



Von der Software Bill of Delivery (SBoD) zur Software Bill of Materials (SBOM)

Ein *Proof of Concept* zur Konvertierung von Open Component Model
Deskriptoren in den CycloneDX SBOM Standard

BACHELORARBEIT

des Studienganges
Wirtschaftsinformatik Business Engineering
an der
Dualen Hochschule Baden-Württemberg Ravensburg

von
Olison Sturm

23. September 2025

Matrikelnummer, Kurs: 2296089, WWIBE222
Dualer Partner: SAP SE, Walldorf
Gutachter: Prof. Dr. Kirchberg

Vorwort

Diese Bachelorarbeit entstand in Zusammenarbeit mit meinem dualen Partner *SAP SE*, konkret im *Open Component Model* (OCM)-Team. OCM wird inzwischen als Projekt innerhalb der *NeoNephos Foundation* (Linux Foundation Europe) geführt. Der inhaltliche Fokus und die Datengrundlage dieser Arbeit wurden im Rahmen dieser Zusammenarbeit gesetzt.

Die Bearbeitung erfolgte im Zeitraum *1. Juli bis 22. September 2025*. Zu Beginn verfügte ich über kein nennenswertes Vorwissen in *Cybersecurity* und lediglich über grundlegende Kenntnisse in verteilten Systemen und containerbasierten Umgebungen. Die intensive Einarbeitung in die zugrundeliegenden Spezifikationen und die Auseinandersetzung mit fachlichen Begriffen und Artefaktformaten waren herausfordernd und zugleich sehr lehrreich. Besonders wertvoll erwies sich der Austausch im OCM-Team sowie die Teilnahme an Community-Calls von *OpenSSF* und *Anchore*. Diese haben mir den Einarbeitungsprozess deutlich erleichtert.

Dank. Mein besonderer Dank gilt *Fabian Burth* (unternehmensseitige Betreuung) und *Jakob Möller* (unternehmensseitige Beratung) aus dem OCM-Team für die kontinuierliche Unterstützung und die vielen klärenden Gespräche. Danken möchte ich zudem den Experten, die mir in Interviews wertvolle Einblicke gegeben haben: *Steven Springett*, *Josh Bressers* und *Jakob Möller*. Mein Dank gilt ebenso allen weiteren OCM-Kollegen, die mir bei technischen Fragen und Programmierproblemen zur Seite standen.

Walldorf, den 23. September 2025

Olison Sturm

Inhaltsverzeichnis

Vorwort	II
Inhaltsverzeichnis	III
Abkürzungsverzeichnis	V
Abbildungsverzeichnis	VII
Tabellenverzeichnis	VIII
Listingsverzeichnis	IX
1 Einleitung	1
1.1 Motivation und Problemstellung	1
1.2 Zielsetzung und Forschungsfragen	2
1.3 Aufbau der Arbeit	3
2 Theoretische Grundlagen und Hintergrund	4
2.1 <i>Software Supply Chain Security</i>	4
2.2 <i>Software Bill of Materials</i> (SBOM) und der CycloneDX-Standard	4
2.3 <i>Software Bill of Delivery</i> (SBoD) und das Open Component Model (OCM)	12
3 Methodisches Vorgehen	16
3.1 Methodenauswahl für die Anforderungserhebung	17
3.2 Ausgestaltung der gewählten Erhebungsmethoden	20
3.2.1 Stakeholderanalyse zur Expertenauswahl	20
3.2.2 Semi-strukturiertes, leitfadengestütztes Experteninterview	22
3.3 Auswertungsmethodik	24
3.3.1 Qualitative Inhaltsanalyse nach Mayring	24
3.3.2 Methodische Triangulation und Anforderungskatalog-Erstellung .	27
4 Anforderungsanalyse	30
4.1 Auswahl der Interviewpartner	30
4.2 Erstellung des Interviewleitfadens	32
4.3 Durchführung der Interviews	33
4.4 Auswertung der Experteninterviews	34
4.5 Triangulation und Erstellung des Anforderungskatalogs	37
5 Konzeption und prototypische Umsetzung	41
5.1 Rahmen der Konzeption	41

5.2	Architektur und Prototypische Implementierung	44
5.2.1	Architekturüberblick	44
5.2.2	Traversierung und <i>Bottom-Up-Merging</i>	45
5.2.3	Zwei Hierarchie-Szenarien	47
5.3	Entwurfsentscheidungen	48
5.4	Anwendungsbeispiel	49
6	Evaluation und Ergebnisinterpretation	53
6.1	Bewertung der funktionalen Anforderungen	53
6.2	Bewertung der nicht-funktionalen Anforderungen	56
6.3	Erkenntnisse und Limitationen	57
7	Fazit und Ausblick	59
Anhang A: Interviewleitfaden		62
Anhang B: Experteninterview mit Jakob Möller		65
Anhang C: Experteninterview mit Steve Springett		84
Anhang D: Experteninterview mit Josh Bressers		106
Anhang E: Kodierleitfaden (Codes + Memos)		145
Anhang F: Summary-Tabelle		148
Anhang G: Table of Minimum Elements Data Fields		171
Anhang H: Exemplarischer OCM Component Descriptor		172
Anhang I: Quellcode <code>cyclonedx_merge.go</code>		174
Anhang J: Quellcode <code>convert_impl.go</code>		191
Literaturverzeichnis		199
Verzeichnis sonstiger Quellen		204
Ehrenwörtliche Erklärung zur Selbständigkeit		205
Erklärung Einsatz von KI-basierten Werkzeugen		206

Abkürzungsverzeichnis

BOM	Bill of Materials
BOV	Bill of Vulnerabilities
CD	Continuous Deployment
CI	Continuous Integration
CISA	Cybersecurity and Infrastructure Security Agency (US)
CLI	Command Line Interface
CPE	Common Platform Enumeration
CRA	Cyber Resilience Act
CTF	Common Transport Format
CVE	Common Vulnerabilities and Exposures
Ecma TC54	Ecma Technical Committee 54 (CycloneDX-Standardisierung)
FANF	Funktionale Anforderung
Go	Go Programming Language
HBOM	Hardware Bill of Materials
IPCEI-CIS	Important Projects of Common European Interest – Cloud Infrastructure and Services
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
JSF	JSON Signature Format
JSON	JavaScript Object Notation
MAXQDA	Software für qualitative Datenanalyse
NANF	Nicht-funktionale Anforderung
NIS2	Richtlinie (EU) 2022/2555 zu Netz- und Informationssicherheit
NTIA	National Telecommunications and Information Administration (US)
NVD	National Vulnerability Database
OCI	Open Container Initiative
OCM	Open Component Model
OSS	Open-Source-Software
OWASP	Open Worldwide Application Security Project
PoC	Proof of Concept
PURL	Package URL
SaaS	Software as a Service
SBoD	Software Bill of Delivery
SBOM	Software Bill of Materials
SHA	Secure Hash Algorithm
SPDX	Software Package Data Exchange
SWID	Software Identification Tags

VEX	Vulnerability Exploitability eXchange
xBOM	Sammelbegriff für domänenspezifische „Bills of Materials“
XML	Extensible Markup Language
XMLsig	XML Signature
YAML	YAML Ain't Markup Language

Abbildungsverzeichnis

2.1	CycloneDX Object Model	11
3.1	Stakeholder-Typologie: eins, zwei oder drei Attribute	21
3.2	Ablaufmodell zusammenfassender Inhaltsanalyse	26
3.3	Prozessmodell induktiver Kategorienbildung	27
3.4	FunktionsMASTER	29
3.5	EigenschaftsMASTER	29
4.1	MAXQDA24 Software für qualitative Datenanalyse	35
4.2	Kodierleitfaden für „Best Practices“	36
5.1	GitHub-Issue: Spike	41
5.2	CycloneDX CLI (cyclonedx) not found	49
5.3	Hoppr (hopctl) CLI not found	50
5.5	Auszug der Root-SBOM	51
5.4	Terminal: Erfolgreiche CLI SBoD zu SBOM Konvertierung	52
7.1	Table of Minimum Elements Data Fields	171

Tabellenverzeichnis

5.1	Verwendete Go-Module und ihr Zweck	49
6.1	Erfüllungsgrad der funktionalen Anforderungen.	54
6.1	Erfüllungsgrad der funktionalen Anforderungen (Fortsetzung).	55
6.2	Erfüllungsgrad der nicht-funktionalen Anforderungen.	56

Listingsverzeichnis

2.1	Minimalbeispiel CycloneDX SBOM	8
2.2	Minimaler OCM Component Descriptor	14
5.1	BFS zum Erzeugen eines Komponentenbaums und der Resource-SBOMs .	45
5.2	Bottom-up-Merge der SBOMs über die Komponentenstruktur	46
5.3	Lesen & Zusammenführen mehrerer R-SBOMs zur C-SBOM	46
5.4	Quellcode: Beispiel component-constructor.yaml	50
8.1	Exemplarischer OCM Component Descriptor	172
9.1	Quellcode <code>cyclonedx_merge.go</code>	174
10.1	Quellcode <code>convert_impl.go</code>	191

1 Einleitung

Ein oft zitiertes Leitmotiv der US-Cybersicherheitsagenda (Executive Order 14028, 2021) bringt den Ausgangspunkt dieser Arbeit auf den Punkt: „Die in digitale Infrastrukturen gesetzte Vertrauenshöhe sollte der tatsächlichen Vertrauenswürdigkeit und Transparenz dieser Infrastrukturen entsprechen.“ [eigene Übersetzung] (NTIA, 2021, S. 5.)

Moderne Software wird entlang dynamischer Lieferketten entwickelt und setzt in großem Umfang auf Open-Source-Komponenten (vgl. Carmody u. a., 2021, S. 1; Xia u. a., 2023, S. 1). Die daraus resultierende *Intransparenz* der Zusammensetzung trägt nachweislich zu Cybersicherheitsrisiken und steigenden Lebenszykluskosten bei (vgl. NTIA Framing Working Group, 2021, S. 5).

Der Vorfall „Log4Shell“ (CVE-2021-44228) in der weltweit verbreiteten Java-Bibliothek Apache Log4j zeigte 2021 beispielsweise, wie kritisch Bibliotheksabhängigkeiten für ganze Ökosysteme sein können. Ein einfacher, präparierter Log-String konnte Remote-Code-Ausführung auslösen. Diese Schwachstelle wurde global für Angriffe auf IT Systeme ausgenutzt. Weil Log4j als Standard-Logging-Baustein in zahllosen Java Anwendungen und Diensten direkt oder transitiv eingebunden ist, hatte diese Schwachstelle eine besonders große Tragweite (vgl. CISA, 2021; National Vulnerability Database, 2025).

Empirische Analysen zeigen zudem eine Zunahme von Angriffen auf Software-Lieferketten, was robuste Transparenzmechanismen erforderlich macht (vgl. Xia u. a., 2023, S. 1).

1.1 Motivation und Problemstellung

Als zentraler Mechanismus zur Herstellung von Transparenz haben sich *Software Bill of Materials* (SBOMs) etabliert: strukturierte, maschinenlesbare Stücklisten von Komponenten, Versionen und Abhängigkeitsbeziehungen (vgl. NTIA, 2021, S. 3).

Parallel dazu bündelt das *Open Component Model* (OCM) komplettete Delivery-Artefakte in einem maschinenlesbaren *Component Descriptor* und fungiert als transport- und signierfähige *Software Bill of Delivery* (SBoD) (vgl. Hohage u. a., 2023; OCM Working Group, 2025a). Es modelliert Komponenten mitsamt ihren *Resources* (lieferbare Artefakte) und *Sources* (Quellcode) in einer standardisierten Struktur (vgl. OCM Working Group, 2025a; Linux Foundation, 2023).

Trotz der komplementären Rollen von *SBoD* und *SBOM* fehlt bisher eine direkte *Interoperabilität* zwischen diesen Standards. In der Praxis bedeutet dies, dass aus einem OCM *Component Descriptor* nicht ohne Weiteres eine *de-facto eingesetzte SBOM* erzeugt werden kann. Unternehmen, die OCM zur Paketierung ihrer Software nutzen, wie etwa SAP, stehen jedoch vor der Herausforderung, für Compliance-Zwecke oder Lieferketten-

Transparenz dennoch SBOMs bereitstellen zu müssen (vgl. Möller, 2025a).

1.2 Zielsetzung und Forschungsfragen

Primäres Ziel ist die Konzeption und Validierung eines *Konvertierungsansatzes*, der die Informationsstruktur des OCM *verlustarm* in *CycloneDX* überführt und so eine unmittelbare Nutzung in Security- und Compliance-Tools ermöglicht. Hierbei werden in dieser Arbeit vor allem *Open Source Vulnerability Scanner Tools* im Fokus stehen. und eine hohe Kompatibilität mit diesen erwartet, um das Ergebnis der Zielspezifikation (*CycloneDX*) zu validieren und Schwachstellen in den OCM Komponenten zu identifizieren.

Daraus ergibt sich für die Thesis folgende Forschungsfrage:

„Wie kann die Informationsstruktur des Open Component Models in den CycloneDX SBOM Standard ohne Informationsverluste überführt werden, sodass die Kompatibilität mit etablierten Software-Supply-Chain-Security-Tools gewährleistet ist?“

Die Forschungsfrage folgt hierbei einem konstruktiven Erkenntnisinteresse, wie es Wieringa (2014, S. 20 f.) beschreibt. Konstruktive Forschungsfragen zielen, laut Wieringa (2014, S. 20 f.), auf die Entwicklung und Evaluation von Artefakten ab, die zur Lösung eines praktischen Problems beitragen. Sie unterscheiden damit grundlegend von deskriptiven oder erklärenden Fragen, da sie nicht das „Was“ oder „Warum“, sondern das „Wie kann“ adressieren.

