme müssen allerdings durch Dritte bzw. die Community erstellt werden, da sie weit komplexer ausfallen werden.

Bei Builds und Reviews handelt es sich um ressourcenintensive Aufgaben, die nicht freiwillig bzw. ohne Gegenleistung ausgeführt werden. Daher müssen diese mit Anreizen verbunden werden, die unabhängige Teilnehmer dazu bewegen, ihre Ressourcen zur Verfügung zu stellen. Gleichzeitig wächst das Bewusstsein für die Verwundbarkeit der Software Supply Chain in der Öffentlichkeit, insbesondere in größeren Unternehmen sowie Regierungen. Daher könnte angenommen werden, dass solche Initiativen monetäre Unterstützung finden würden.

Es ist zu bedenken, dass Software Dependencies einen Graphen darstellen. Daher müssen auch die Abhängigkeiten von Abhängigkeiten auf den Prüfstand gestellt werden, vielleicht mehr als Top-level Dependencies wie Frameworks. Konzepte wie Quadratic Funding könnten für eine angemessene Ressourcenverteilung sorgen. In jedem Fall sind solche Systeme sorgfältig zu planen, sodass die verfügbaren Mittel sinnvoll genutzt werden.

Ein allgemein sinnvoller Einsatzzweck wäre die Einrichtung von Faucets[140]. Diese bieten eine geringe Menge kostenfreier Ressourcen für einen Einstieg in das Ökosystem an, um beispielsweise ein erstes Software Release publizieren zu können. Sie könnten durch Mittel aus dem erweiterten Repository finanziert werden.

6.5. Rechtliche Aspekte

Das dezentrale Repository steht im Gegensatz zur Regulierung des Internets. Durch seine Offenheit und Endgültigkeit aufgrund von Immutability sind rechtliche Datenschutzvorgaben explizit nicht einhaltbar. Es können personenbezogene oder sensible Daten wie Passwörter, die durch Anwender leaked werden, nicht gelöscht werden. Ebenso können Autoren ihre Softwarepakete, welche ihr Eigentum sind, nicht zurückziehen, anders als es derzeit in Repositories möglich ist. Sie verlieren somit die Kontrolle über ihre Software[141].

Weiterhin existiert keine Entität, die direkte Verantwortung für das System übernimmt. Sämtliche administrative Interaktion mit dem Repository sollte unbedingt

[140] Faucets sind Applikationen, die geringe Mengen an Kryptowährungen an meist leere Wallets verschenken, sodass diese minimale Transaktionen ausführen können. Sie unterstützen beim Einstieg in das System.

[141] Vgl. Bundesamt für Sicherheit in der Informationstechnik 2019, S. 57-60.

anonym stattfinden. Eine etwaige Community Governance sollte sich ebenfalls durch Anonymität auszeichnen. Da auch empfohlen wird, dass jede Interaktion mit dem Index und den Storage Systemen durch die Teilnehmer mithilfe Privacy Preserving Tools stattfindet, ist es schwierig kryptographisch Verantwortung nachzuweisen.

Aus diesen Gegebenheiten kann sich das Spamming von unerwünschten, ggf. rechtswidrigen, Inhalten als problematisch erweisen. In der konzipierten Implementation kann dies nicht verhindert werden. Außerdem sind zumindest die Metainformationen im Index für immer verfügbar. Im aktuellen Konzept wird lediglich die Spamming-Komponente mittels Gas Fees mitigiert. Dies könnte sich aufgrund von Preisgestaltung allerdings als wirkungslos erweisen.

Das Publizieren unerwünschter Inhalte lässt sich im aktuellen Konzept nicht unterbinden, insbesondere wenn die entsprechenden Teilnehmer ihre Privatsphäre schützen. Es ist an dieser Stelle auch fraglich, ob dies überhaupt verhindert werden sollte, da dies im direkten Konflikt mit der Anforderung auf Zensursicherheit steht. Im Rahmen des erweiterten Repositorys sollten solche Inhalte allerdings markiert werden. Dies würde jedoch auch Aufmerksamkeit auf diese ziehen, was die Sinnhaftigkeit des Mechanismus wiederum untergräbt.

7. Fazit

Das Ziel dieser Arbeit war es, ein Konzept für ein universelles, dezentrales Software Repository zu erarbeiten, welches existierende Repositories weitgehend ersetzen könnte. Initiator für dieses Unterfangen ist die wachsende Bedeutung und Beachtung der Software Supply Chain durch Industrie und Regierungen. Software Repositories nehmen eine essentielle Rolle dabei ein und stellen gleichzeitig einen Single Point of Failure für das gesamte Konstrukt dar. Sie sind aufgrund ihrer Zentralisierung verwundbar. Repositories können durch technische Angriffe lahmgelegt, Inhalte aufgrund rechtlicher Dispute für alle Nutzer gesperrt oder durch böswillige Akteure manipuliert werden. Solche Eingriffe beeinflussen die Softwareentwicklung und Verwendung von Software weltweit und müssen daher minimiert werden.

Dezentralisierung ist eine Möglichkeit, dies zu erreichen. Technologien und Konzepte in diesem Bereich konnten in den letzten Jahren vermehrt erfolgreich angewendet werden. Sie ermöglichen weitgehend gesicherte, manipulationsfreie Applikationen, die für Jedermann frei zugänglich sind. Software Repositories könnten von diesen und weiteren Eigenschaften profitieren, um so die Software Supply Chain ebenfalls zu stärken.

Um ein Konzept für ein dezentrales Software Repository erarbeiten zu können, war zunächst eine Diskussion des Themas Dezentralisierung notwendig. In diesem Zuge wurden zentralisierte und dezentralisierte Systeme gegenübergestellt, sodass die Eigenschaften der Dezentralisierung herausgestellt werden konnten. Trustless, permissionless und immutable sind drei der Kernphilosophien, die für dezentrale Softwaresysteme von besonderer Bedeutung sind und daher in einem neuen Repository aufgegriffen werden müssen.

Da es die Aufgabe war, auch ein universell einsetzbares Repository zu konzipieren, mussten die generellen Merkmale von Software Repositories identifiziert werden. Dazu wurden eine Reihe von Software Repositories sowie verteilte Content Repository Systeme analysiert und verglichen. Es konnten Gemeinsamkeiten und Unterschiede

festgestellt sowie wichtige Umsetzungshinweise für ein dezentrales Repository ausgemacht werden. Auf diese Weise konnte ein hinreichendes Feature-Set für ein Software Repository aufgestellt werden.

Aus den Erkenntnissen der Repositiory-Analyse und den Dezentralisierungseigenschaften wurden Anforderungen erhoben, die als Basis für das Konzept dienten. Im Rahmen der Konzeption wurde allerdings die Bedeutung des Themas Datensicherheit klar. Die untersuchten Repositories verwenden Prüfsummen und Signaturen, um Artefakte, die aus einem Repository bezogen wurden, verifizieren zu können. Die Bedeutung der genutzten Verfahren ist allerdings in dem Rahmen nicht hinreichend für den Nachweis der Authentizität der Inhalte. Die Anwendung von Reproducible Builds könnte dies jedoch erfüllen. Daher werden diese im Konzept essentiell vorausgesetzt.