Zur Beantwortung dieser Frage dient vorab eine methodische Triangulation aus formloser Dokumentenanalyse, Experteninterview und der Ableitung technischer Anforderung an den *Proof of Concept* (PoC). Die Fragestellung wird anschließend im Rahmen einer prototypischen Implementierung erprobt und validiert.

Diese Arbeit fokussiert bewusst *CycloneDX* als Zielspezifikation. *CycloneDX* hat eine starke sicherheitsgetriebene Ausrichtung, entwickelt sich hin zu einem *Transparency Standard* v2.0 und verfügt über breite Toolunterstützung für Erzeugung, Validierung und Analyse. Zugleich erleichtert eine schlanke Designphilosophie die Integration in bestehende Prozesse (vgl. Balliu u. a., 2023, S. 2). *Software Package Data eXchange* (SPDX) ist ein weiter Standard, dieser wird auf Grund des größeren Umfangs und der zeitlichen Machbarkeit in dieser Arbeit nicht vertieft. Sämtliche konzeptionellen und praktischen Überlegungen sind auf *CycloneDX* (v1.6) ausgerichtet.

Die Forschung in dieser Arbeit fokussiert die *Erzeugung*, *Qualität* und *Nutzung* von SBOMs in standardisierten Formaten (*SPDX*; *CycloneDX*) sowie unter regulatorischen Richtlinien. Offen bleibt, wie *lieferorientierte Modelle*, wie das OCM, systematisch mit der *SBOM* Theorie verknüpft werden. Außerhalb von OCM fehlt bislang eine breit ak-

zeptierte *SBoD* Norm. Die gerade vorgestellte Zielsetzung wird daher als *exemplarisches Integrationsproblem* gewählt, an dem sich eine *allgemeine Lösungsklasse* ableiten lässt. Gelingt eine verlustarme Überführung eines hinreichend spezifizierten Liefermodells in ein SBOM, ergibt sich ein *verallgemeinerbares Integrationsprinzip* für weitere *lieferorientierte* Artefaktmodelle.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in sieben Kapitel gegliedert. *Kapitel 2* schafft die theoretischen Grundlagen und erläutert die zentralen Konzepte der Software-Lieferkettensicherheit. Im Mittelpunkt stehen hierbei SBOMs mit Schwerpunkt auf dem CycloneDX-Standard sowie das OCM als Ausprägung einer *SBoD*. *Kapitel 3* beschreibt das methodische Vorgehen, das auf einer systematischen Anforderungserhebung basiert. Dabei wird eine formlose Dokumentenanalyse mit dem Ergebnis der Experteninterviews aus der qualitativen Inhaltsanalyse, trianguliert. Sodass sowohl Vollständigkeit als auch Anwendungsnähe der Anforderungen gewährleistet sind. *Kapitel 4* fasst die aus der Datenerhebung resultierenden Erkenntnisse in einem priorisierten Anforderungskatalog zusammen, der als Grundlage für die Konzeption des Prototyps dient. *Kapitel 5* überführt diese Anforderungen in einen Konvertierungsansatz und zeigt die prototypische Implementierung als *Command Line Tool*. Architekturentscheidungen und ein Anwendungsbeispiel verdeutlichen den praktischen Einsatz. *Kapitel 6* validiert und diskutiert die Ergebnisse sowohl hinsichtlich funktionaler als auch nicht-funktionaler Anforderungen und diskutiert die identifizierten Limitationen sowie weiterführende Implikationen. *Kapitel 7* schließt die Arbeit mit einem Fazit und einem Ausblick auf zukünftige Forschungs- und Entwicklungsperspektiven ab. Ergänzende Materialien wie Interviewleitfaden, Tabellen und Listings sind im *Anhang* dokumentiert.

2 Theoretische Grundlagen und Hintergrund

2.1 *Software Supply Chain Security*

Die Sicherheit von Software-Lieferketten ist die Praxis, die Integrität, Authentizität und Sicherheit von Software während des gesamten Lebenszyklus zu gewährleisten, von der Entwicklung über die Bereitstellung bis zur Wartung (vgl. O'Donoghue u. a., 2025, S. 10 f. NTIA, 2021, S. 5). Sie befasst sich mit den Risiken, die durch die komplexen Abhängigkeiten zu Third-Party-Komponenten wie OSS entstehen (vgl. Xia u. a., 2023, S. 1). Diese Risiken reichen von der unbeabsichtigten Einschleusung von Schwachstellen bis hin zu gezielten Manipulationen durch böswillige Akteure (vgl. Jaatun u. a., 2023, S. 1).

Transparenz und ein besseres Verständnis aller Software-Komponenten ist entscheidend für ein angemessenes Lieferketten-Risikomanagement und das gesamte Unternehmensrisikomanagement (vgl. U.S. Office of the Director of National Intelligence u. a., 2023, S. 1). Durch verbesserte Sichtbarkeit können Unternehmen Schwachstellen effizienter identifizieren und beheben (vgl. Carmody u. a., 2021, S. 1). In diesem Kontext spielen Software Bill of Materials (SBOMs) eine zentrale Rolle, da sie die erforderliche Transparenz schaffen und zur Sicherheit von Software-Lieferketten beitragen (vgl. Balliu u. a., 2023, S. 1). SBOMs werden im Folgenden (Abschnitt 2.2) formal, mit Fokus auf CycloneDX, definiert.

2.2 *Software Bill of Materials (SBOM) und der CycloneDX-Standard*

Kapitel 2.2 konkretisiert den Begriff der *Software Bill of Materials* (SBOM) und verortet ihn im Kontext etablierter Formate und Anforderungen. Ausgangspunkt bilden die von der NTIA und CISA definierten Mindestinhalte und Prozessanforderungen an SBOMs (vgl. NTIA, 2021; CISA, 2025), die als Referenzrahmen für Standardisierung und Interoperabilität dienen. Vor diesem Hintergrund werden die maßgeblichen SBOM-Standards kurz eingeordnet (SPDX, SWID) und die Fokussierung dieser Arbeit auf *CycloneDX* begründet. Der Schwerpunkt liegt dabei auf Zielsetzung, Governance und Datenmodell von CycloneDX sowie auf sicherheits- und compliance-relevanten Aspekten wie Hashes, Signierung, Lizenzangaben und der Anbindung von Vulnerability-Artefakten (z. B. VEX). Damit schafft das Kapitel die fachliche Basis für die spätere theoretische Abgrenzung zu SBoD/OCM und das darauf aufbauende Mapping.

Software Bill of Materials (SBOM)

Eine *Software Bill of Materials* (SBOM) ist „[...]ein formelles, maschinenlesbares Inventar aller Komponenten und deren Abhängigkeiten, die zur Erstellung eines Softwareprodukts verwendet wurden“ (NTIA Framing Working Group, 2021, S. 7). Ähnlich einer Zutaten-

liste auf Lebensmittelverpackungen (vgl. Carmody u. a., 2021, S. 7) dient eine SBOM dazu, relevante Informationen über die Bestandteile eines Softwareartefakts zu erfassen (vgl. Balliu u. a., 2023, S. 1). Ziel ist es, Transparenz über Abhängigkeiten in Software zu schaffen, insbesondere in Fällen, in denen der Quellcode den Benutzern nicht zugänglich ist, als auch zur Standardisierung und Automatisierung bei OSS (vgl. Jaatun u. a., 2023, S. 1 f.). Dies ermöglicht ein verbessertes Management von Kosten und Risiken, indem das Vorhandensein anfälliger Komponenten offengelegt wird (vgl. Carmody u. a., 2021, S. 2).

Zahan (2023, S. 7) sagt, dass SBOMs von Praktikern vor allem als wichtiges Instrument zur Verbesserung der Abhängigkeits-Transparenz, der Schwachstellenanalyse, des proaktiven Risikomanagements sowie der Lizenz-Compliance, genutzt werden. Sie ermöglichen es Entwicklern, anfällige Komponenten zu identifizieren (vgl. Balliu u. a., 2023, S. 1) und reduzieren so Sicherheits- sowie Lizenzrisiken (vgl. Hendrick, 2022, S. 10). O'Donoghue u. a. (2025, S 4-8) überführten diese praktikerzentrierten Vorteile in fünf zentrale Anwendungsbereiche:

- Vulnerability Management (Schwachstellenmanagement)
- Transparency (Transparenz)
- Component Assessment (Komponentenbewertung)
- Risk Assessment (Risikobewertung)
- SSC Integrity (Sicherstellung der Integrität der Lieferkette)

Im Übrigen nutzen die Autoren O'Donoghue u. a. (2025, S. 5) zwei populäre SBOM-Analysetools, Trivy von Aqua Security Software Ltd. (2021) und Grype von Anchore (2025), um basierend auf den SBOM-Informationen Schwachstellen in 1.151 SBOMs zu identifizieren. Sie gewannen dadurch Einblicke in die Verteilung von Schwachstellen und die Sicherheitsrisiken verbreiteter Third-Party-Dependencies, was die Wirksamkeit des Ansatzes bei der Erkennung von Schwachstellen entlang der Software-Lieferkette bestätigte.

Obgleich SBOMs bedeutende Vorteile bieten, sind sie jedoch kein hinreichendes Mittel zur Lösung aller Cybersicherheitsprobleme (vgl. European Parliament and Council, 2022, S. 19). Sie sind primär statisch, während Schwachstellendaten dynamisch sind (vgl. NTIA, 2021, S. 16 f.). Ein SBOM allein zeigt nicht, ob eine Schwachstelle ausgenutzt wurde, ermöglicht aber eine schnelle Überprüfung der Betroffenheit (vgl. Hendrick, 2022, S. 44). Nicht jede Schwachstelle in einer Komponente ist tatsächlich relevant für abhängige Softwareprodukte (vgl. NTIA, 2021, S. 17)

Auch bestehen Herausforderungen hinsichtlich Vollständigkeit, Genauigkeit und Integrität der SBOM-Tools selbst (vgl. Xia u. a., 2023, S. 9), sowie ein Mangel an Branchenkonsens, wie Produktion und Nutzung von SBOMs in die Softwareentwicklung integriert werden

sollen (vgl. Hendrick, 2022, S. 39).

Auf regulatorischer Ebene wird die Bedeutung von SBOMs durch verschiedene Initiativen verstärkt. Die NIS2-Richtlinie (EU 2022/2555) verlangt ein hohes Cybersicherheitsniveau, Berichtspflichten und die Förderung offener Standards (vgl. European Parliament and Council, 2022, S. 90, S. 108). Der Cyber Resilience Act (CRA) schreibt für neue Produkte mit digitalen Elementen, die auf den EU-Markt gebracht werden, die Erstellung eines SBOMs vor (vgl. European Parliament and Council, 2024, S. 2 f.). Als Reaktion auf die Executive Order 14028 legte die National Telecommunications and Information Administration (NTIA) Mindestanforderungen an SBOMs fest, darunter u. a. Angaben zu Lieferantenname, Komponentenname, Version, eindeutigen Identifikatoren, Abhängigkeitsbeziehungen, Autor und Zeitstempel (vgl. NTIA, 2021, S. 3). Diese schreibt zudem die Bereitstellung von SBOMs für Unternehmen vor, die mit der US-Regierung zusammenarbeiten (vgl. NTIA, 2021, S. 5). Die CISA (2025) führt diese Vorgaben fort und präzisiert sie für den operativen Einsatz. Im **Anhang G** sind alle mindestens verlangten Datenfelder einzusehen.

Die NTIA (2021, S. 11) identifizierte **drei Formate**, die ihre **Anforderungen** erfüllen:

- **Software Package Data eXchange (SPDX®)**
- **Software Identification (SWID) tags**
- **CycloneDX**

SWID wird zwar als von der NTIA anerkanntes Format geführt, ist in der Praxis jedoch deutlich seltener verbreitet (vgl. NTIA, 2021, S. 11).

SPDX ist ein weiterer wichtiger Industriestandard zur Beschreibung von Software artefakten und ihren Abhängigkeiten (vgl. Kağızmandere und Arslan, 2024, S. 34). Er wurde als internationaler Standard ISO/IEC JTC1 5962:2021 anerkannt (vgl. SPDX Workgroup, 2025).

Während SPDX durch seine ISO-Standardisierung ein hohes Maß an Verbindlichkeit aufweist, fokussiert sich die Community-Dynamik in den letzten Jahren stärker auf CycloneDX. Der CycloneDX-Standard geht in zentralen Aspekten deutlich über die von der NTIA definierten Mindestanforderungen hinaus (vgl. CycloneDX Core Working Group, 2024, S. 8). Vor diesem Hintergrund wird im Folgenden CycloneDX vertieft betrachtet.

CycloneDX Bill of Materials

CycloneDX strebt an, ein umfassender Standard für Stücklisten (Bill of Materials/BOM) für Software, Hardware, SaaS und Betriebsabläufe zu sein (vgl. Ecma Technical Committee 54, 2024, S. 3). Der Standard hat einen starken Sicherheitsfokus und wurde ursprünglich vom **Open Worldwide Application Security Project (OWASP)** entwickelt (vgl.

Balliu u. a., 2023, S. 2). Dieser ist als leichtgewichtiges Format konzipiert, das eine einfache Einführung und Automatisierung der SBOM-Generierung über den gesamten Softwareentwicklungszyklus hinweg ermöglicht (vgl. NTIA Standards and Formats Working Group, 2019, S. 24). Damit versetzt CycloneDX Organisationen in die Lage, eine detaillierte Transparenz ihrer Software-Ökosysteme zu erreichen (vgl. OWASP Foundation, 2025a).

CycloneDX ist ein Flagship-Projekt der OWASP Foundation und wird zusätzlich über das Ecma Technical Committee 54 (TC54) standardisiert (vgl. Ecma Technical Committee 54, 2024, S. 3). Unterstützt von einer breiten „Global Information Security Community“ (vgl. CycloneDX Core Working Group, 2024) verfügt CycloneDX über ein großes Ökosystem an Tools und eine der weltweit größten Gemeinschaften von SBOM-Anwendern (vgl. OWASP Foundation, 2025b, S. 1). Die aktuelle Version 1.6 wurde am 9. April 2024 veröffentlicht und setzt auf unveränderliche, abwärtskompatible Releases (vgl. Balliu u. a., 2023, S. 2).

Der CycloneDX Standard ist hochmodular und erweiterbar aufgebaut, was die Darstellung einer breiten Palette von Lieferketteninformationen ermöglicht (vgl. OWASP Foundation, 2024). Über das reine SBOM hinaus bietet CycloneDX spezialisierte Erweiterungen auch xBOMs genannt, wie SaaSBOM (Software-as-a-Service), HBOM (Hardware) und VEX (Vulnerability Exploitability eXchange) (vgl. CycloneDX Core Working Group, 2024, S. 8). Diese Aspekte werden in den folgenden Kapiteln genauer erläutert.

Der CycloneDX Standard ist hochmodular und erweiterbar aufgebaut, was die Darstellung einer breiten Palette von Lieferketteninformationen ermöglicht (vgl. OWASP Foundation, 2024). Über das reine SBOM hinaus bietet CycloneDX spezialisierte Erweiterungen auch xBOMs genannt, wie SaaSBOM (Software-as-a-Service), HBOM (Hardware) und VEX (Vulnerability Exploitability eXchange) (vgl. CycloneDX Core Working Group, 2024, S. 8). Diese Aspekte werden in den folgenden Kapiteln genauer erläutert.

Datenmodell und Serialisierungen Das CycloneDX-Datenmodell (eng. „Object Model“) ist im JSON Schema, XML Schema und Protocol Buffers definiert (vgl. CycloneDX Core Working Group, 2024, S. 3). Die gesamte Spezifikation kann mit offiziell unterstützten CycloneDX-Tools oder anderen Validatoren, die JSON Schema, XML Schema oder Protocol Buffers unterstützen, validiert werden (vgl. CycloneDX Core Working Group, 2024, S. 14). Die Kernelemente (vgl. Ecma Technical Committee 54, 2024) eines CycloneDX SBOMs umfassen:

- **bomFormat, specVersion, serialNumber, version:** Diese Felder geben grundlegende Informationen über das SBOM-Dokument selbst an, wie das Format, die Spezifikationsversion, eine eindeutige Seriennummer und die Version des SBOMs.
- **metadata:** Dieses Element enthält Informationen über das Tool, das das SBOM

erstellt hat, und das Projekt, auf dem das Tool ausgeführt wurde. Es umfasst Metadaten zum Lieferanten, Hersteller, Zielkomponente sowie Lizenzinformationen für das SBOM-Dokument selbst.

- **components**: Eine Liste, die Informationen zu jeder im Projekt gefundenen Abhängigkeit enthält. Jedes Komponentenelement kann Hashes enthalten, um den Inhalt der Komponente zu identifizieren und die Build-Integrität zu gewährleisten. Weitere Felder sind group, name und type.
- **services**: Dieses Element beschreibt externe APIs, Endpunkt-URLs und Authentifizierungsanforderungen. Es befasst sich auch mit Vertrauengrenzenüberschreitungen und anderen externen Anforderungen der Software.
- **dependencies**: Eine Liste, die die Beziehungen zwischen allen zuvor aufgelisteten Abhängigkeiten aufzeichnet (z. B. dependsOn).
- **bom-ref**: Referenzen zur Identität eines Objekts innerhalb des BOMs.

Unten ist ein exemplarisches CycloneDX SBOM im JSON-Format (siehe Listing 2.1), das die in Ihrem Abschnitt beschriebenen Kernelemente berücksichtigt. Es zeigt die Minimalstruktur mit dem Dokumentheader, Metadaten zum Erstellwerkzeug, eine Anwendungskomponente samt Hash und PURL sowie die Abhängigkeitsbeziehungen. Die Felder **services** und weitere optionale Elemente werden leer gelassen, können aber je nach Bedarf ergänzt werden.

```
1  {
2    {
3      "bomFormat": "CycloneDX",
4      "specVersion": "1.6",
5      "serialNumber": "urn:uuid:3e48ea66-2f30-472b-a618-91db3d6076e9",
6      "version": 1,
7      "metadata": {
8        "timestamp": "2025-09-23T08:00:00Z",
9        "tools": [
10          {
11            "vendor": "ExampleVendor",
12            "name": "ocm-sbom",
13            "version": "0.2.0"
14          }
15        ],
16        "authors": [
17          {
18            "name": "Example Org",
19            "email": "sbom@example.org"
20          }
21        ],
22        "component": {
```

```
23     "type": "application",
24     "name": "my-app",
25     "version": "1.0.0",
26     "bom-ref": "pkg:maven/com.example/my-app@1.0.0",
27     "hashes": [
28       {
29         "alg": "SHA-256",
30         "content": "ffaf53c22e3b4bdd1bde3de2075a4a3f02fdc01730c8e60b9c761623739a3e3f"
31       }
32     ],
33     "licenses": [
34       {
35         "license": {
36           "id": "Apache-2.0"
37         }
38       }
39     ],
40     "purl": "pkg:maven/com.example/my-app@1.0.0"
41   },
42 },
43 "components": [
44   {
45     "bom-ref": "pkg:maven/com.fasterxml.jackson.core/jackson-databind@2
46     .15.2",
47     "type": "library",
48     "group": "com.fasterxml.jackson.core",
49     "name": "jackson-databind",
50     "version": "2.15.2",
51     "hashes": [
52       {
53         "alg": "SHA-256",
54         "content": "2
55         c1d9ff5b0264f8b8a5f433718f789f35f94758b9473168c4140fda56bd94072"
56       }
57     ],
58     "purl": "pkg:maven/com.fasterxml.jackson.core/jackson-databind@2
59     .15.2"
60   },
61 ],
62 "services": [],
63 "dependencies": [
64   {
65     "ref": "pkg:maven/com.example/my-app@1.0.0",
66     "dependsOn": [
67       "pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.15.2"
```

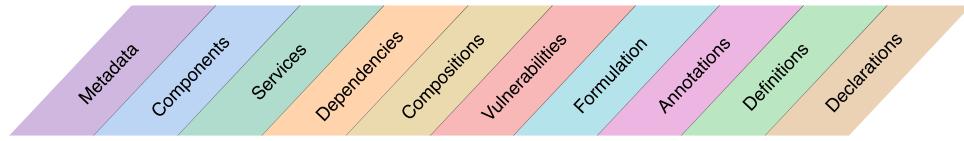
```
65     ]
66   }
67 ]
68 }
```

Listing 2.1: Minimalbeispiel CycloneDX SBOM

Dieses Listing verdeutlicht die Verwendung von `bomFormat`, `specVersion`, `serialNumber` und `version` zur Kennzeichnung des SBOM-Dokuments. Das `metadata`-Element enthält das erzeugende Tool und die Hauptkomponente mit Hash- und Lizenzangaben. Unter `components` finden sich die weiteren Bibliotheken des Projekts, jeweils mit PURL und Hash. Das Feld `dependencies` beschreibt die Beziehungen zwischen den Komponenten.

Das Datenmodell umfasst jedoch noch viele weitere Objekte. Diese sind alle in der folgenden **Abbildung 2.1** inklusive deren zulässige Typen zu sehen.

Within the root bom element, CycloneDX defines the following object types:



The object types are arranged in order and contain (but are not limited to) the following types of data:



Abb. 2.1: CycloneDX Object Model.
(vgl. CycloneDX Core Working Group, 2024, S. 11)

Sicherheits- und Compliance-Aspekte. **Hashes und Signaturen:** CycloneDX unterstützt die Einbindung von kryptographischen Hashes (z.B. SHA-256, SHA-384, SHA-512) für jede Softwarekomponente zur Verifikation der Software-Integrität und der Erkennung von unautorisierten Modifikationen (vgl. CycloneDX Core Working Group, 2024, S. 21 f.). Darüber hinaus ermöglicht CycloneDX die Verwendung von Signaturen (enveloped signing) für SBOMs mittels JSON Signature Format (JSF) und XML Signature (XMLsig), sie sichern zwar die Authentizität der SBOM selbst, ersetzen jedoch nicht die inhaltlichen Hashes auf Komponentenebene, die für Integritätsprüfungen der referenzierten Artefakte maßgeblich sind. (vgl. CycloneDX Core Working Group, 2024, S. 23).

Lizenzzangaben: CycloneDX bietet umfassende Unterstützung für Lizenzinformationen,

einschließlich der Integration von SPDX-Lizenz-IDs und -Ausdrücken (vgl. OWASP Foundation, 2025b, S. 1). Der Standard unterstützt sowohl Open-Source- als auch kommerzielle Lizenzmanagement-Anwendungsfälle (vgl. CycloneDX Core Working Group, 2024, S. 23). Dies ermöglicht die Identifizierung von inkompatiblen Lizenzen oder spezifischen Compliance-Pflichten wie Namensnennung oder der Weitergabe von Quellcode (vgl. CycloneDX Core Working Group, 2024, S. 23). Lizenzangaben können deklarierte und beobachtete Lizenzen sowie Urheberrechtshinweise umfassen (vgl. Ecma Technical Committee 54, 2024, S. 54).

Vulnerabilities: CycloneDX unterstützt die Konzepte Bill of Vulnerabilities (BOV) und Vulnerability Exploitability eXchange (VEX) (vgl. CycloneDX Core Working Group, 2024, S. 8). Ein VEX-Dokument gibt an, ob ein Produkt von einer bekannten Schwachstelle betroffen ist oder nicht (vgl. U.S. Office of the Director of National Intelligence u. a., 2023, S. 18). Dies ist besonders nützlich, um Falsch-Positive zu reduzieren (vgl. U.S. Office of the Director of National Intelligence u. a., 2023, S. 17). CycloneDX ermöglicht die Verknüpfung von VEX-Informationen mit präzisen Angaben, welche Komponenten betroffen sind, innerhalb eines SBOMs (vgl. CycloneDX Core Working Group, 2024, S. 56).

2.3 *Software Bill of Delivery (SBoD) und das Open Component Model (OCM)*

Dieses Kapitel legt die theoretischen Grundlagen der OCM-Spezifikation dar und erläutert die zentralen Konzepte, die für das Verständnis der nachfolgenden praktischen Analyse und Umsetzung relevant sind.

Die moderne Softwareentwicklung und -auslieferung ist von einer hohen Komplexität geprägt (vgl. Mens, 2016, S. 1; Brooks, 1987, S. 3). Insbesondere bei der Bereitstellung von Cloud-nativen Anwendungen über verschiedene Umgebungen wie Public Clouds, Private Clouds oder sogar Air-Gapped-Systeme hinweg entstehen erhebliche Herausforderungen (vgl. OCM Working Group, 2025a; Linux Foundation, 2023). Softwareanbieter müssen nicht nur die Artefakte selbst, sondern auch deren Abhängigkeiten, Konfigurationen und Metadaten konsistent und sicher verwalten und transportieren (vgl. Kober, 2023; Linux Foundation, 2023). Traditionelle Ansätze stoßen hier an ihre Grenzen, da sie oft keine einheitliche Methode bieten, um komplexe Softwarepakete über Sicherheitsgrenzen hinweg zu beschreiben, zu bündeln und nachvollziehbar bereitzustellen (vgl. Kober, 2023; Linux Foundation, 2023).

Als Antwort auf diese Herausforderungen wurde das ***Open Component Model*** (OCM) entwickelt. OCM ist ein *ein offener Standard*, der eine einheitliche und maschinenlesbare

Methode zur Beschreibung und zum Transport von Softwarekomponenten bzw. Softwareartefakten bietet und als sogenannte ***Software Bill of Delivery*** (SBoD) fungiert (vgl. Linux Foundation, 2023; Hohage u. a., 2023; OCM Working Group, 2025a). Ursprünglich von SAP entwickelt wurde OCM im Zuge des offiziellen Launches der *NeoNephos Foundation*, part der *Linux Foundation Europe* am 31. März 2025 an die NeoNephos übergeben und wird seither dort als Projekt geführt (vgl. Linux Foundation, 2025). Die Projekte der NeoNephos Foundation zählen zu den ersten konkreten Open-Source-Ergebnissen des EU-Investitionsvorhabens IPCEI-CIS und werden im Rahmen dieser Initiative gefördert, um sichere, skalierbare und transparente Cloud-Lösungen im Sinne europäischer digitaler Souveränität voranzutreiben (vgl. Linux Foundation, 2025).

Unter SBoD wird in diesem Kontext lediglich die strukturierte Repräsentation einer konkreten *Lieferinstanz* des OCM, auch OCM Component Descriptor (dt. Komponentenbeschreiber) genannt, verstanden, die für Transport, Nachvollziehbarkeit und Wiederherstellbarkeit notwendigen Informationen konsolidiert (vgl. OCM Working Group, 2025a; Linux Foundation, 2023; Hohage u. a., 2023).

Struktur und Kernprinzipien des Open Component Models

Um die OCM-Spezifikation (vgl. OCM Working Group, 2025b) praktisch nutzbar zu machen, existieren *Referenz-Implementierungen*, wie ein CLI (Command line interface) und Controller, die das im Standard definierte Verhalten exemplarisch umsetzen (vgl. Linux Foundation, 2023; OCM Working Group, 2025a). Ein OCM Component Descriptor ist textbasiert serialisiert, typischerweise YAML und schema-geführts versioniert (vgl. Hohage u. a., 2023 und OCM Working Group, 2025b) und umfasst folgende Kernprinzipien:

- **Eindeutige Identifikation.** Komponenten besitzen eine stabile Identität unter anderem Name, Version, Provider) zur präzisen Referenzierung (vgl. OCM Working Group, 2025b).
- **Explizites Ressourceninventar.** Jede Komponente führt ihre auslieferbaren Ressourcen (z. B. Container-Images, Charts, Pakete) mit Typ und Zugriffsdefinition auf (vgl. OCM Working Group, 2025b).
- **Trennung von Identität und Speicherort.** OCM trennt die logische Identität einer Ressource (zum Beispiel den Name eines Images) von ihrem physischen Speicherort (zum Beispiel die URL zu einer Registry) durch „access/Repository“ (vgl. OCM Working Group, 2025b).
- **Komposition und Referenzen.** Über componentReferences lassen sich Komponenten hierarchisch zusammensetzen (vgl. Linux Foundation, 2023). Eine übergeordnete Komponente kann nicht nur aus direkten Ressourcen, sondern auch aus Referenzen auf andere OCM-Komponenten bestehen (vgl. OCM Working Group,

2025b). Hierdurch lassen sich komplexe Produkte als ein Graph von signierten Komponenten abbilden (vgl. Linux Foundation, 2023).