Das Konzept selbst unterscheidet sich in zwei Bereiche: Das Kern-Repository und das erweiterte Repository. Der Kern besteht lediglich aus zwei Komponenten, dem Index und dem Storage, Datenspeicher. Die Index-Komponente ist ein Smart Contract, der auf einer permissionless Blockchain deployed werden soll. Er ist für die Organisation des Repositorys verantwortlich. Er verwaltet die minimalen Benutzerrechte und sorgt dafür, dass die Metadaten von Softwarepaketen immutable sind. Die Hauptaufgabe besteht jedoch darin, dass von Release Informationen einzelner Packages auf die Nutzdaten verwiesen wird. Der Storage des Konzepts ist keine dedizierte Softwarekomponente, sondern stellt eine Sammlung aus möglichen Datenspeichern dar. Diese halten die Nutzdaten, die Build-Artefakte, Quellcode, Dokumentation usw. eines Software Releases. Der Index verweist auf die Lokation dieser Daten und stellt gleichzeitig Metadaten zur Verifikation dieser Inhalte bereit. Hauptsächlich sollen dezentrale Datensysteme eingesetzt werden, primär das Peer-to-Peer-Netzwerk IPFS. Diese zeichnen sich durch ihre Zensurresistenz aus. Ein Mix aus möglichen Bezugspunkten sorgt für hohe Verfügbarkeit und Interoperabilität.

Im erweiterten Repository werden zusätzliche Komponenten definiert, die das System in Praktikabilität und Sicherheit unterstützen. So ist beispielsweise die Verifikation von Builds mithilfe Reproducible Builds in einem dezentralen Netzwerk notwendig, um es Anwendern zu ersparen, die Authentizität jeder Dependency mithilfe eines Builds zu beweisen. Diese kann über ein solches Netzwerk mit hoher Wahrscheinlichkeit nachgewiesen werden. Ähnliche Systeme sollten zur Evaluierung des Quellcodes zum Einsatz kommen.

Anschließend wurde das Konzept modellhaft an der NPM Registry und dessen Frontend-Development-Ökosystem überprüft. Neben einer möglichen Migration müs-

sen ebenfalls Entwicklungswerkzeuge angepasst werden, sodass diese aus dem neuen Repository ihre Daten beziehen können. Dies bedeutet im einfachsten Fall, dass die Repository-Schnittstelle angepasst werden muss, sodass sie aus dem Smart Contract Daten lesen können. Build-Artefakte könnten weiterhin aus der NPM Registry heruntergeladen werden. Mit dem Einsatz von beispielsweise IPFS würden jedoch Netzwerkeffekte und höhere Sicherheit den Entwicklern zugutekommen. Die Anpassung genereller Workflows ist nicht notwendig, da das dezentrale Repository dementsprechend konzipiert wurde.

Größere Effekte können erzielt werden, wenn das dezentrale Repository bis zum Endanwender zum Einsatz kommt. Würden Applikationen ein Inhaltsverzeichnis ihrer Softwarekomponenten in Form einer Software Bill of Materials bereitstellen, könnten Anwender diese aus dem dezentralen Repository beziehen und deren Authentizität prüfen. Zudem ergeben sich Caching- und weitere Netzwerkeffekte, die für schnellere Datenübertragung und höhere Verfügbarkeit sorgen könnten, während für den Betrieb einer Anwendung weniger Aufwand notwendig werden würde. Nutzer könnten ebenfalls diese Informationen benutzen, um unerwünschte oder unsichere Software zu identifizieren und ggf. zu blockieren. Basierend auf diesen Anwendungsfällen könnte ein transparenteres Internet für Entwickler, Anwender, Unternehmen und Regierungen entstehen.

Wie die naive Implementation des Repository-Index zeigt, ist das Kern-Repository mit existierenden Technologien schnell umzusetzen. Hingegen besteht das erweiterte Repository aus weit komplexeren Komponenten, die noch nicht tiefgreifend erforscht wurden. Auch ist noch nicht klar, ob diese tatsächlich alle Hoffnungen an ein dezentrales Repository erfüllen können. Hier sind ggf. noch nicht alle notwendigen Aspekte in Gänze erfasst worden. Es wird jedoch davon ausgegangen, dass sich in der Software Community die entsprechenden Komponenten bilden werden.

Ein weiteres Problem entsteht auch aus dem Fakt, dass die avisierten Techniken wie Blockchains und Peer-to-Peer-Netzwerke immer wieder kontrovers diskutiert werden, insbesondere aufgrund von Umweltaspekten durch Proof-of-Work Blockchains und durch den NFT-Hype von 2021. Dies könnte die Akzeptanz bei Entwicklern und in der Öffentlichkeit beeinflussen.

In demselben Zuge sind auch rechtliche und ethische Fragen noch zu beantworten. Das erstellte Konzept stellt die Dezentralisierungsanforderungen, wie Zensurrestistenz und Immutability, über die Möglichkeit unerwünschte Inhalte entfernen zu können. Dies bedeutet, dass in einem solchen System widerrechtliche oder problematische Inhalte publiziert und effektiv nie mehr entfernt werden können.

Im Zusammenhang der Zensurresistenz muss das Thema Sicherheit für die Teilnehmer des Systems betont werden. Dezentrale Systeme sind offen gestaltet und geben ihren Nutzern häufig nur Pseudoanonymität. Damit das System tatsächlich zensurresistent und darüber hinaus hochverfügbar werden kann, müssen durch die Teilnehmer Techniken eingesetzt werden, um Anonymität herzustellen

Abschließend ist zu sagen, dass das Repository so minimalistisch konzipiert wurde, dass es sich für eine Vielzahl von Anwendungsfällen neben Software Repositories eignet. Generell kann es, ggf. unter wenigen Anpassungen, für die sichere Verwahrung und freie Verteilung digitaler Informationen eingesetzt werden.

Das heutige Internet basiert, ebenso wie unsere Gesellschaft, auf *Vertrauen*. Es wird darauf vertraut, dass demokratische Regierungen im Sinne des Volks handeln, Certificate Authorities nicht korrupt sind und zentralisierte Software Repositories ihre Inhalte nicht manipulieren. Allerdings ist Vertrauen nicht mit *absoluter Gewissheit* gleichzusetzen. Ein dezentrales Repository, wie es hier konzipiert wurde, kann uns dem ein Stückchen näher bringen.

Literaturverzeichnis

Abadi, J. & Brunnermeier, M. (2018). *Blockchain economics* (Techn. Ber.). National Bureau of Economic Research. Abgerufen am 16. Februar 2022 von https://scholar.princeton.edu/sites/default/files/markus/files/blockchain_paper_v3g.pdf

Aggarwal, S. & Kumar, N. (2021). Basics of blockchain. *Advances in Computers* (S. 129–146). Elsevier. https://doi.org/https://doi.org/10.1016/bs.adcom.2020.08.007

Arch Linux. (2022a). Arch User Repository. Abgerufen am 15. Februar 2022 von https://wiki.archlinux.org/title/Arch_User_Repository

Arch Linux. (2022b). AUR submission guidelines. Abgerufen am 15. Februar 2022 von https://wiki.archlinux.org/title/AUR_submission_guidelines

Arch Linux. (2022c). Official repositories. Abgerufen am 15. Februar 2022 von https://wiki.archlinux.org/title/official_repositories

Arweave. (n. d.). What is Arweave? Abgerufen am 15. Februar 2022 von https://arwiki.wiki/#/en/main

Azmat, F., Hamdan, M. I. & Kamaruzaman, J. H. (2009). Analysis of BitTorrent on an Open Source VPN Technology to End User. *Comput. Inf. Sci.*, *2*(1), 168–183. https://doi.org/10.5539/cis.v2n1p168

Batt, S. (2021). Your Files for Keeps Forever with IPFS. Abgerufen am 15. Februar 2022 von https://blogs.opera.com/tips-and-tricks/2021/02/opera-crypto-files-for-keeps-ipfs-unstoppable-domains/

Belshe, M., Peon, R. & Thomson, M. (2015). Hypertext transfer protocol version 2 (HTTP/2). Abgerufen am 15. Februar 2022 von https://datatracker.ietf.org/doc/html/rfc7540

Benet, J. (2014). IPFS - Content Addressed, Versioned, P2P File System. *arXiv preprint arXiv:1407.3561*.

Benz, A. (2011). *Der moderne Staat*. Oldenbourg Wissenschaftsverlag.

Berndt, R., Altobelli, C. F. & Sander, M. (2016). Zentralisierung versus Dezentralisierung von Entscheidungskompetenzen in internationalen Unternehmen. *Internationales Marketing-Management* (S. 555–559). Springer. https://doi.org/10.1007/978-3-662-46787-9_22

Bharambe, A. R., Herley, C. & Padmanabhan, V. N. (2005). Analyzing and improving BitTorrent performance. *98052*. Abgerufen am 15. Februar 2022 von http://www.cs.cmu.edu/~ashu/papers/msr-tr-2005-03.pdf

Biden, J. R. (2021). Executive Order on Improving the Nation's Cybersecurity. Abgerufen am 15. Februar 2022 von https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Bisht, P., Heim, M., Ifland, M., Scovetta, M. & Skinner, T. (2017). Managing security risks inherent in the use of third-party components. *Executive Information Systems, Inc., White Paper No. Eleven*. Abgerufen am 15. Februar 2022 von https://safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf

BitTorrent.org. (2015). What is BitTorrent? Abgerufen am 15. Februar 2022 von https://www.bittorrent.org/introduction.html

Bondy, B. (2021). IPFS Support in Brave. Abgerufen am 15. Februar 2022 von https://brave.com/ipfs-support/

Brühl, V. (2017). Bitcoins, Blockchain und Distributed Ledgers: Funktionsweise, Marktentwicklungen und Zukunftsperspektiven. *Wirtschaftsdienst*, *97*(2), 135–142. Abgerufen am 15. Februar 2022 von https://www.econstor.eu/bitstream/10419/196496/1/135-142-Bruehl.pdf

Bundesamt für Sicherheit in der Informationstechnik. (2019). *Towards Secure Blockchains - Concepts, Requirements, Assessments* (Techn. Ber.). Abgerufen am 15. Februar 2022 von https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Secure_Blockchain.pdf?___blob=publicationFile&v=3

Buterin, V. et al. (2013). Ethereum Whitepaper. *GitHub repository*. Abgerufen am 15. Februar 2022 von https://ethereum.org/en/whitepaper/

Buterin, V., Conner, E., Dudley, R., Slipper, M., Norden, I. & Bakhta, A. (2019). EIP-1559: Fee market change for ETH 1.0 chain. Abgerufen am 15. Februar 2022 von https://eips.ethereum.org/EIPS/eip-1559

Cappos, J., Samuel, J., Baker, S. & Hartman, J. H. (2008). A Look in the Mirror: Attacks on Package Managers, 565–574. https://doi.org/10.1145/1455770.1455841

Catuogno, L., Galdi, C. & Persiano, G. (2020). Secure Dependency Enforcement in Package Management Systems. *IEEE Transactions on Dependable and Secure Computing*, *17*(02), 377–390. https://doi.org/10.1109/TDSC.2017.2777991

Chen, L., Cong, L. W. & Xiao, Y. (2021). A Brief Introduction to Blockchain Economics. *Information for Efficient Decision Making: Big Data, Blockchain and Relevance* (S. 1–40). World Scientific. https://doi.org/10.1142/9789811220470_0001

Chen, Y., Li, H., Li, K. & Zhang, J. (2017). An improved P2P file system scheme based on IPFS and Blockchain. *2017 IEEE International Conference on Big Data (Big Data)*, 2652–2657. https://doi.org/10.1109/BigData.2017.8258226

Cloudflare Docs. (2022). IPFS Gateway. Abgerufen am 15. Februar 2022 von https://developers.cloudflare.com/distributed-web/ipfs-gateway

Coghlan, N. & Stufft, D. (2013). PEP 440 – Version Identification and Dependency Specification. Abgerufen am 16. Februar 2022 von https://www.python.org/dev/peps/pep-0440/

Crust. (n. d.). Crust - FAQ. Abgerufen am 16. Februar 2022 von https://crust.network/faq/

Danielis, P., Gotzmann, M., Timmermann, D., Bahls, T. & Duchow, D. (2010). A peer-to-peer-based storage platform for storing session data in internet access networks. *Telecommunications: The Infrastructure for the 21st Century*, 1–6. Abgerufen am 16. Februar 2022 von https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.188.5663&rep=rep1&type=pdf

De Filippi, P., Mannan, M. & Reijers, W. (2020). Blockchain as a confidence machine: The problem of trust challenges of governance. *Technology in Society*, *62*, 101284. https://doi.org/https://doi.org/10.1016/j.techsoc.2020.101284

Debian. (2021). Debian Sicherheitshandbuch - Paketsignierung in Debian. Abgerufen am 16. Februar 2022 von https://www.debian.org/doc/manuals/securing-debian-manual/deb-pack-sign.de.html

Debian. (2022a). DebianRepository. Abgerufen am 16. Februar 2022 von https://wiki.debian.org/DebianRepository

Debian. (2022b). Keysigning. Abgerufen am 13. Januar 2022 von https://wiki.debian.org/Keysigning

Denicola, D. (2021). Import Maps. Abgerufen am 15. Februar 2022 von https://wicg.github.io/import-maps/

D'mello, G. & Gonzalez-Velez, H. (2019). Distributed Software Dependency Management Using Blockchain. *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 132–139. https://doi.org/10.1109/EMPDP.2019.8671614

Docker. (2021). Content trust in Docker. Abgerufen am 16. Februar 2022 von https://docs.docker.com/engine/security/trust/

Docker. (2022a). Docker Official Images. Abgerufen am 16. Februar 2022 von https://docs.docker.com/docker-hub/official_images/

Docker. (2022b). Image Access Management. Abgerufen am 16. Februar 2022 von https://docs.docker.com/docker-hub/image-access-management/

Eckert, C. (2018). *IT-Sicherheit: Konzepte - Verfahren - Protokolle* (10. Auflage). De Gruyter Oldenbourg. https://doi.org/10.1515/9783110563900

Elliott, E. (2014). *Programming JavaScript applications: Robust web architecture with node, HTML5, and modern JS libraries*. O'Reilly Media, Inc.

Estes, P. & Brown, M. (2018). OCI Image Support Comes to Open Source Docker Registry. Abgerufen am 15. Februar 2022 von https://opencontainers.org/posts/blog/2018-10-11-oci-image-support-comes-to-open-source-docker-registry/

Ethereum. (2021). Expressions and Control Structures - Error handling: Assert, Require, Revert and Exceptions. Abgerufen am 16. Februar 2022 von https://docs.soliditylang.org/en/v0.8.11/control-structures.html#error-handling-assert-require-revert-and-exceptions

Ethereum. (2022). Blocks. Abgerufen am 16. Februar 2022 von https://ethereum.org/en/developers/docs/blocks/

Ethereum Wiki. (2021). Mining. Abgerufen am 16. Februar 2022 von https://eth.wiki/fundamentals/mining#mining-rewards

Etherscan. (2021). What are the Reasons for Failed Transactions? Abgerufen am 16. Februar 2022 von https://info.etherscan.com/reason-for-failed-transaction/

ethPM. (2019). ethPM CLI Quickstart - Registry. Abgerufen am 16. Februar 2022 von https://ethpm-cli.readthedocs.io/en/latest/quickstart.html#registry

ethPM. (2021a). EthPM Package Specification. Abgerufen am 16. Februar 2022 von https://github.com/ethpm/ethpm-spec

ethPM. (2021b). Introduction. Abgerufen am 16. Februar 2022 von https://docs.ethpm.com/

Eve, M. P. (2021). Warez: The Infrastructure and Aesthetics of Piracy.