- **Erweiterbare Metadaten über Labels.** OCM erlaubt kontextspezifische **Labels** also Key-Value-Paare, um zusätzliche Semantik, wie zum Beispiel Compliance-Hinweise und oder, um seine SBoD manuell mit eigenen Informationen anzureichern (vgl. Hohage u. a., 2023; OCM Working Group, 2025b).

Wie in **Listing 2.2** zu sehen ist, besteht ein OCM Component Descriptor im Kern aus den Abschnitten **resources**, **sources** und **componentReferences**. Das Beispiel zeigt die grundlegende Struktur an einem minimalen Artefakt, auf das im Text verwiesen werden kann. Eine ausführlichere Ausprägung mit Labels, Digests und Signaturen findet sich im **Anhang H**.

```

1 component:
2   name: example.org/myapp
3   version: 1.0.0
4   provider: example-inc
5   resources:
6     - name: app-image
7       type: ociImage
8       version: "1.0.0"
9       access:
10      type: ociArtifact
11      imageReference: ghcr.io/example/myapp:1.0.0
12   sources:
13     - name: myapp-src
14       type: git
15       access:
16         type: github
17         repoUrl: github.com/example/myapp
18         ref: refs/tags/v1.0.0
19   componentReferences:
20     - name: crypto-lib
21       componentName: example.org/libs/crypto
22       version: "0.9.0"
```

Listing 2.2: Minimaler OCM Component Descriptor

Sicherheit und Integrität im Lieferprozess

OCM adressiert die Absicherung der Softwarelieferkette entlang des gesamten Transportpfads (Supply Chain), von der Erstellung bis zur Auslieferung in restriktive Zielumgebungen durch klar definierte, maschinenlesbare Artefaktbeschreibungen, Digest- und Signaturverfahren sowie reproduzierbare Transfermechanismen (vgl. OCM Working Group,

2025b; OCM Working Group, 2025a). Um dies zu gewährleisten, integriert OCM mehrere Sicherheitsmechanismen:

- **Kryptografische Signaturen:** OCM definiert signierbare *Component Descriptors* und signaturrelevante Felder. Diese Signaturen stellen sicher, dass die Integrität und Authentizität der Artefakte während des gesamten Transports – vom Build bis zum Deployment – gewahrt bleibt, sodass beschriebenen Inhalte *zeitpunkt- und transportunabhängig* verifizierbar sind (vgl. OCM Working Group, 2025b; OCM Working Group, 2025a).
- **Integritätsnachweise (Digests):** OCM verankert für alle beschriebenen Ressourcen und referenzierten Komponenten inhaltliche Prüfsummen (inklusive Normalisierungs- und Hash-Algorithmus). Diese Digests sind transport- und zeitpunktunabhängig verifizierbar und bilden die modellinterne Grundlage für Herkunftsbezüge (Provenance) (vgl. OCM Working Group, 2025b; OCM Working Group, 2025a). Für den vorliegenden Konvertierungsansatz sind sie maßgeblich, da sie im Mapping unmittelbar in die Hash-Felder der Ziel-SBOM überführt werden.
- **Transport über Sicherheitsgrenzen (CTF):** OCM nutzt hierfür das *Common Transport Format* (CTF). CTF paketiert den *Component Descriptor* samt zugehöriger Inhalte zu einer transportfähigen, formatstabilen Repräsentation (Verzeichnis/Archiv). Die Semantik bleibt unverändert, lediglich die Materialisierung unterscheidet sich (by-reference vs. by-value via `localBlob`). CTF ermöglicht Air-Gap-Übertragungen, ohne Digests/Signaturen zu verändern. Für die Konvertierung genügt, dass by-value-Inhalte vollständig vorliegen und die zugehörigen Digests konsistent sind (vgl. OCM Working Group, 2025b; OCM Working Group, 2025a; Linux Foundation, 2023).

3 Methodisches Vorgehen

In diesem Kapitel wird das methodische Vorgehen der Arbeit dargelegt. Die Forschung folgt einer designwissenschaftlichen Logik im Sinne des „Design-Science“-Paradigmas nach Hevner u. a. (2004, S. 82–86, 88). Entsprechend dieses Paradigmas wird das Artefakt – ein PoC zur Konvertierung von SBoD (OCM) nach SBOM (CycloneDX) – in einer Erhebung praxisrelevanter technischer Anforderungen, einer Fundierung in Spezifikationen sowie einem Designzyklus, also der Konzeption, prototypischen Umsetzung und Evaluation, entwickelt (vgl. Hevner u. a., 2004, S. 82–86, 88).

Der übergeordnete Forschungsrahmen wird zwar durch das Design-Science-Paradigma strukturiert, wobei für die operative Ausgestaltung der Anforderungserhebung Requirements Engineering (RE)-spezifische Vorgehensempfehlungen maßgeblich sind (vgl. Rupp und SOPHIST, 2021, S. 30). Innerhalb des RE, das eine Reihe von Tätigkeiten wie das Ermitteln, Dokumentieren, Prüfen und Abstimmen, sowie das Verwalten von Anforderungen einschließt, fokussiert sich die vorliegende Arbeit bewusst auf die Anforderungserhebung als zentralen Teilschritt (vgl. Herrmann, 2022, S. 28). Dies ist begründet, die Ermittlung von technischen Anforderungen, welche die informationsschaffende Grundlage für den PoC bildet (vgl. Herrmann, 2022, S. 28).

Die Erhebung wird dabei entlang der Erhebungsgegenstände nach Rupp und SOPHIST (2021, S. 85–86, 107–114) strukturiert:

1. **Erhebungsziel:** Anforderungen müssen einem expliziten Erhebungsziel dienen. Dieses wird zu Beginn festgelegt und auf den Konzeption des PoC ausgerichtet (vgl. Rupp und SOPHIST, 2021, S. 85).
2. **Ermittlungsquellen:** Quellen sind Objekte oder Personen, die Anforderungswissen tragen, wie zum Beispiel Spezifikationen, Altsysteme und vor allem Stakeholder (vgl. Rupp und SOPHIST, 2021, S. 86).
3. **Stakeholder:** Identifikation primäre Wissensträger, nämlich salienzstarke (vgl. Mitchell u. a., 1997) Stakeholder, wie Nutzer, Entwickler, Projektleitung und so weiter (vgl. Herrmann, 2022, S. 28; Rupp und SOPHIST, 2021, S. 107 f.).
4. **System und Systemkontext:** Klare Abgrenzung, welche Teile Anforderungen tragen und welche lediglich interagieren, um Schnittstellen- und Kompatibilitätsanforderungen korrekt zu adressieren (vgl. Rupp und SOPHIST, 2021, S. 113 f.).
5. **Dokumentationsform:** Anforderungen werden normgerecht dokumentiert (ISO/-IEC/IEEE 29148-konform), unter anderem mit SOPHIST-Satzschablonen, um Eindeutigkeit, Vollständigkeit und Prüfbarkeit sicherzustellen (vgl. Rupp und SOPHIST, 2021, S. 86).

3.1 Methodenauswahl für die Anforderungserhebung

Ausgehend von den Erhebungsgegenständen nach Rupp und SOPHIST (2021, S. 143–201) kommen grundsätzlich verschiedene Methoden zur Anforderungsermittlung in Frage. Diese lassen sich in beobachtende Techniken (zum Beispiel Feldbeobachtung), befragende Techniken (Interviews, Fragebögen, Workshops) und andere Techniken (Dokumentenanalyse, Systemarchäologie, perspektivenbasierte Ansätze) kategorisieren. Ergänzend dazu unterscheidet Herrmann (2022, S. 31–33) zwischen kreativen Techniken (Brainstorming, Kreativitätstechniken) und analytischen Techniken.

Die vorliegende Forschungsfrage zielt auf die technische Überführung einer spezialisierten Datenstruktur (OCM) in ein etabliertes SBOM-Format (CycloneDX) ab und weist dabei explorativen Charakter auf. Qualitative Erhebungsverfahren sind insbesondere dann geeignet, wenn explorative Fragestellungen im Zentrum stehen, die nicht auf Hypothesenprüfung, sondern auf Generierung von Einsichten und Konzepten abzielen (vgl. Mayring, 2015, S. 85 ff.). Die Methodenauswahl folgt deshalb dem „Appropriateness“-Prinzip, auch Gegenstandsangemessenheit genannt, der qualitativen Forschung nach Flick (2009), welches als „wesentliches Merkmal [...]“ (Flick, 2009, S. 14) qualitativer Forschung gilt, um die richtige Wahl angemessener Methoden und Theorien zu treffen (vgl. Flick, 2009, S. 14 f.).

Für diese spezifische Forschungssituation erweist sich eine methodische Triangulation aus Dokumentenanalyse und Experteninterviews als besonders geeignet. Die Dokumentenanalyse dient dabei der systematischen Erschließung der normativen Grundlagen beider Standards und ihrer technischen Spezifikationen (vgl. Rupp und SOPHIST, 2021, S. 86), während Experteninterviews das praktische Erfahrungswissen von Domänenexperten erschließen und kontextspezifische Anforderungen identifizieren (vgl. Rupp und SOPHIST, 2021, S. 141–145).

Abgrenzung zu alternativen Methoden. Die Entscheidung gegen andere etablierte Erhebungsmethoden (vgl. Rupp und SOPHIST, 2021, S. 141–145) begründet sich durch die spezifischen Charakteristika der Forschungsfrage und des Untersuchungsgegenstands:

- **Workshops und Gruppendiskussionen:** Gruppensettings erzeugen Interaktionseffekte (Dominanz, Konsensdruck) und sind für sensible, sicherheitsrelevante Inhalte weniger geeignet. Zudem erschwert die internationale Verteilung und die unterschiedlichen Zeitzonen der Experten die Durchführung. Einzelinterviews erlauben dagegen die ungestörte Entfaltung individueller Fragen bei gleichzeitiger thematischer Steuerung (vgl. Bogner u. a., 2014, S. 28; Aghamanoukjan u. a., 2009, S. 425).
- **Standardisierte Fragebögen:** Hohe Standardisierung restriktiert die kommunikative Entfaltung und erfasst primär leicht verbalisierbares *diskursives* Wissen. Für

spezifische Erfassung von *Prozess-* und *Deutungswissen* (in Kapitel 3.2.2 erläutert) sind sie ungeeignet (vgl. Bogner u. a., 2014, S. 53; Meuser und Nagel, 2009, S. 470, 472).

- **Beobachtende Verfahren:** Feldbeobachtung oder andere beobachtende Verfahren sind ressourcenintensiv und setzen stabile, zugängliche Untersuchungssituations voraus. Für explorative Fragestellungen mit verteilten Experten ist der Ertrag im Verhältnis zum Aufwand ungünstig. Leitfadengestützte Einzelinterviews ermöglichen hier eine effiziente, zugleich offene Erhebung (vgl. Bogner u. a., 2014, S. 23-24; Kaiser, 2021, S. 34).

Begründung für semi-strukturierte Experteninterviews. Die Wahl semi-strukturierter, leitfadengestützter Experteninterviews als primäre Erhebungsmethode begründet sich durch folgende Punkte:

- **Explorative Forschungssituation:** Für ein bisher wenig theorisiertes und empirisch kaum erschlossenes Problemfeld sind offene, rekonstruktive Zugänge erforderlich (vgl. Flick, 2009, S. 156 ff.), weil explorative Studien in dieser Situation unternommen werden, wenn zu dem wissenschaftlich relevanten Problembereich bisher keine gesicherten theoretischen Annahmen oder belastbare empirische Daten vorliegen (vgl. Kaiser, 2021, S. 34). Semi-strukturierte Interviews ermöglichen Feldorientierung, Präzisierung des Problembewusstseins und Hypothesengenerierung bei zugleich geringer Vorstrukturierung (vgl. Bogner u. a., 2014, S. 23-24). Sie entfalten, im Vergleich zu strukturierten Interviews, die subjektive Perspektive der Befragten und erfassen kontextgebundene Bedeutungen (vgl. Aghamanoukjan u. a., 2009, S. 425).
- **Komplexität und Wissensarten:** Relevantes Expertenwissen in Software-Supply-Chain-Security und SBOM-Management umfasst neben explizitem Fachwissen insbesondere *Prozesswissen* (präreflexives Erfahrungswissen) und *Deutungswissen* (perspektivische Relevanzsetzungen). Dieses Wissen lässt sich nicht standardisiert „abfragen“, sondern muss im Gespräch rekonstruiert werden. Genau hier ist das offene Leitfadeninterview passend (vgl. Meuser und Nagel, 2009, S. 470, 472; Bogner u. a., 2014, S. 19).
- **Stakeholder-Heterogenität und internationale Kontexte:** Die in Kapitel 4.1 identifizierten Stakeholder vertreten divergierende Perspektiven und Prioritäten. Einzelinterviews mit flexiblem Leitfaden erlauben es, diese Relevanzstrukturen ohne gruppendifferenzielle Verzerrungen differenziert zu erfassen (vgl. Bogner u. a., 2014, S. 28; Aghamanoukjan u. a., 2009, S. 425; Rupp und SOPHIST, 2021, S. 144 f.). Die methodische Anlage des Experteninterviews ist dabei für unterschiedliche *institutionelle* und *internationale* Kontexte gleichermaßen anschlussfähig, sodass dieses bei

den später identifizierten Stakeholdern Anwendung finden kann (vgl. vgl. Meuser und Nagel, 2009, S. 470-474).

Dokumentenanalyse als komplementäre Erhebungsmethode Während Flick (2009, S. 444 f.) Triangulation als methodisches Grundprinzip der Perspektivenerweiterung hervorhebt, betont Mayring (2015, S. 68), dass auch unterschiedliche Materialien in einer gemeinsamen Kategorienbildung konsolidiert werden können.

Die Dokumentenanalyse wurde *vorgelagert* zur Interviewphase durchgeführt, um aus den Spezifikationen *deduktive Startkategorien* für die spätere qualitative Inhaltsanalyse abzuleiten. Zugleich diente sie als Grundlage für die Entwicklung des Interviewleitfadens (siehe Abschnitt 3.2.2). Methodisch orientiert sich dieses Vorgehen an der Dokumentenanalyse nach Bowen (2009) und der deduktiven Kategorienbildung im Sinne von Mayring (2015). In einem *nachgelagerten* Schritt dient dieselbe Dokumentenbasis als *normativer Prüfanker* innerhalb der Triangulation, um die aus Experteninterviews gewonnenen Informationen auf Vollständigkeit und Normkonformität zu spiegeln (vgl. Flick, 2009, 444 f.).

Somit erfüllt die Dokumentenanalyse eine dreifache Funktion: sie liefert

- deduktive Analysekategorien im Sinne von Mayring (2015, S. 68),
- strukturiert den Interviewleitfaden, der die Erhebung praxisnaher Kernthemen aus den Spezifikationen systematisch in die Befragung integriert,
- und fungiert nach Abschluss der Interviewauswertung als Prüfanker im Rahmen der Triangulation (siehe Abschnitt 3.3.2).

Die Dokumentenanalyse umfasst folgende Materialien:

- **OCM-Spezifikation:** Analyse der Component Descriptor Spezifikation, Datenmodelle und Schema-Definitionen
- **CycloneDX-Standard:** Untersuchung der SBOM-Spezifikation, Formatvarianten (JSON, XML) und Validierungsregeln
- **Tool-Dokumentationen:** Analyse bestehender OCM- und CycloneDX-Tools bezüglich Schnittstellen und Kompatibilitätsanforderungen
- **Regulatorien:** Berücksichtigung relevanter Regulatorien für SBOMs, wie die NIS2-Richtlinie (EU 2022/2555), der CRA und die Executive Order 14028

Da der Untersuchungsfokus auf einem PoC liegt und der Schwerpunkt auf der Erhebung praxisrelevanter Anforderungen durch Experteninterviews liegt, wurde bewusst auf eine formale, regelgeleitete Vollaralyse nach Bowen (2009) verzichtet. Stattdessen erfolgt eine gezielte formlos-dokumentarische Analyse, die sich an den funktionalen und technischen Anforderungen der Konvertierung orientiert (siehe Kapitel 4.5).

3.2 Ausgestaltung der gewählten Erhebungsmethoden

Die vorliegende Arbeit nutzt die Stakeholderanalyse als übergeordneten Rahmen. Sie begründet die theoriebasierte Auswahl von Experten und identifiziert gezielt jene Personen, deren spezifisches Wissen und Erfahrungen einen hohen Informationsgehalt versprechen (Flick, 2009, S. 30 f., 109, 123).

3.2.1 Stakeholderanalyse zur Expertenauswahl

Freeman (1984, S. 46 ff.) definiert Stakeholder als „any group or individual who can affect or is affected by the achievement of the organization's objectives“. Diese Definition ist bewusst weit gefasst, um sicherzustellen, dass keine potenziell relevante Partei außer Acht gelassen wird (vgl. Freeman, 1984, S. 52).

Zur Priorisierung der Relevanz konkurrierender Stakeholder-Ansprüche greift die Arbeit auf das Salienzmodell von Mitchell u. a. (1997) zurück. Mitchell u. a. (1997, S. 854) erweitern Freemans Konzept, indem sie Kriterien zur Bestimmung der Salienz von Stakeholdern vorschlagen, also des Grades, in dem konkurrierenden Stakeholder-Ansprüchen Priorität eingeräumt wird. Sie identifizieren drei Hauptattribute:

- **1. Macht (Power):** Die Fähigkeit eines Stakeholders, seinen Willen gegenüber einer Organisation durchzusetzen (vgl. Mitchell u. a., 1997, S. 865). Für die Expertenauswahl ist dies relevant, da Personen mit hoher Machtposition oft Zugang zu strategisch wichtigen Informationen und Entscheidungsprozessen haben, die für die Forschungsfrage von hoher Bedeutung sind.
- **2. Legitimität (Legitimacy):** Die verallgemeinerte Wahrnehmung, dass die Handlungen eines Stakeholders wünschenswert, angemessen oder passend sind und bezieht sich somit auf die Berechtigung des Anspruchs (vgl. Mitchell u. a., 1997, S. 866). Experten, deren Wissen und Erfahrungen als legitim und glaubwürdig wahrgenommen werden, sind für eine theoriebasierte Auswahl unerlässlich.
- **3. Dringlichkeit (Urgency):** Das Ausmaß, in dem ein Anspruch sofortige Aufmerksamkeit erfordert. Dringlichkeit wird als katalytische Komponente beschrieben, die Aufmerksamkeit fordert (vgl. Mitchell u. a., 1997, S. 867).

Die Kombination dieser Attribute führt zu unterschiedlichen Salienzstufen und damit zu einer Priorisierung der Expertenrollen (vgl. Mitchell u. a., 1997, S. 873). Stakeholder, die lediglich eines der drei Attribute besitzen, werden als „Latente Stakeholder“ bezeichnet. Stakeholder, die zwei dieser Attribute besitzen, werden als „expectant stakeholders“ mit moderater Salienz betrachtet (vgl. Mitchell u. a., 1997, S. 876). Insbesondere „Dominant Stakeholders“ (Macht und Legitimität) sind von hoher Bedeutung, da ihre Erwartungen maßgeblich sind (vgl. Mitchell u. a., 1997, S. 876). Außerdem werden Stakeholder mit allen

drei Attributen „Definitive Stakeholder“ genannt und weisen somit die höchste Salienz auf (vgl. Mitchell u. a., 1997, S. 878). Die genaue Zuordnung kann in **Abbildung 3.1** abgelesen werden.

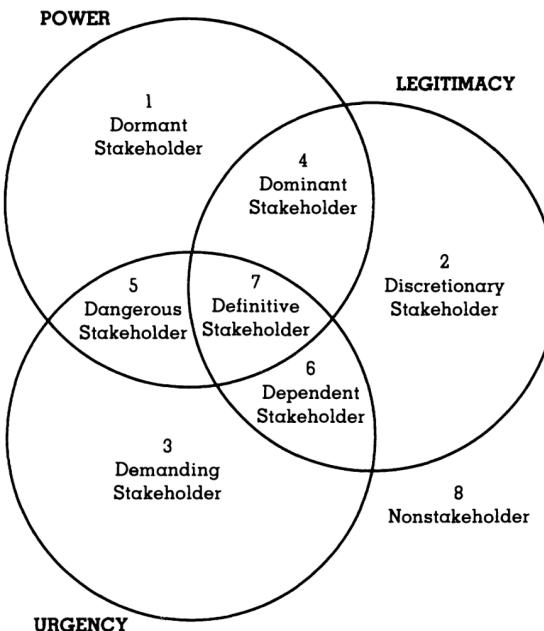


Abb. 3.1: Stakeholder-Typologie: eins, zwei oder drei Attribute
(vgl. Mitchell u. a., 1997, S. 874)

Im Kontext der qualitativen Forschung ist die Auswahl von Experten kein statistischer, sondern ein strategischer Prozess, der als Purposive Sampling (Zweckgerichtetes Sampling) bezeichnet wird (vgl. Flick, 2009, S. 109). Flick (2009, S. 30 f., 123) betont, dass das Ziel des Purposive Samplings nicht die statistische Repräsentativität einer größeren Population ist, sondern die Auswahl von Fällen, die reich an relevanten Informationen sind.

In der qualitativen Methodik bezeichnet „Fall“ die Einheit der Analyse (vgl. Flick, 2009, S. 134). Das kann zum Beispiel eine Person, ein Team oder eine Organisation sein (vgl. Flick, 2009, S. 134). Folglich ist das Interview das Mittel mit dem Daten über einen Fall gesammelt werden, denn das Wissen des Interviewpartners gehört der Person, somit ist die Person die Einheit die analysiert wird (vgl. Flick, 2009, S. 124).

Die Logik dahinter ist explizit Fälle auszuwählen, welche zur Maximierung des Informationsgehalts und der theoretischen Abdeckung beitragen (vgl. Flick, 2009, S. 123).

- **Informationsgehalt:** Experten werden ausgewählt, weil sie über das notwendige Wissen und die Erfahrung im Untersuchungsfeld verfügen und in der Lage sind, dies zu reflektieren und zu artikulieren (vgl. Flick, 2009, S. 123). Es geht darum, „welche Fälle?“ und nicht „wie viele?“ (vgl. Flick, 2009, S. 31).

- **Theoretische Abdeckung:** Die Auswahl zielt darauf ab, entweder eine breite Abdeckung des Feldes in seiner Vielfalt zu erreichen oder tiefe Einblicke in spezifische Aspekte zu gewinnen, die für die Theorieentwicklung relevant sind (vgl. Flick, 2009, S. 123). Dies kann auch durch die gezielte Suche nach extremen, abweichenden oder kritischen Fällen erfolgen, um die Variation der Bedingungen zu verstehen (vgl. Flick, 2009, S. 122, 130).

Zusammenfassend lässt sich sagen, dass die Expertengewinnung mit einer umfassenden „Longlist“ potenziell relevanter Gruppen auf Grundlage der breiten Stakeholder-Definition nach Freeman (1984) beginnt. Anschließend wird diese Liste durch die systematische Priorisierung nach Mitchell u. a. (1997) reduziert. Auf Basis dieser Gruppen kann schlussendlich die Auswahl des Falls im Sinne des Purposive Samplings durchgeführt werden (siehe Kapitel 4.1).

3.2.2 Semi-strukturiertes, leitfadengestütztes Experteninterview

Das Experteninterview ist die mit am häufigsten eingesetzte Methode der empirischen Forschung und wird methodologisch im Kontext des semi-strukturierten Interviews verortet (vgl. Meuser und Nagel, 2009, S. 465; Bogner u. a., 2014, S. 27). Es unterscheidet sich beispielsweise fundamental von standardisierten Fragebögen, da es nicht primär numerisch Variablen, sondern textuelle, kontextgebundene Bedeutungen erhebt (vgl. Aghamanoukjan u. a., 2009, S. 417).

Die Wahl des qualitativen, semi-strukturierten Ansatzes ist eine Folge der methodischen Unzulänglichkeit anderer Verfahren zur Erfassung von Expertenwissen. Das strukturierte Interview, auch standardisiertes Interview genannt, folgt einem strikten Fragenkatalog mit festgelegter Reihenfolge und nutzt dabei oft geschlossene Fragen, sodass es meist nur in der quantitativen Forschung Anwendung findet (vgl. Aghamanoukjan u. a., 2009, S. 421; Kaiser, 2021, S. 278). Dies ist für diese Arbeit jedoch ungeeignet, da sonst die Kommunikationsform restriktiert wird (vgl. Bogner u. a., 2014, S. 53) und so lediglich Wissen auf der Ebene des diskursiven Bewusstseins, also explizites, leicht kommunizierbares Wissen, erfasst werden kann (vgl. Meuser und Nagel, 2009, S. 425). Die genaue Begründung wurde jedoch schon am Anfang des Kapitels 3 geliefert.

Das offene Interview ist somit das angemessene Instrument, um die Mischung aus praktischem Erfahrungswissen und Deutungswissen, also subjektiven Sichtweisen (vgl. Meuser und Nagel, 2009, S. 470-472), zu erheben. Da es nicht wie standardisierte Fragen, die Reihenfolge und Themen einschränkt oder gar die spezifische Sichtweise des Interviewpartners verdeckt (vgl. Flick, 2009, S. 171). Folglich ermöglicht das Experteninterview, wie beschrieben, dem Interviewtem, tiefgreifendes und funktionsbezogenes Wissen preiszugeben, in dem er erläutert und Beispiele nennt (vgl. Meuser und Nagel, 2009).

Charakteristisch ist dabei dennoch der Einsatz eines Interviewleitfadens, der das Gespräch zwar strukturiert, aber es nicht in ein starres Schema zwingt.

Kernfunktionen des Leitfadens. Dieser fungiert laut Kaiser (2021, S. 64) als Übersetzung der Forschungsfragen in die Erfahrungswelt der Befragten und erfüllt dabei folgende Kerfunktionen:

- **Strukturierung des Gesprächs:** Der Leitfaden definiert Themenblöcke und Hauptfragen, die dem Interview eine nachvollziehbare Argumentationslogik verleihen. In der Regel von allgemeinen Fragen zu spezielleren Fragen (vgl. Kaiser, 2021, S. 65).
- **Rahmenbedingungen:** Er informiert über Ziel, Vorgehen, mögliche Anonymisierung und Datenschutz und schafft so Transparenz gegenüber den Befragten (vgl. Kaiser, 2021, S. 66).
- **Kompetenznachweis:** Er dokumentiert die Einarbeitung des Forschenden und stützt dessen Rolle als fachlich anschlussfähiger Gesprächspartner (vgl. Meuser und Nagel, 2009, S. 473).