Feather, C. (2006). Network News Transfer Protocol (NNTP). Abgerufen am 16. Februar 2022 von https://datatracker.ietf.org/doc/html/rfc3977

Federal Aviation Administration. (1988). AC 25.1309-1A - System Design and Analysis. Abgerufen am 16. Februar 2022 von https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_25_1309-1A.pdf

Filecoin Docs. (2022). Using IPFS and Filecoin. Abgerufen am 16. Februar 2022 von https://docs.filecoin.io/about-filecoin/ipfs-and-filecoin/#using-ipfs-and-filecoin

Filecoin Spec. (2021). Introduction - Sector Faults. Abgerufen am 16. Februar 2022 von https://spec.filecoin.io/#section-systems.filecoin_mining.sector.sector-faults

Gamma, E., Helm, R., Johnson, R., Johnson, R. E., Vlissides, J. et al. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Pearson Deutschland GmbH.

Gatteschi, V., Lamberti, F., Demartini, C., Pranteda, C. & Santamaría, V. (2018). To Blockchain or Not to Blockchain: That Is the Question. *IT Professional*, *20*(2), 62–74. https://doi.org/10.1109/MITP.2018.021921652

GitHub Docs. (2021). GitHub Terms of Service - User-Generated Content. Abgerufen am 16. Februar 2022 von https://docs.github.com/en/github/site-policy/github-terms-of-service#d-user-generated-content

Goldman, E. (2003). Warez trading and criminal copyright infringement. *J. Copyright Soc'y USA*, *51*.

Google Cloud. (2021). Container-Image-Digests verwenden. Abgerufen am 16. Februar 2022 von https://cloud.google.com/architecture/using-container-images

Habermann, H.-J. & Leymann, F. (2020). *Repository: Eine Einführung* (Bd. 8). Walter de Gruyter GmbH & Co KG.

Helliar, C. V., Crawford, L., Rocca, L., Teodori, C. & Veneziani, M. (2020). Permissionless and permissioned blockchain diffusion. *International Journal of Information Management*, *54*, 102136. https://doi.org/https://doi.org/10.1016/j.ijinfomgt.2020.102136

Holth, D. (2012). PEP 427 – The Wheel Binary Package Format 1.0. Abgerufen am 16. Februar 2022 von https://www.python.org/dev/peps/pep-0427/

Huizing, A. & van der Wal, J. A. (2014). Explaining the rise and fall of the Warez MP3 scene: An empirical account from the inside. *First Monday*. Abgerufen am 16. Februar 2022 von https://firstmonday.org/ojs/index.php/fm/article/download/5546/4125

Iansiti, M. & Lakhani, K. R. (2017). The Truth About Blockchain. Abgerufen am 16. Februar 2022 von https://hbr.org/2017/01/the-truth-about-blockchain

Import Maps. (2021). Import Maps. Abgerufen am 16. Februar 2022 von https://github.com/WICG/import-maps

IPFS Docs. (2021). Content addressing and CIDs. Abgerufen am 16. Februar 2022 von https://docs.ipfs.io/concepts/content-addressing/

Isotton, A. (2002). How Software Producers Can Distribute Their Products Directly in DEB Format. Abgerufen am 16. Februar 2022 von https://www.debian.org/doc/manuals/distribute-deb/distribute-deb.html

Jockisch, P. (2021). Practical Application of Cryptographic Checksums. Abgerufen am 16. Februar 2022 von http://www.peterjockisch.de/texte/computerartikel/Kryptographische-Pruefsummen/Kryptographische-Pruefsummen_EN.html

Keybase Book. (n. d.). Server. Abgerufen am 16. Februar 2022 von https://book.keybase.io/docs/server

Khare, R. (2002). Decentralized software architecture. Abgerufen am 16. Februar 2022 von https://apps.dtic.mil/sti/pdfs/ADA441133.pdf

Ma, Y. (2018). Constructing Supply Chains in Open Source Software. *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 458–459.

Merriam, P. (2018). EthPM Package Manifest – Specification. Abgerufen am 16. Februar 2022 von https://ethpm.github.io/ethpm-spec/package-spec.html

Merriam, P., Gheorghita, N., Ryan, D. & g. nicholas d'andrea. (2018). Revised Ethereum Smart Contract Packaging Standard. Abgerufen am 16. Februar 2022 von https://eips.ethereum.org/EIPS/eip-1123

Merrriam, P., Gewecker, C., Gheorghita, N. & g. nicholas d'andrea. (2018). EIP-1319: Smart Contract Package Registry Interface. Abgerufen am 16. Februar 2022 von https://eips.ethereum.org/EIPS/eip-1319

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*. Abgerufen am 16. Februar 2022 von https://bitcoin.org/bitcoin.pdf

National Telecommunications and Information Administration. (2020). Software Bill of Materials. Abgerufen am 16. Februar 2022 von https://www.ntia.gov/sbom

Niranga, S. S. (2015). *Mobile web performance optimization*. Packt Publishing Ltd.

Node.js. (2022). Node.js v17.5.0 documentation. Abgerufen am 16. Februar 2022 von https://nodejs.org/api/esm.html#terminology

npm Docs. (n. d.[a]). About package PGP signatures. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/about-pgp-signatures-for-packages-in-the-public-registry

npm Docs. (n. d.[b]). About scopes. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/about-scopes

npm Docs. (n. d.[c]). About semantic versioning. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/about-semantic-versioning

npm Docs. (n. d.[d]). Creating and publishing unscoped public packages. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/creating-and-publishing-unscoped-public-packages

npm Docs. (n. d.[e]). npm Open-Source Terms - Acceptable Content. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/policies/open-source-terms#acceptable-content

npm Docs. (n. d.[f]). npm Open-Source Terms - Your Content. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/policies/open-source-terms#your-content

npm Docs. (n. d.[g]). npm package scope, access level, and visibility. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/package-scope-access-level-and-visibility

npm Docs. (n. d.[h]). npm-cache - Manipulates packages cache. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/cli/v8/commands/npm-cache

npm Docs. (n. d.[i]). package-lock.json. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/cli/v8/configuring-npm/package-lock-json

npm Docs. (n. d.[j]). package.json - Specifics of npm's package.json handling. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/cli/v8/configuring-npm/package-json

npm Docs. (n. d.[k]). Verifying the PGP signature of a package from the npm public registry. Abgerufen am 16. Februar 2022 von https://docs.npmjs.com/verifying-the-pgp-signature-for-a-package-from-the-npm-public-registry

npm registry. (2020). Package Metadata. Abgerufen am 16. Februar 2022 von https://github.com/npm/registry/blob/master/docs/responses/package-metadata.md

Office of the Historian, Department of State. (1948). National Security Council Directive on Office of Special Projects. Abgerufen am 16. Februar 2022 von https://history.state.gov/historicaldocuments/frus1945-50Intel/d292

Oracle. (2015). Understanding Maven Version Numbers. Abgerufen am 16. Februar 2022 von https://docs.oracle.com/middleware/1212/core/MAVEN/maven_version.htm#MAVEN8855

Oreizy, P. (1999). A flexible approach to decentralized software evolution. *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*, 730–731. https://doi.org/10.1145/302405.303000

Piatek, M., Kohno, T. & Krishnamurthy, A. (2008). Challenges and directions for monitoring P2P file sharing networks, or, why my printer received a DMCA takedown notice. *HotSec*. Abgerufen am 16. Februar 2022 von https://www.usenix.org/legacy/event/hotsec08/tech/full_papers/piatek/piatek_html/index.html

Puthal, D., Malik, N., Mohanty, S. P., Kougianos, E. & Yang, C. (2018). The Blockchain as a Decentralized Security Framework. *IEEE Consumer Electronics Magazine*, *7*(2), 18–21. https://doi.org/10.1109/MCE.2017.2776459

PyPA. (2022a). Core metadata specifications. Abgerufen am 16. Februar 2022 von https://packaging.python.org/en/latest/specifications/core-metadata/

PyPA. (2022b). Python Packaging User Guide - Packaging and distributing projects. Abgerufen am 16. Februar 2022 von https://packaging.python.org/en/latest/guides/distributing-packages-using-setuptools/

Python Wheels. (2021). Python Wheels. Abgerufen am 16. Februar 2022 von https://pythonwheels.com/

Raemaekers, S., van Deursen, A. & Visser, J. (2014). Semantic Versioning versus Breaking Changes: A Study of the Maven Repository. *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, 215–224. https://doi.org/10.1109/SCAM.2014.30

Ramsdell, B. (2004). Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling. Abgerufen am 16. Februar 2022 von https://datatracker.ietf.org/doc/html/rfc3850

Rauch, T. (2009). *Entwicklungspolitik: Theorien, Strategien, Instrumente*. Westermann.

Rauschmayer, A. (2015). Exploring ES6. *Upgrade to the next version of JavaScript*.

Raval, S. (2016). *Decentralized Applications: Harnessing Bitcoin's Blockchain Technology* (1st). O'Reilly Media, Inc.

Reproducible Builds. (n. d.). Reproducible Builds. Abgerufen am 16. Februar 2022 von https://reproducible-builds.org/

Rösler, P., Schlich, M. & Kneuper, R. (2013). *Reviews in der System-und Softwareentwicklung: Grundlagen, Praxis, kontinuierliche Verbesserung*. Dpunkt.Verlag.

Schlatt, V., Schweizer, A., Urbach, N. & Fridgen, G. (2016). Blockchain: Grundlagen, Anwendungen und Potenziale. Abgerufen am 16. Februar 2022 von https://fim-rc.de/Paperbibliothek/Veroeffentlicht/642/wi-642.pdf

Schoder, D., Fischbach, K. & Schmitt, C. (2005). Core Concepts in Peer-to-Peer Networking. *Peer-to-peer computing: The Evolution of a Disruptive Technology*, 1–27. https://doi.org/10.4018/978-1-59140-429-3.ch001

Sharma, A. (2022). Npm Libraries 'colors' and 'faker' Sabotaged in Protest by their Maintainer—What to do Now? Abgerufen am 16. Februar 2022 von https://blog.sonatype.com/npm-libraries-colors-and-faker-sabotaged-in-protest-by-their-maintainer-what-to-do-now

Soto-Valero, C., Benelallam, A., Harrand, N., Barais, O. & Baudry, B. (2019). The Emergence of Software Diversity in Maven Central. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 333–343. https://doi.org/10.1109/MSR.2019.00059

Starke, G. (2015). *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. Carl Hanser Verlag GmbH Co KG.

Storey, G., Reisman, D., Mayer, J. & Narayanan, A. (2017). The future of ad blocking: An analytical framework and new techniques. *ArXiv*, *abs/1705.08568*. Abgerufen am 16. Februar 2022 von https://arxiv.org/pdf/1705.08568.pdf

The Apache Software Foundation. (2021a). Guide to uploading artifacts to the Central Repository. Abgerufen am 16. Februar 2022 von https://maven.apache.org/repository/guide-central-repository-upload.html

The Apache Software Foundation. (2021b). POM Reference - Maven Coordinates. Abgerufen am 16. Februar 2022 von https://maven.apache.org/pom.html#Maven_Coordinates

The Apache Software Foundation. (2022). Maven Central Repository. Abgerufen am 16. Februar 2022 von https://maven.apache.org/repository/index.html

The Arweave Project. (2019). Arweave+IPFS: Persistence for the InterPlanetary File System. Abgerufen am 16. Februar 2022 von https://arweave.medium.com/arweave-ipfs-persistence-for-the-interplanetary-file-system-9f12981c36c3

The Central Repository Documentation. (2021). The Central Repository Documentation - About. Abgerufen am 16. Februar 2022 von https://central.sonatype.org/pages/about/

The Graph. (2022). The Graph - Introduction. Abgerufen am 16. Februar 2022 von https://thegraph.com/docs/en/about/introduction/

TheoryOrg. (2016). BitTorrentApplications. Abgerufen am 16. Februar 2022 von https://wiki.theory.org/BitTorrentApplications

TheoryOrg. (2017). Bittorrent Protocol Specification v1.0. Abgerufen am 16. Februar 2022 von https://wiki.theory.org/BitTorrentSpecification

Thomson, E. (2021). GitHub Advisory Database now powers npm audit. Abgerufen am 16. Februar 2022 von https://github.blog/2021-10-07-github-advisory-database-now-powers-npm-audit/

TIOBE. (2022). TIOBE Index for January 2022: January Headline: Python Programming Language of the Year 2021. Abgerufen am 16. Februar 2022 von https://www.tiobe.com/tiobe-index/

Tsitoara, M. (2019). *Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer*. Apress.

Ubuntu. (2019). Debian package version number format. Abgerufen am 16. Februar 2022 von https://manpages.ubuntu.com/manpages/trusty/man5/deb-version.5.html

Uniswap. (2020). Uniswap Interface + IPFS. Abgerufen am 16. Februar 2022 von https://uniswap.org/blog/ipfs-uniswap-interface

Vollmer, A. (2020). Standing up for developers: youtube-dl is back. Abgerufen am 16. Februar 2022 von https://github.blog/2020-11-16-standing-up-for-developers-youtube-dl-is-back/

Webpack. (2020). babel-loader. Abgerufen am 16. Februar 2022 von https://webpack.js.org/loaders/babel-loader/

Wieczner, J. (2017). Hacking Coinbase: The Great Bitcoin Bank Robbery. Abgerufen am 16. Februar 2022 von https://fortune.com/2017/08/22/bitcoin-coinbase-hack/

Williams, S. A., Diordiiev, V., Berman, L., Raybould, I. & Uemlianin, I. (2019). Arweave: A protocol for economically sustainable information permanence. Abgerufen am 16. Februar 2022 von https://www.arweave.org/yellow-paper.pdf

Williams, S. & Jones, W. (2018). Arweave Lightpaper. Abgerufen am 16. Februar 2022 von https://www.arweave.org/files/arweave-lightpaper.pdf

Wittern, E., Suter, P. & Rajagopalan, S. (2016). A Look at the Dynamics of the JavaScript Package Ecosystem. *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories*, 351–361. https://doi.org/10.1145/2901739.2901743

Anhang

A. Übersicht über die Ergebnisse der Software-Repository-Analyse

Bei der nachfolgenden Tabelle handelt es sich um eine reduzierte Form der Originalauswertung, die im Rahmen der Untersuchung von existierenden Software Repositories erarbeitet wurde. Es wurden einige Details sowie Aspekte entfernt, die für das Verständnis der Arbeit nicht zwingend notwendig sind. Fußnoten mit einigen Detailerklärungen sowie Quellenangaben sind nach der Tabelle gesammelt angeordnet. Die komplette Tabelle befindet sich im digitalen Anhang.