Aufbau und Fragetypen Ein Leitfaden ist in Themenblöcke gegliedert, die jeweils durch Hauptfragen eröffnet und durch Unterfragen vertieft werden können (Bogner et al. 2014: 28). Typische Fragetypen, als Gesprächsanreize, sind

- **Einleitungsfragen**, die den Einstieg erleichtern und den Gesprächsfluss in Gang setzen,
- **strukturierende Fragen**, die den Übergang zwischen Themenbereichen steuern,
- **direkte Fragen**, die gezielt notwendige Sachverhalte adressieren (vgl. Kaiser, 2021, S 77-78).

Trotz des semi-strukturierten Charakters kann es notwendig sein, den Leitfaden an spezifische Gesprächspartner anzupassen, besonders wenn diese sich in ihrer beruflichen Position oder Disziplin unterscheiden (vgl. Bogner u. a., 2014, S. 30). In diesem Fall kann ein allgemeiner Basisleitfaden als Grundlage für speziellere Fassungen dienen (vgl. Bogner u. a., 2014, S. 30). Bei theoriegenerierenden Interviews (die auf Deutungswissen abzielen) ist die spontane Äußerung des Experten vorzuziehen, weshalb der Leitfaden hier tendenziell offener und lockerer ist als bei systematisierenden Interviews (vgl. Bogner u. a., 2014, S. 30).

Das Ziel des Experteninterviews ist schlussendlich ein „[...] ungezwungenes Fachgespräch [...]“ (Kaiser, 2021, S. 94), das Schilderungen und Erzählungen hervorbringt und nicht die wörtliche Transformation der Forschungsfrage (vgl. Bogner u. a., 2014, S. 33-34).

3.3 Auswertungsmethodik

Bislang existiert kein allgemein anerkanntes und standardisiertes Verfahren zur systematischen Analyse von Experteninterviews (vgl. Bogner u. a., 2014, S. 71). In der Literatur werden unterschiedliche methodische Ansätze zur Auswertung von Experteninterviews diskutiert (vgl. Meuser und Nagel, 2009, S. 476; Mayring, 2015, S. 11 ff.). Diese Herangehensweisen lassen sich grundsätzlich dem Bereich der qualitativen Inhaltsanalyse zuordnen (vgl. Kaiser, 2021, S. 106). Obwohl die Auswahl des Auswertungsverfahrens stark von den jeweiligen Forschungsinteressen abhängt, lassen sich für Experteninterviews dennoch methodische Präferenzen ableiten, die sich an der spezifischen Rolle des Interviews im Forschungsdesign orientieren (vgl. Bogner u. a., 2014, S. 71). Für informatorische Interviews bietet sich vor allem zur Auswertung die qualitative Inhaltsanalyse an (vgl. Bogner u. a., 2014, S. 71 f.). Diese Methode wurde für Erhebungssituationen entwickelt, in denen in erster Linie Texte und nicht quantitative Daten gewonnen werden (vgl. Aghamanoukjan u. a., 2009, S. 417).

Vor diesem Hintergrund wird in dieser Arbeit die *qualitative Inhaltsanalyse* (QIA) nach Mayring (2015) eingesetzt, da sie für textbasierte Interviewdaten ein regelgeleitetes, transparentes Vorgehen mit deduktivem Start und kontrollierter induktiver Kategorienbildung bereitstellt und so eine intersubjektiv nachvollziehbare Rekonstruktion von Prozess- und Deutungswissen ermöglicht (vgl. vgl. Mayring, 2015, S. 11 ff.). Diese wird im nachfolgenden Abschnitt genauer erläutert.

3.3.1 Qualitative Inhaltsanalyse nach Mayring

Die QIA erlaubt eine strukturierte und methodisch kontrollierte Auswertung von Textmaterial, sodass die Ergebnisse sowohl nachvollziehbar als auch überprüfbar bleiben (vgl. Mayring, 2015, S. 51 f.). Das Ziel der Auswertung besteht darin, die Interviewtranskripte so aufzubereiten, dass sie als belastbare Datenbasis für die Beantwortung der Forschungsfrage dienen können (vgl. Bogner u. a., 2014, S. 73). Ein wesentliches Anliegen ist es dabei, sprachliches Material strukturiert zusammenzufassen (vgl. Mayring, 2015, S. 47) und das untersuchte Material anhand eines festgelegten Kategoriensystems systematisch zu klassifizieren (vgl. Bogner u. a., 2014, S. 73).

Nach Mayring (2015, S. 61) erfolgt zunächst die Auswahl des zu analysierenden Materials und dessen Überprüfung hinsichtlich inhaltlicher Relevanz. Dabei ist der Entstehungskontext bedeutsam, um die Bedingungen der Datenerhebung angemessen einordnen zu können. Im Anschluss wird die Beschaffenheit des Materials präzise beschrieben. Die Analyse folgt einer leitenden Fragestellung, auf deren Grundlage eine geeignete Analysemethode ausgewählt wird. Schließlich erfolgt die Auswertung nach einem systematischen Ablaufplan. Grundsätzlich lassen sich drei zentrale Analysetechniken (vgl. Mayring, 2015, S. 66)

unterscheiden:

Zusammenfassung. Ziel dieser Technik ist es, das Ausgangsmaterial so zu reduzieren, dass lediglich die wesentlichen Inhalte erhalten bleiben. Auf diese Weise entsteht eine prägnante, aber zugleich aussagekräftige Darstellung.

Explikation. Hierbei wird der Text durch zusätzliche Hintergrundinformationen ergänzt, um bestimmte Passagen genauer zu erläutern und ein besseres Verständnis zu ermöglichen.

Strukturierung. Das Material wird nach zuvor definierten Kriterien systematisch geordnet, sodass gezielt bestimmte inhaltliche Aspekte herausgearbeitet werden.

Die drei Techniken stehen unabhängig voneinander und sollten jeweils in Bezug auf die Forschungsfrage sowie die Beschaffenheit des vorliegenden Materials gezielt ausgewählt werden (vgl. Mayring, 2015, S. 66).

In dieser Arbeit wird die Methode der Zusammenfassung angewendet, denn in erster Linie stehen die inhaltlichen Kernaussagen der Interviews im Fokus, um die Anforderungen abzuleiten. Nach Mayring (2015, S. 71) ist der Ablauf dieser Vorgehensweise in Abbildung 3.2 dargestellt und umfasst folgende Schritte:

Schritt 1: Zu Beginn wird die **Analyseeinheit festgelegt**. Dabei kennzeichnet die Kontexteinheit den größten Textabschnitt, der einer Kategorie zugeordnet werden kann, während die Auswertungseinheit die Reihenfolge bestimmt, in der die Textsegmente bearbeitet werden.

Schritt 2: Anschließend folgt die **Paraphrasierung relevanter Textstellen**, bei der nebенächliche Inhalte entfernt und die Kernaussagen in eine einheitliche Ausdrucksform überführt werden.

Schritt 3: Daraufhin wird das **gewünschte Abstraktionsniveau** definiert, anhand dessen die Paraphrasen generalisiert werden. Hier werden Inhalte und Begriffe an den festgelegten Abstraktionsgrad angepasst. Die *erste Reduktion* zielt darauf ab, sinngleiche Paraphrasen zu eliminieren und somit nur bedeutsame Inhalte zu erhalten. In der zweiten Reduktion werden Paraphrasen mit ähnlichem Gehalt gebündelt und weiter verdichtet. Dabei können auch theoretische Vorannahmen zur Strukturierung herangezogen werden.

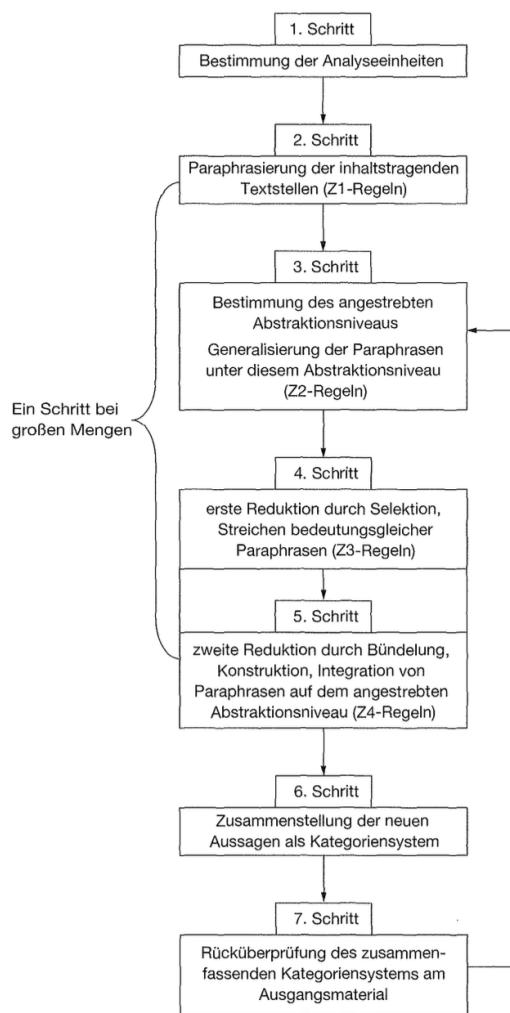


Abb. 3.2: Ablaufmodell zusammenfassender Inhaltsanalyse
(vgl. Mayring, 2015, S. 70)

Ein zentraler Bestandteil dieser Vorgehensweise ist zudem die Entwicklung eines **induktiven Kategoriensystems**, das im wechselseitigen Prozess zwischen theoretischer Fragestellung und den Inhalten des Materials ständig weiterentwickelt wird (vgl. Mayring, 2015, S. 85 f.).

Einzelne Textstellen werden entweder bestehenden Kategorien zugeordnet oder führen zur Bildung neuer Kategorien oder Subkategorien, sofern keine passende vorhanden ist (vgl. Mayring, 2015, S. 85 f.). Die **Abbildung 3.3** zeigt ein Ablaufmodell zur dynamischen Kategoriengestaltung. Die genau Anwendung und Erklärung dieser findet sich in Kapitel 4.4.

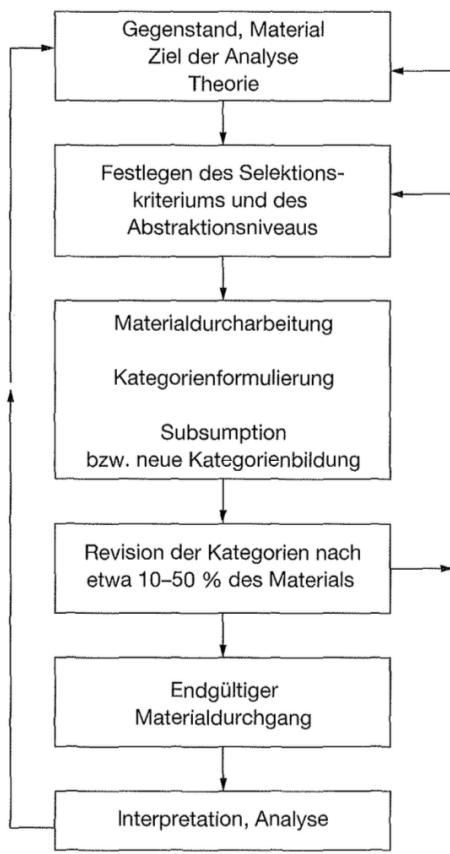


Abb. 3.3: Prozessmodell induktiver Kategorienbildung
(vgl. Mayring, 2015, S. 70)

3.3.2 Methodische Triangulation und Anforderungskatalog-Erstellung

Die systematische Zusammenführung der Erkenntnisse aus Dokumentenanalyse und Experteninterviews erfolgt durch eine methodische Triangulation. Ziel ist es, die Qualität und Validität des entstehenden Anforderungskatalogs abzusichern und so eine robuste Grundlage für die PoC-Entwicklung zu schaffen. Triangulation wird in der qualitativen Forschung als ein zentrales Verfahren verstanden, um Befunde zu validieren und zugleich verschiedene Dimensionen eines Untersuchungsgegenstandes sichtbar zu machen (vgl. Flick, 2009, S. 405, 444–445).

Methodische Triangulation. In dieser Arbeit wird Triangulation schlank und zweckgebunden umgesetzt:

1. Die *formlose Dokumentenanalyse* der OCM- und CycloneDX-Spezifikationen liefert deduktive Startkategorien für die anschließende QIA der Experteninterviews (vgl. Mayring, 2015, S. 11 ff.).
2. Nach der Interviewauswertung dient dieselbe Dokumentenbasis hier erneut als Prüfanker, um den konsolidierten Anforderungskatalog auf Vollständigkeit und Spezifi-

kationskonformität zu verifizieren und, wo nötig, zusammenzuführen. Dieses Vorgehen folgt dem Verständnis von Triangulation als Validierungsstrategie zur Erhöhung der Befundqualität durch die geplante Kombination komplementärer Zugänge (vgl. Flick, 2009, S. 444–445).

Im nächsten Abschnitt wird darauf aufbauend die Erstellung der Anforderungen nach Rupp und SOPHIST (2021) sowie deren Ausrichtung an *ISO/IEC/IEEE 29148* (2018) dargestellt.

Strukturierung des Anforderungskatalogs. Der resultierende Anforderungskatalog wird entsprechend der etablierten RE-Taxonomie nach Rupp und SOPHIST (2021, S. 165 ff.) abgespeckt strukturiert, um eine ausreichende Anforderungsanalyse für einen PoC zu erreichen.

Begriffsdefinition. *Funktionale Anforderungen* (FANF) beschreiben eine Funktion oder ein Verhalten des IT-Systems und legen fest, was das System tun soll (vgl. Herrmann, 2022, S. 37). Sie definieren jene Leistungen oder Funktionen, die das System unter bestimmten Bedingungen einer nutzenden Person oder einem Nachbarsystem zur Verfügung stellt (vgl. Rupp und SOPHIST, 2021, S. 31).