	Maven	**NPM**	**PyPI**	**Docker**	**Debian**	**AUR**	**Github**	**WAREZ**	**BitTorrent**	**Usenet**	**ethPM**
ALLGEMEIN											
Sprachen	JVM Sprachen	js, ts, css, Assets, etc	Python	OCI Images	Debian Packages	Build-Anweisungen	Git Repositories	-	-	-	Solidity
Dependency Tools	Maven, Gradle, IVY, SBT, etc[142]	npm, yarn, etc	pip	Docker, podman, kubernetes	apt	makepkg/ pacman, yay, etc[143]	Git	-	BitTorrent Clients[144]	Usenet Reader	ethpm-cli
Packaging-Format	jar, zip	tgz	tar.gz, wheel[145]	Container, Layers (compressed)	deb[146], tar.gz	Git/ PKGBUILD, pkg.tar.zst[147]	git	rar, zip	-	rar, zip	JSON, weitere Daten[148]
Release-Form	compiled	source, compiled	source (+ compiled)	compiled/ packed	compiled, source	source, binary	-	-	-	-	source
Hauptsponsor	Sonatype	npm Inc	Python Software Foundation, Sponsoren	Docker Inc	SPI, Sponsoren	Arch Linux Team	GitHub Inc	-	-	-	-
Kommunikationsprotokoll	http, https	http, https	https	http, https	https	https, git	https, ssh, git	ftp, fxp[149]	BitTorrent[150]	nntp[151]	RPC, IPFS, Swarm[152]

(Weiter auf der nächste Seite)

(Fortsetzung von vorheriger Seite)

	Maven	**NPM**	**PyPI**	**Docker**	**Debian**	**AUR**	**Github**	**WAREZ**	**BitTorrent**	**Usenet**	**ethPM**
INHALTSSTRUKTUR											
Gruppen	ja	ja	nein	ja	nein	nein	ja	-	-	-	über eigene Registry
Versionierung[153]	SemVer-ähnlich[154]	(SemVer)[155]	PEP-440[156,157]	(SemVer)	Debian Package Versioning[158]	Git, App abhängige Versionen	Git, Hashes, Tags	pre	-	-	(SemVer)[159]
Meta-informationen	ja	ja[160]	ja[161]	ja	ja	ja	ja	ja	ja	ja	ja
Explizites Dokumentation Hosting	ja	nein	nein	nein	nein	nein	ja	nein	nein	nein	ja, optional
Explizites Source Code Hosting	ja	nein	ja	nein	ja	nein	ja	nein	nein	nein	ja
CONTENT GUIDELINES											
open source	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja
closed source	nein	ja	nein	ja	ja	ja	ja	ja	ja	ja	nein
Proaktive Inhaltskontrolle	nein	nein	nein	nein	ja	nein	nein	nein	nein	nein	nein, optional

(Weiter auf der nächste Seite)

(Fortsetzung von vorheriger Seite)

	Maven	**NPM**	**PyPI**	**Docker**	**Debian**	**AUR**	**Github**	**WAREZ**	**BitTorrent**	**Usenet**	**ethPM**
Bearbeitung der Inhalte durch Admins	ja	ja	ja	ja	ja	ja	ja	ja	nein	durch Provider	optional
REPOSITORY-STRUKTUR											
Hosting	zentral	zentral	zentral	zentral	zentral mit Mirrors	zentral	zentral	dezentral[162]	p2p	Netz von Usenet Providern	Blockchain + p2p
Ausfallsicherheit	mittel	mittel	mittel	mittel	hoch	mittel bis gering	mittel	gering	hoch	gering	hoch
Ausfallsicherheits-maßnahmen	CDN, wenige Mirrors[163]	CDN, wenige Mirrors	CDN, wenige Mirrors	CDN, wenige Mirrors	viele unabhängige Mirrors	-	zentralisiertes System	geringe Menge an Sites	p2p (ohne zentrale Tracker)	Abgleich zwischen Providern	On-Chain, p2p
Alternative Repositories	ja	nein	nein	ja	ja	nein	ja, GitLab, Bitbucket	ja	-	-	ja
VERZEICHNISSTRUKTUR											
Web Interface	extern	ja	ja	ja	extra	extra	ja	-	ja	ja	nein
API	nein	ja	ja	ja	nein	nein	ja /git	-	-	ja	ja
File System	ja	nein	nein	nein	ja	ja	nein	(ja)	nein	(ja)	nein
Struktur sichtbar/navigierbar	ja	nein	nein	nein	ja	nein	nein	ja	nein	nein	nein

(Weiter auf der nächste Seite)

(Fortsetzung von vorheriger Seite)

	Maven	**NPM**	**PyPI**	**Docker**	**Debian**	**AUR**	**Github**	**WAREZ**	**BitTorrent**	**Usenet**	**ethPM**
SECURITY - INTEGRITÄT											
Überschreibbarkeit	nein	ja	ja[164]	ja	ja	ja, Git force push	ja, Git force push	(nein)[165]	nein	nein	nein, optional
Überschreibbarkeit - Admin	ja	ja	ja	ja[166,167]	ja	ja	ja	ja	nein[168]	ja[169]	nein, optional
Signierung	ja	ja, durch NPM[170, 171]	ja, optional versteckt	optional	ja[172]	ja, optional	ja, optional	nein	nein	nein	nein
Checksumme	ja	ja	ja	ja, Layer und Image Digest[173]	ja	ja, optional, Git	ja	ja	ja	ja, par2	ja
Verschlüsselung	ja	ja	ja	ja	optional	ja	ja	ja, und VPNs	ja, optional + VPN	optional	ja
Zensursicher	nein	nein	nein	nein	nein	nein	nein	nein	ja	nein	ja
SECURITY - ZUGRIFF											
Autor-Registrierungsprozess	manuell	automatisiert	automatisiert	automatisiert	manuell	automatisiert	automatisiert	manuell	automatisiert	automatisiert	Registry abhängig

(Weiter auf der nächste Seite)

(Fortsetzung von vorheriger Seite)

	Maven	**NPM**	**PyPI**	**Docker**	**Debian**	**AUR**	**Github**	**WAREZ**	**BitTorrent**	**Usenet**	**ethPM**
Registrierung für Schreibzugriff	ja	ja	ja	ja	ja	ja	ja	ja	nein	nein	Registry abhängig
Lesen Permissionless	ja	ja	ja	ja	ja	ja	ja	nein	ja	ja	ja
Lese-Einschränkungen	nein	nein	nein	ja, Rate Limit	nein	nein	ja, Rate Limit	nein, Credit System[174]	nein	nein	nein
Geschlossene Nutzergruppen	nein	ja, nur kostenpflichtig	nein	ja	nein	nein	ja	nein	nein, außer private Tracker	nein	nein

[142]Vgl. The Central Repository Documentation 2021.
[143]Vgl. Arch Linux 2022c.
[144]Vgl. TheoryOrg 2016.
[145]Vgl. Python Wheels 2021.
[146]Vgl. Isotton 2002.
[147]Vgl. Arch Linux 2022b.
[148]Vgl. Merriam et al. 2018.
[149]Vgl. Eve 2021, S. 76-78.
[150]Vgl. TheoryOrg 2017.
[151]Vgl. Feather 2006.
[152]Vgl. Merriam et al. 2018.
[153](SemVer) = Empfohlen aber nicht verbindlich.
[154]Vgl. Oracle 2015.
[155]Vgl. npm Docs n. d.(c).
[156]Vgl. Coghlan und Stufft 2013.
[157]Vgl. PyPA 2022b.
[158]Vgl. Ubuntu 2019.
[159]Vgl. Merriam 2018.
[160]Vgl. npm registry 2020.
[161]Vgl. PyPA 2022a.
[162]Vgl. Eve 2021, S. 135-136.
[163]Vgl. The Apache Software Foundation 2022.
[164]yank ist bevorzugt.
[165]Pakete können aus Platzgründen gelöscht werden, defekte Releases werden logisch durch funktionierende ersetzt, das defekte Paket bleibt erhalten.
[166]Inaktive Images werden gelöscht, Administratoren oder Org. Admins können Images löschen, Images können inaktiv geschaltet und anschließend gelöscht werden.
[167]Vgl. Docker 2022b.
[168]Nur Links von Trackern etc. können entfernt werden.
[169]Provider können aufgrund z.B. DMCA Daten entfernen, Nutzer können Daten nicht löschen.
[170]Vgl. npm Docs n. d.(a).
[171]Vgl. npm Docs n. d.(k).
[172]Vgl. Debian 2022b.
[173]Vgl. Google Cloud 2021.
[174]Vgl. Eve 2021, S. 30.

B. Naive Repository Index - Implementation

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity ^0.8.0;
4
5  import "./RepositoryIndex.sol";
6
7  import "@openzeppelin/contracts/utils/Context.sol";
8
9  /**
10  * @dev Naive Implementation of the decentralized Repository Index
       Interface
11 *      This implementation is not optimized
12 */
13 contract NaiveRepositoryIndex is
14   RepositoryIndex,
15   Context {
16
17     // TODO: Add events
18
19   struct Group {
20     address[] owners;
21     mapping(string => Package) packages;
22     string[] packageNames;
23     bool exists;
24   }
25
26   struct Package {
27     mapping(string => Release) releases;
28     string[] versions;
29     bool exists;
30   }
31
32   struct Release {
33     string[] content;
34     bool nuked;
35   }
36
37   mapping(string => Group) private _groups;
38
39   modifier onlyGroupOwner(string memory groupName) {
40     Group storage group = _groups[groupName];
41
42     require(group.exists, "RepoIndex: group does not exist");
43
44     bool isOwner = false;
45     for (uint i = 0; i < group.owners.length; i++) {
46       if (group.owners[i] == _msgSender()) {
47         isOwner = true;
48         break;
49       }
50     }
```

```
51
52      require(isOwner, "RepoIndex: msg sender is not owner");
53      _;
54    }
55
56    function register(string memory groupName) external override {
57      Group storage group = _groups[groupName];
58
59      require(!group.exists, "RepoIndex: group does already exist");
60
61      group.exists = true;
62      group.owners.push(_msgSender());
63    }
64
65    function addOwner(string memory groupName, address newOwner)
66      external override onlyGroupOwner(groupName) {
67
68      Group storage group = _groups[groupName];
69      group.owners.push(newOwner);
70    }
71
72    function removeOwner(string memory groupName, address owner)
73      external override onlyGroupOwner(groupName) {
74
75      Group storage group = _groups[groupName];
76      if (group.owners.length == 1) {
77        // owner deletes himself
78        group.owners.pop();
79      } else {
80        uint index = 0;
81        for (uint i = 0; i < group.owners.length; i++) {
82          if (group.owners[i] == owner) {
83            index = i + 1; // hack
84            break;
85          }
86        }
87
88        require(index > 0, "RepoIndex: user is not part of owner group");
89
90        group.owners[index - 1] = group.owners[group.owners.length-1];
91        group.owners.pop();
92      }
93    }
94
95
96    function release( string memory groupName,
97                      string memory package,
98                      string memory version,
99                      string[] memory content) external override
100                     onlyGroupOwner(groupName) {
101
102     require(content.length > 0, "RepoIndex: no content provided");
103
104     Group storage group = _groups[groupName];
```

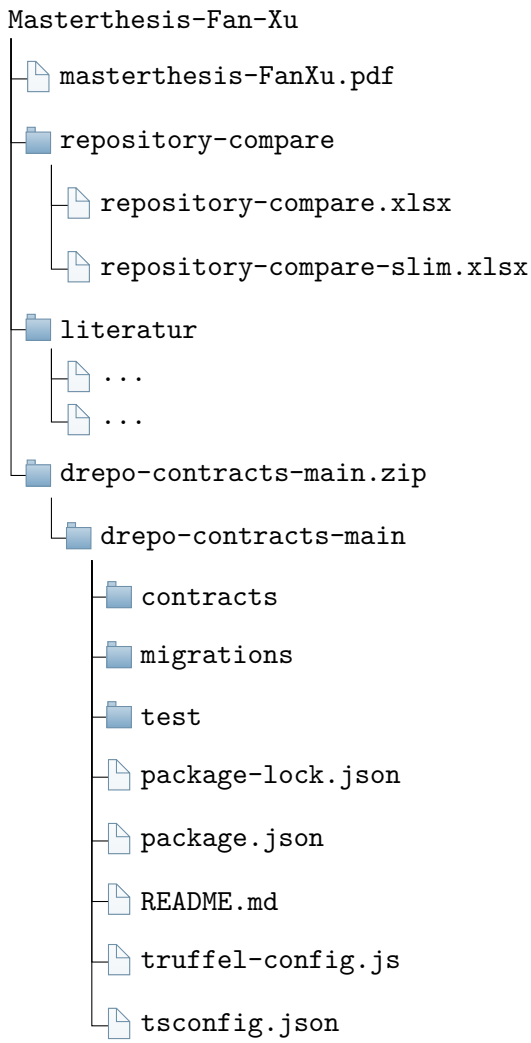
```
105     if (!group.packages[package].exists) {
106       group.packages[package].exists = true;
107       group.packageNames.push(package);
108     }
109     if (group.packages[package].releases[version].content.length == 0) {
110       group.packages[package].versions.push(package);
111     }
112
113     _addContent(groupName, package, version, content);
114   }
115
116   function nuke( string memory groupName,
117                  string memory package,
118                  string memory version,
119                  string[] memory content) external override
120                  onlyGroupOwner(groupName) {
121
122     Group storage group = _groups[groupName];
123     Release storage _release = group
124                               .packages[package]
125                               .releases[version];
126
127     _addContent(groupName, package, version, content);
128     _release.nuked = true;
129   }
130
131   function _addContent( string memory groupName,
132                         string memory package,
133                         string memory version,
134                         string[] memory content) internal {
135     Group storage group = _groups[groupName];
136     string[] storage _content = group
137                                 .packages[package]
138                                 .releases[version]
139                                 .content;
140
141     for (uint i = 0; i < content.length; i++) {
142       _content.push(content[i]);
143     }
144   }
145
146   function groupExists(string memory groupName)
147     external view override returns (bool) {
148
149       return _groups[groupName].exists;
150   }
151
152   function packages(string memory groupName)
153     external view override returns (string[] memory) {
154       return _groups[groupName].packageNames;
155     }
156
157   function releases(string memory groupName,
158                     string memory package)
```