Nicht-funktionale Anforderungen (NANF) fassen den gesamten Rest der Anforderungen zusammen (vgl. Rupp und SOPHIST, 2021, S. 31) und bestimmen das „Wie“ der Funktionsausführung, indem sie die Qualität des Systems definieren (vgl. Herrmann, 2022, S. 37). Sie dienen dazu, die Lösung einzuschränken und legen grundlegende Eigenschaften fest, die im Architekturentwurf berücksichtigt werden müssen (vgl. Herrmann, 2022, S. 37). Typischerweise werden nicht-funktionale Anforderungen in Qualitätsanforderungen (wie Performance oder Verfügbarkeit) und Anforderungen an die Technologie unterteilt (vgl. Rupp und SOPHIST, 2021, S. 31).

Formulierung des Anforderungskatalogs. Zur konsistenten Formulierung des Anforderungskatalogs werden zwei Vorlagen genutzt: Der FunktionsMASTER (siehe Abbildung 3.4) für funktionale Anforderungen und der EigenschaftsMASTER (siehe Abbildung 3.5) für nicht-funktionale Anforderungen. Diese dienen der Priorisierung, also der Wichtigkeit, der Anforderung und werden durch „muss“, „sollte“, „wird“ differenziert (vgl. Rupp und SOPHIST, 2021, S. 368). Rupp und SOPHIST (2021, S. 32) sind hier der Meinung, denn es gäbe zwar noch weitere, wie beispielsweise „kann“, dass diese drei Schlüsselwörter ausreichen, um gute Anforderungen festzulegen. Auch für diese Arbeit ist diese Anforderungsdefinition ausreichend, denn sie dient lediglich zum Erstellen und Validieren des PoC. Deshalb wird auch keine juristische Verbindlichkeit erwartet. Eine juristische Verbindlichkeit wäre außerdem schon durch lediglich die drei Schlüsselwörter nach Rupp und

SOPHIST (2021, S. 32) erreicht.

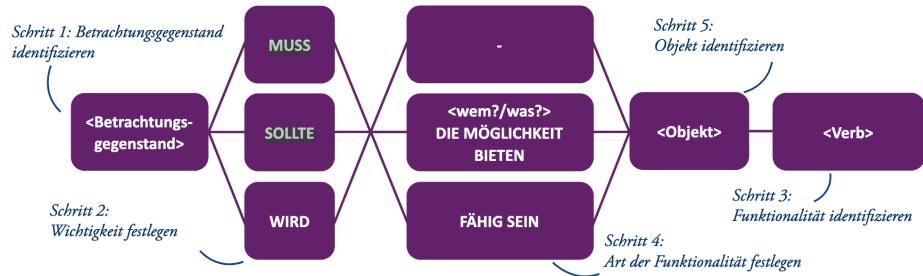


Abb. 3.4: FunktionsMASTER
(vgl. Rupp und SOPHIST, 2021, S. 361)

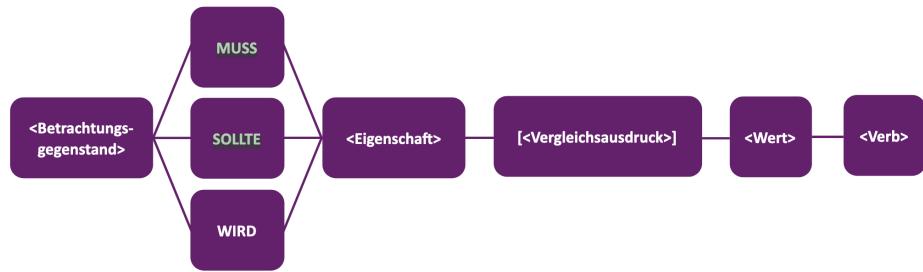


Abb. 3.5: EigenschaftsMASTER
(vgl. Rupp und SOPHIST, 2021, S. 378)

Die **Abbildungen 3.4 und 3.5** sind bewusst ohne Bedingung dargestellt, um die Grundform klar zu halten. Bedingungen, wie Auslöser und Randfälle, werden nur ergänzt, wenn sie das Verhalten tatsächlich steuern oder die Messung beeinflussen. Das detaillierte Regelwerk kann bei Rupp und SOPHIST (2021) nachgelesen werden.

4 Anforderungsanalyse

4.1 Auswahl der Interviewpartner

Die Auswahl folgt den in **Abschnitt 3.2.1** dargelegten Grundsätzen und fokussiert wenige, salienzstarke Experten mit hohem Informationsgehalt. Nachfolgend wird das operative Vorgehen schrittweise dokumentiert.

Stakeholderidentifikation nach Freeman (1984). Auf Basis der Quellenrecherche und gestützt auf die breite Stakeholder-Definition (siehe Abschnitt 3.2.1) wurde eine personenungebundene Longlist potenziell relevanter Gruppen gebildet:

- *SBOM-Standardgeber* (CycloneDX-Core Working Group, ECMA TC54),
- *OCM-Modellverantwortliche* (OCM-Maintainer),
- *SBOM-Producer* (Build/Release-Engineering),
- *SBOM-Consumer/Scanner* (Grype, Trivy),
- *Infrastruktur* (OCI/Registry-Betreiber),
- *Regulatorik/Audit* (NTIA),
- *Intermediäre Tools* (Konverter/Parser-Bibliotheken).

Salienzbewertung nach Mitchell u. a. (1997). Für alle Gruppen der Longlist wurden die Attribute *Macht* (P), *Legitimität* (L) und *Dringlichkeit* (U) bewertet (siehe Abschnitt 3.2.1). Da Mitchell u. a. (1997) die Attribute als variabel und nicht binär verstehen und explizit zur Entwicklung „operational definitions“ aufrufen (vgl. Mitchell u. a., 1997, S. 881), werden diese Attribute in dieser Arbeit zusätzlich auf einer dreistufigen Skala von **0–2** bewertet (0 = nicht vorhanden, 1 = kontextabhängig, 2 = stark ausgeprägt). Die Einstufungen werden jeweils kurz begründet:

- *SBOM-Standardgeber (CycloneDX)*: P=2, L=2, U=1
(Spezifikations-Hoheit; hohe Normautorität; Dringlichkeit bei sicherheitsgetriebenen Schemaanpassungen)
- *OCM-Modellverantwortliche*: P=2, L=2, U=1
(Feldsemantik des Quellmodells (Descriptor); hohe Gestaltungsmacht und Legitimität)
- *SBOM-Producer*: P=1, L=1, U=0
(erzeugen Artefakte, definieren aber nicht die Zielsemantik)
- *SBOM-Consumer/Scanner*: P=2, L=2, U=1
(de-facto-Macht durch Toolpraxis und Marktadoption; legitime Sicherheitsanforderungen; hohe operative Dringlichkeit (Vulnerabilities))
- *Infrastruktur*: P=1, L=1, U=0

(Bereitstellungs- und Distributionsmacht, aber keine Spezifikationshoheit über SBOM-/SBoD-Semantik)

- *Regulatorik/Audit*: P=1, L=2, U=0
(setzen Rahmen und Pflichten (hohe Legitimität), jedoch indirekte Wirkung auf konkrete Feldsemantik)
- *Intermediäre Tools (Konverter/Parser-Bibliotheken)*: P=1, L=1, U=0
(beeinflussen Praktiken, jedoch ohne normative Autorität)

Die Bewertung der Attribute erfolgte stringent im Lichte der Forschungsfrage. Somit weisen die folgenden Gruppen **CycloneDX**, **OCM** und **Konsument/Scanner** eine *hohe Salienz* auf. Aus diesen drei Gruppen erfolgte die Besetzung durch Experten per *Purposive Sampling*.

Purposive Sampling nach Flick (2009). Zur Maximierung des *Informationsgehalts* und der *theoretischen Abdeckung* wurden vorab drei Fälle ausgewählt, je ein Fall (Person/-Experte) pro Kerngruppe (Im Weiteren Verlauf so genannt: *OCM*, *CycloneDX*, *Scanner*). Die Auswahl und Eignung dieser ausgewählten Experten ist nachfolgend Erläutert:

- **Jakob Möller (Lead Architekt des OCM, SAP)** *Begründung der Auswahl*: Die SAP ist der Haupttreiber und Maintainer des Open Component Models. Möller ist als Lead Architekt direkt an der Entwicklung und den konzeptionellen Grundlagen von OCM beteiligt. Seine Expertise ist von unschätzbarem Wert, da er Einblicke in die Semantik, die Designentscheidungen und die Vision hinter OCM geben kann, die für die präzise Überführung der Datenstruktur unerlässlich sind. Außerdem gilt er auch als Auftraggeber dieser Forschung.
- **Steve Springett (Chair of CycloneDX SBOM Standard, Chair Ecma TC54)** *Begründung der Auswahl*: Springett ist Mitbegründer des CycloneDX-Standards und hat diesen maßgeblich geprägt. Sein tiefes Verständnis für die Spezifikation sind entscheidend, um die Anforderungen an ein konvertiertes SBOM zu verstehen. Seine Aussagen stellen die höchste normative Autorität dar und stellen sicher, dass die Anforderungen des PoC den Standards entsprechen. Die Tatsache, dass er auch der Hauptentwickler von Dependency-Track ist, einem prominenten SBOM-Scanners, stärkt seine Eignung zusätzlich.
- **Josh Bressers (Vice President of Security, Anchore)** *Begründung der Auswahl*: Anchore ist ein führender Anbieter von Software-Supply-Chain-Security-Tools, insbesondere im Bereich der SBOM-Erstellung und -Analyse mit Tools wie *Syft* und *Grype*. Bressers verkörpert die Perspektive eines *SBOM Konsumenten* und kann bewerten, welche Informationen in einem CycloneDX-SBOM für eine praktische Sicherheitsanalyse relevant und nutzbar sind. Seine Einblicke sind entscheidend, um die Kompatibilität des Konvertierungsergebnisses mit den Werkzeugen zu gewährleisten, die in der

Branche de facto eingesetzt werden.

Methodische Legitimation der kleinen Stichprobe. Die Fallzahl ($n=3$) ist damit für die angestrebte semantische Präzisierung hinreichend, da die qualitative Inhaltsanalyse vor allem *regelgeleitete Nachvollziehbarkeit und Interpretationsqualität* zum Ziel hat (vgl. Flick, 2009, S. 31, 123, 130; Mayring, 2015, S. 12 f.).

4.2 Erstellung des Interviewleitfadens

Auf Grundlage der theoretischen Ausführungen in Kapitel 3.2.2 und der vorab durchgeführten Expertenauswahl (Stakeholderanalyse) wurde ein semi-strukturierter Interviewleitfaden entwickelt. Dieser dient dazu, die Forschungsfrage in die Erfahrungswelt der Befragten zu übersetzen und ein „[...] ungezwungenes Fachgespräch [...]“ (Kaiser, 2021, S. 94) zu ermöglichen, um die technischen Anforderungen an den PoC zu identifizieren. Diese Anforderungen sollten widerspiegeln, wie eine Konvertierung zwischen SBoD (OCM) und SBOM (CycloneDX) ohne Informationsverlust erreicht werden kann und welche Qualitätskriterien für die resultierenden SBOMs gelten.

Der Leitfaden gliedert sich in fünf Hauptteile:

1. **Einstieg** mit Begrüßung, Vorstellung des Projekts, Erläuterung von Rahmenbedingungen, Einholung der Aufzeichnungseinwilligung und für CycloneDX- und Scanner-Experten eine fünf minütige Erklärung des OCMs per Screen-Share,
2. **Hintergrundfragen** zur Einordnung der Expertise der Befragten,
3. **Kernfragen**, die zentrale Aspekte der Transformation von OCM *Component Descriptors* in CycloneDX-SBOMs adressieren,
4. **gruppenspezifische Vertiefungen**, die je nach Expertengruppe (OCM, CycloneDX, Scanner) eingesetzt wurden,
5. sowie **Abschlussfragen** zur Reflexion und Einordnung.

Dieser Aufbau folgt der methodischen Empfehlung, durch offene Einstiegs- und strukturerende Fragen den Gesprächsfluss zu fördern, während direkte Fragen gezielt notwendige Sachverhalte adressieren (vgl. Kaiser, 2021, S. 77–78).

Inhaltliche Ableitung und Anpassungslogik. Die Themenfelder wurden inhaltlich aus den theoretischen Grundlagen (siehe Kapitel 2) abgeleitet. Die Kernfragen adressieren insbesondere die Identifikation verlustfreier Überführungen von OCM-Strukturen in CycloneDX (K3, K4), mögliche Workarounds (K4a) sowie Qualitätskriterien (K6). Um die unterschiedlichen Expertisen optimal zu nutzen, wurde der Leitfaden modular konzipiert: Ein Basisgerüst identischer Kernfragen gewährleistet die Vergleichbarkeit, während gruppenspezifische Vertiefungsblöcke je nach Expertise aktiviert wurden.

OCM-Experten erhielten zusätzliche Fragen zu Identitäten, Labels und Modellierungskonventionen, CycloneDX-Experten wurden zu Feldern, Hierarchiemodellierungen und geplanten Standarderweiterungen befragt, während Scanner-Experten gezielt nach Anforderungen und Limitationen bestehender Analysetools gefragt wurden. Diese Differenzierung entspricht dem Grundsatz, dass Leitfäden an die berufliche Position und Disziplin der Interviewpartner angepasst werden können (vgl. Bogner u. a., 2014, S. 30) und ermöglicht es, sowohl Modellierungsaspekte des OCM als auch die Praxistauglichkeit in CycloneDX-Tools abzudecken.

Damit wurde der semi-strukturierte Charakter des Experteninterviews umgesetzt, bei dem feste Leitfragen durch flexible Vertiefungsfragen ergänzt werden, um Deutungswissen in Form von Beispielen, Erzählungen und fachlichen Einschätzungen zu erfassen. Der vollständige Leitfaden mit allen Fragen ist in **Anhang A** dokumentiert.