```
159     external view override returns (string[] memory) {
160       return _groups[groupName].packages[package].versions;
161     }
162
163   function getRelease(string memory groupName,
164                       string memory package,
165                       string memory version
166                       ) external view override returns (
167                         string[] memory content,
168                         bool nuked) {
169
170       Release storage r = _groups[groupName].packages[package].releases[
              version];
171       content = r.content;
172       nuked = r.nuked;
173   }
174
175   function owners(string memory groupName)
176     external view override returns (address[] memory) {
177
178       Group storage group = _groups[groupName];
179       return group.owners;
180     }
181 }
```

Listing 3: Beispielimplementation des Naive Repository Index

C. Inhalt des beigefügten USBs

Auf dem beigefügten USB-Stick befinden sich diese Masterthesis sowie der Quellcode der Beispielimplementation des dRepo Contract. Die Installation muss nach der Anleitung in der enthaltenen `README.md`-Datei erfolgen. Ebenfalls ist der Quellcode unter https://github.com/FanXuX/drepo-contracts.git verfügbar. Außerdem sind alle Online-Quellen zwecks Nachprüfbarkeit in ihrer aktuellsten Version hinzugefügt worden. Der Inhalt auf dem USB-Stick entspricht folgendem Schema:



Glossar und Begriffsdefinition

A | **B** | **C** | **D** | **G** | **H** | **I** | **M** | **O** | **P** | **Q** | **R** | **S** | **T** | **W**

A

Artefakt

Dateien einer Software in definierten Formaten, die in einem Release zu einem Software Repository hinzugefügt wurden.

Autor

In der Regel ein Softwareentwickler, der eine Softwarebibliothek oder Applikation geschrieben hat und diese in ein Software Repository hinzufügen möchte.

B

BitTorrent

Ein Peer-to-Peer Filesharing Protokoll, das entweder zentralisierte Server (Tracker) oder einen Distributed Hash Table verwenden kann, um Teilnehmer zu finden. Metadaten in .torrent Dateien oder in Magnet Links erlauben das Finden von Inhalten mittels Content Addressing im Netzwerk.

Blockchain

Ein distributed Ledger (»verteiltes Kontenbuch«), in dem Datensätze in Form von Blöcken in einer fortlaufenden Liste verknüpft werden. Diese Verkettung ist mittels kryptographischer Verfahren gegen Manipulation gesichert. Blockchains werden in der Regel in einem verteilten System durch zahlreiche Teilnehmer betrieben. Das Wort Blockchain wird auch als "Chain" abgekürzt.

C

Cross-Origin Resource Sharing

Cross-Origin Resource Sharing ist ein Mechanismus, der es Web-Browsern oder anderen Web-Clients erlaubt, herkunftsübergreifende Anfragen sicher zu stellen. Zugriffe dieser Art sind normalerweise durch die Same-Origin-Policy verboten.

D

Distributed Hash Table

Ein verteiltes System, in dem Key-Value-Paare gespeichert werden. Die enthaltenen Informationen sind über die teilnehmenden Knoten des Systems verteilt und Knoten können im Netzwerk Inhalte über andere Knoten erfragen. Dadurch ist der gesamte Dateninhalt auslesbar.

G

Gas Fee

Um Aktionen auf einer Blockchain ausführen zu können, muss eine Gebühr an die Miner bezahlt werden. Dieser Fee ist variabel und abhängig von mehreren Faktoren wie beispielsweise der Menge an ausstehenden Transaktionen und Dringlichkeit von Transaktionen.

Git

Ein dezentrales Versionsverwaltungssystem, welches keinen zentralen Server benötigt. Entwickler besitzen lokale Kopien der gesamten Softwarehistorie und können so verteilt, nicht-linear Software schreiben.

H

Hashfunktion

Hashfunktionen sind Abbildungsfunktionen, die unter anderem in kryptographischen Verfahren Einsatz finden. Die hierfür verwendeten Varianten zeichnen sich durch ihre Einwegeigenschaft sowie Kollisionsresistenz aus. Sie werden genutzt um größere Datenmengen in einem definierten Raum abzubilden. Dies erlaubt die umgekehrte Identifikation der Originaldaten mit einer hohen Wahrscheinlichkeit.

I

InterPlanetary File System

Ein Dateisystem, das auf einem Peer-to-Peer Datennetzwerk aufbaut. Mithilfe von Hashsummen und Merkle Trees können Inhalte im Netzwerk auf Basis von Content Addressing eindeutig identifiziert und gefunden werden.

M

Miner

Ein blockerzeugender Knoten in einem Blockchain-Netzwerk. In der Regel werden sie für ihren Beitrag zum Netzwerk auf monetäre Art und Weise belohnt.

Mirror

Ein Mirror ist eine Kopie eines Datensatzes an einem anderen Ort mit dem Ziel der Ausfallsicherheit und besserer Performance durch regionale Nähe.

O

off-chain

Daten und Aktionen, die abseits einer Blockchain ausgeführt werden oder existieren, werden als off-chain bezeichnet.

on-chain

Daten und Aktionen, die auf einer Blockchain ausgeführt werden oder existieren, werden als on-chain bezeichnet. Beispiele hierfür sind Daten, die in einem Block gespeichert sind, oder Funktionen eines Smart Contracts, die ausgeführt werden.

P

Package

Eine Softwarebibliothek oder Applikation in einem Software Repository, die in einer oder mehreren Versionen unter einem bestimmten Namen zur Verfügung steht.

Peer-to-Peer-Netzwerk

Ein verteiltes Netzwerk basierend auf einer Vielzahl gleichberechtigter, unabhängiger Knoten.

Public-Key-Verfahren

Auch als asymmetrische Verschlüsselung bekannt. Hierbei besteht ein Schlüssel aus einem öffentlichen und einem privaten Teil. Der öffentliche Schlüssel kann nur zur Verschlüsselung genutzt werden. Er ist daher nicht schützenswert und kann öffentlich verteilt werden. Der private Anteil wird zur Entschlüsselung genutzt und muss entsprechend geheimgehalten werden. Weiterhin kann mit dem privaten Schlüssel eine Signatur für einen Inhalt erzeugt und mit dem öffentlichen Teil dessen Authentizität nachgewiesen werden.

Q

Quadratic Funding

Ein Finanzierungsmodell für öffentliche Güter, das Mittel auf Basis der Anzahl der beitragenden Entitäten anstatt der Betragssummen verteilt.

R

Release

Eine Menge an Artefakten, die eine Version eines (Software) Packages darstellen.

Repository

Ein Dienst, der es erlaubt, digitale Inhalte in einer Struktur zu speichern und zu veröffentlichen. Autoren können neue Inhalte hinzufügen und Konsumenten existierende herunterladen.

S

Signatur

Mithilfe einer digitalen Signatur kann die Authentizität eines digitalen Inhalts nachgewiesen werden. Dafür werden asymmetrische Verschlüsselungsverfahren

eingesetzt. Mit dem privaten Schlüssel kann eine Signatur von einem Datensatz erzeugt werden. Durch den öffentlichen Schlüssel kann anschließend anhand der Signatur nachgewiesen werden, dass der Datensatz zwischendurch nicht manipuliert wurde und von dem Besitzer des privaten Schlüssels signiert wurde.

Single Point of Failure

Ein Single Point of Failure stellt einen zentralen Bestandteil eines Systems dar, dessen Ausfall zum Gesamtausfall des Systems führt.

Single Point of Truth

Single Point of Truth ist eine Informationsquelle, die führend Daten verwaltet und an angeschlossene Systeme weiterleitet.

Smart Contract

Ein Programm, das auf einer Blockchain existiert. Die dort definierten Methoden können in Blockchain-Transaktionen ausgeführt werden.

Software Bill of Materials

Eine Materialliste für eine Software, die exakt die verwendeten Abhängigkeiten beschreibt. Mithilfe dieser Liste können unter anderem Sicherheitslücken im Gesamtsystem erkannt werden.

Software Repository

Ein Repository, in dem Softwareentwickler Softwarebibliotheken und Applikationen mit ihren Metadaten speichern und veröffentlichen können.

Software Supply Chain

Bezeichnet alle Punkte und Einflüsse, die zur Erstellung einer Software beitragen. Dies können beispielsweise Software Dependencies und Repositories sowie Infrastruktur wie Cloud Provider und Entwicklungsmethoden wie SCRUM sein.

Software-Ökosystem

Ein Software-Ökosystem besteht aus einer Anzahl an Softwarelösungen und Services, welche die Aktivitäten der Akteure des zugehörigen sozialen oder wirtschaftlichen Ökosystems und der Organisationen, welche die Software bereitstellen, ermöglicht, unterstützt und automatisiert.

T

Tor

Ursprünglich The Onion Router ist ein Overlay-Netzwerk, welches durch Freiwillige betrieben wird. Es erlaubt den Anwendern die anonyme Nutzung des Internets, indem alle Datenverbindungen verschlüsselt mehrfach durch das Netzwerk geleitet werden, bevor sie an ihr Ziel gelangen.

W

WAREZ

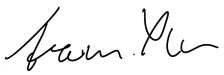
Bezeichnet raubkopierte Software im Internet und im erweiterten Sinne die digitale Verbreitung von urheberrechtlich geschützten Inhalten.

Web3

Im Gegensatz zum Web 2.0 ist das Web3 dezentral organisiert und damit weniger von großen Plattformen abhängig, die mittlerweile weite Teile des Internets kontrollieren. Hier betreiben verteilt alle Nutzer sowie ein Netz unabhängiger Rechner und Server das Internet und dessen Applikationen gemeinsam.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende akademische Abschlussarbeit selbstständig und ohne fremde Hilfe angefertigt habe. Alle Textstellen, die ich wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Quellen übernommen habe, wurden von mir als solche gekennzeichnet.



Hamburg, 17. Februar 2022

.....	.....
Ort, Datum	Unterschrift