4.3 Durchführung der Interviews

Die Interviews mit den ausgewählten Experten – Jakob Möller (2025a), Steve Springett (2025) und Josh Bressers (2025) – wurden methodisch stringent auf Basis des semi-strukturierten Leitfadens durchgeführt. Jedes der drei Gespräche hatte eine durchschnittliche Dauer von rund 45 Minuten, was einen umfassenden Austausch über die relevanten Fachthemen ermöglichte. Die Interviews fanden mit Springett (2025) und Bressers (2025) in englischer Sprache statt, während das Interview mit Möller (2025a) auf Deutsch geführt wurde.

Vorbereitung und Testlauf. Vor der eigentlichen Durchführung der Experteninterviews wurde ein Testlauf mit dem unternehmensseitigen Betreuer dieser Arbeit, Fabian Burth, durchgeführt. Da er selbst Entwickler und Maintainer des OCMs ist, bot der Testlauf die Möglichkeit, sowohl die Struktur des Interviewleitfadens, als auch die technische Präsentation des OCMs zu überprüfen und anzupassen. Dies stellte sicher, dass die externen Experten, Springett (2025) und Bressers (2025), schnell und effizient in das Thema und Konzept des OCMs eingeführt werden konnten, was sich als essenziell für den Gesprächsfluss und der Beantwortung der Fragen erwies.

Relevanz der Experten im Interview. Die Eignung der Interviewpartner erwies sich unmittelbar durch ihre ausgewiesene Leitungs- und Gestaltungskompetenz im Themenfeld: Möller (2025a) beschreibt sich als „Lead Architect des OCM“ und betont seine Verantwortung „das OCM Projekt [...] in der Open Source Schirmherrschaft unter [der] Neo-Nephos [Foundation] zu begleiten“, wobei er das OCM dort sowohl im „Technical Advisory Council und Technical Steering Committee Member“ repräsentiert, als auch „praktisch [...] beteiligt an der Mitgestaltung der OCM Kernspezifikation“ beteiligt ist (vgl. Möller,

2025a, 0:03:12–0:03:48). Springett (2025) charakterisiert sich als „original author [von CycloneDX]“ und den „chair of the core working group“ sowie „chair of ECMA TC54“, der internationalen Standardisierungsgruppe (vgl. Springett, 2025, 0:08:30–0:08:51). Bressers (2025) verweist auf seine langjährige Praxiserfahrung „since 2004“, bei unter anderem *Red Hat*, *Elastic* und nun als *Vice President of Security* bei *Anchore*, mit der aktuellen Verantwortung für die „supply chain security of the [...] Anchor software, [...] like Syft and Grype“ (vgl. Bressers, 2025, 0:01:57, 0:04:08–0:04:22).

Die Kombination aus formaler Autorität, tiefem Fachwissen und praktischer Anwendungserfahrung der drei Experten deckt alle relevanten Perspektiven ab. Von der Quellspezifikation (OCM) über die Zielspezifikation (CycloneDX) bis hin zu praktischen Scanner-Tools. Die Sättigung der gewonnenen Erkenntnisse war so umfassend, dass eine Ausweitung der Stichprobenfälle (Interviews) nicht erforderlich war.

4.4 Auswertung der Experteninterviews

Dieses Kapitel beschreibt das konkrete, regelgeleitete Vorgehen der im Kapitel 3.3 vorgestellten *zusammenfassenden Inhaltsanalyse* nach Mayring (2015) für die drei Interviews und deren Umsetzung in MAXQDA24 (siehe Abbildung 4.1). Die in der Abbildung gezeigt Software hilft bei der Strukturierung des Projekts, des Codesystem, der Memos (Notizen) und beim Export. Ziel ist eine nachvollziehbare Verdichtung des Materials zu kategorialen Kernaussagen bei gleichzeitigem Erhalt des inhaltlichen Kerns zur Ableitung der technischen Anforderungen des PoC.

Untersuchungsmaterial. Analysiert wurden die drei erhobenen Transkripte: Möller (2025a) in Deutsch, Springett (2025) in Englisch und Bressers (2025) in Englisch, jeweils vollständig und *wortgetreu* transkribiert mit *Zeitstempeln*. Die Transkripte aus eigener Erhebung liegen der Arbeit als **Anhang B, C und D** vor.

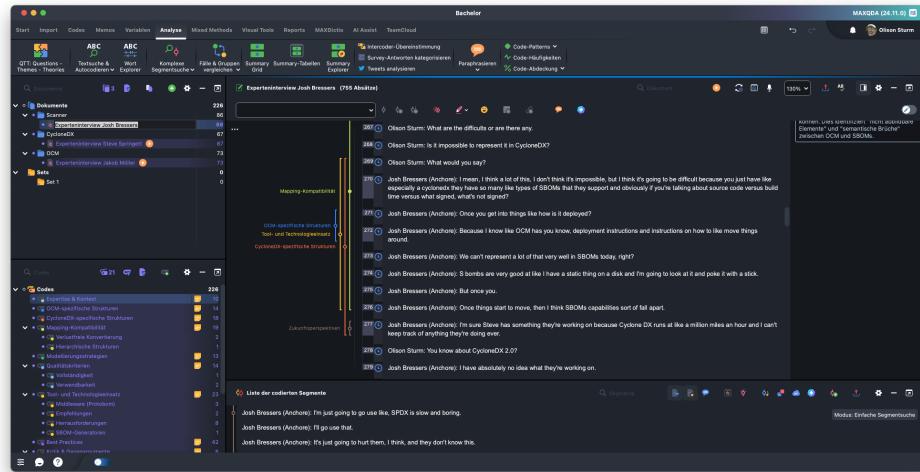


Abb. 4.1: MAXQDA24 Software für qualitative Datenanalyse

Theoriegeleitete Festlegung der Hauptkategorien. Vor der eigentlichen Analyse wurden, basierend auf der Forschungsfrage und der Sichtung der Materialinhalte, auf Basis einer *formlosen Dokumentenanalyse*, wie in Kapitel 3 beschrieben, zehn deduktive Hauptkategorien festgelegt. Diese Kategorien dienten als thematischer Rahmen für die Kodierung und die spätere induktive Subkategorienbildung.

Die Hauptkategorien lauten:

1. Expertise und Kontext
2. OCM-spezifische Strukturen
3. CycloneDX-spezifische Strukturen
4. Mapping-Kompatibilität
5. Modellierungsstrategien
6. Qualitätskriterien
7. Tool- und Technologieeinsatz
8. Kritik und Gegenargument an Forschung
9. Zukunftsperspektiven
10. Konkrete Anforderungen

Erstellung des Kodierleitfadens. Für jede deduktive Hauptkategorie wurde ein detaillierter Kodierleitfaden erstellt. Er fungierte als zentrales Hilfsmittel für die Kategorisierung. Alle Kodierleitfäden (Memos) sind im **Anhang E** dokumentiert. Beispielhaft wird der Leitfaden für die Kategorie „Best Practices“ in **Abbildung 4.2** dargestellt.

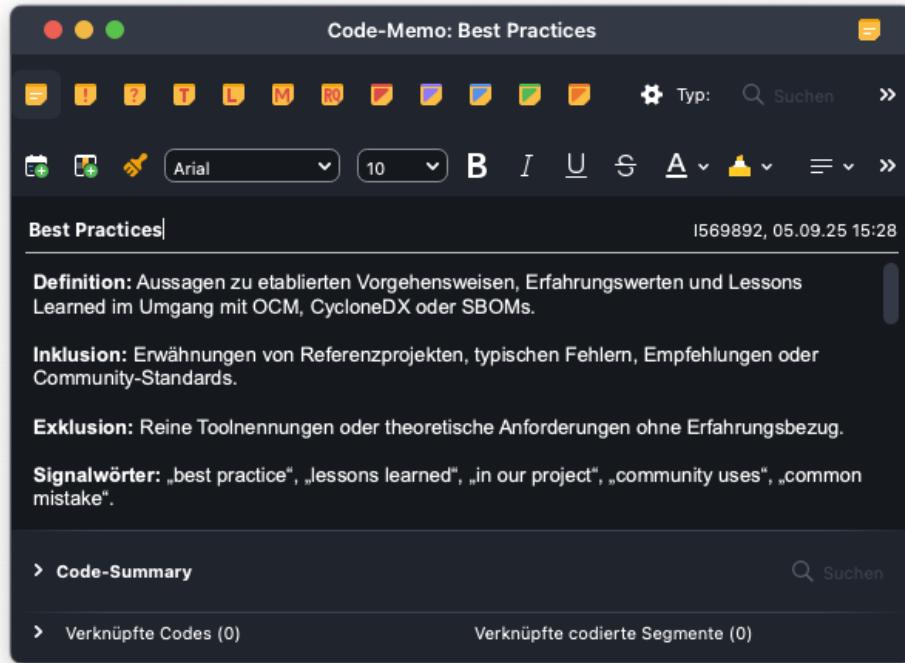


Abb. 4.2: Kodierleitfaden für „Best Practices“

Kompakte Umsetzung der zusammenfassenden Inhaltsanalyse (Z1–Z4). Die Auswertung erfolgte nach dem Ablaufmodell der zusammenfassenden Inhaltsanalyse (entsprechend Abbildung 3.2). Anstelle einer streng sequenziellen Abarbeitung der sieben Einzelschritte wurden die Z1–Z4-Regeln in einem komprimierten Zyklus auf die deduktiven Hauptkategorien angewandt und in MAXQDA24 praktisch umgesetzt:

1. *Paraphrasierung* in knapper, fachsprachlicher Form (Z1),
2. *Generalisierung* der Paraphrasen auf ein einheitliches Abstraktionsniveau (Z2) hierbei wurden auch induktiv Subkategorien angelegt,
3. erste *Reduktion* durch Selektion bedeutungsgleicher Formulierungen (Z3) sowie
4. zweite *Reduktion* durch Bündelung prägnanter Kernaussagen je Kategorie (Z4).

Dieses Vorgehen entspricht Mayrings Prozesslogik (siehe Abbildung 3.3), wonach die induktive Kategorienbildung im Analyseverlauf schrittweise am Material verfeinert und nach einem Teil (ca. 10–50%) *revidiert* wird (vgl. Mayring, 2015, S. 70, 85 f.). Die Revision (Zusammenlegung, Umbenennung und gegebenenfalls die Neuanlage) erfolgte nach etwa der Hälfte des Materials.

Ergebnisdarstellung. Die dabei entstandenen Subkategorien bilden somit eine verfeinerte Struktur des Kategoriensystems, um weitere wichtige Details zu dokumentieren. Die Auswertung wurde durch einen abschließenden Materialdurchlauf konsolidiert, bei dem sämtliche Interviews nach dem finalen Kategoriensystem kodiert wurden.

Das Ergebnis dieser Verdichtung ist in Form einer *Summary-Tabelle* mittels MAXQDA24 dokumentiert worden. Diese Tabelle enthält für jede Haupt- und Subkategorie eine kurze, abstrahierte Zusammenfassung der Kernaussagen und ist vollständig im **Anhang F** einsehbar. Sie bildet, als zentrales Endprodukt der QIA, die Grundlage für die Überführung in technische Anforderungen.

4.5 Triangulation und Erstellung des Anforderungskatalogs

Ausgehend von der in Kapitel 3.3.2 beschriebenen Vorgehensweise, wurden die Ergebnisse der *zusammenfassenden Inhaltsanalyse* der Experteninterviews mit der *Wissensbasis* aus der *formlosen Dokumentenanalyse* der Spezifikationen (OCM, CycloneDX) und den genannten Regularien zu SBOMs *trianguliert*. Ziel ist eine schlanke Validierung und Konsolidierung der interviewbasierten Befunde zu einem belastbaren Anforderungskatalog für den PoC.

Triangulation. Die *Summary-Tabelle* (siehe Anhang F) diente als Ausgangspunkt: Jede Anforderung wurde aus den kodierten Kernaussagen abgeleitet, *explizit* mit Expertenbezug begründet, durch die *Wissensbasis* verifiziert und mit fehlenden Informationen ergänzt, sodass keine Widersprüche entstehen konnten. Formulierung und Priorisierung folgten der RE-Taxonomie mit „muss“, „sollte“, „wird“, wie in Kapitel 3.3.2 dargelegt. Weitergehende Detailregeln bleiben diesem Methodik-Kapitel vorbehalten.

Funktionale Anforderungen (FANF).

FANF #1 Transformation OCM → CycloneDX (MUSS)

Der PoC **muss** OCM *Component Descriptor Trees* verlustarm in ein CycloneDX-SBOM überführen (Komponenten, Relationen, relevante Metadaten).

Begründung: Springett (2025) betont, dass CycloneDX in der Praxis breit unterstützt wird und sich als gemeinsame „Transparenzsprache“ etabliert. Springett (2025) und Bressers (2025) bestätigen, dass OCM-Inhalte so ohne Sonderwege in bestehenden Sicherheits- und Compliance-Toolchains nutzbar werden.

FANF #2 Ressourcen → Komponenten (MUSS)

Der PoC **muss** OCM *Resources*, prototypisch zunächst nur `ociImage`, als CycloneDX-Komponenten abbilden; Herkunft/Access fließt in Identität/Metadaten ein.

Begründung: Möller (2025a) unterscheidet klar zwischen Lieferartefakten (Ressourcen) und Quellen. Ressourcen lassen sich direkt als SBOM-Komponenten abbilden.

FANF #3 Hierarchie & Verknüpfungen (MUSS)

Der PoC **muss** die OCM Komponentenhierarchie (`componentReferences`) mittels verschachtelter Komponenten und externer Referenzen (BOM-Links) in CycloneDX ausdrücken.

Begründung: Möller (2025a) und Springett (2025) bestätigen, dass sich OCM-Komponentenreferenzen in CycloneDX mit verschachtelten Komponenten und BOM-Links abbilden lassen. So bleibt die Struktur des OCM-Component-Descriptor-Trees einschließlich externer Verweise erhalten.

FANF #4 Paketidentität via PURL (MUSS)

Der PoC **muss** Paket- und Komponentenidentitäten primär über PURL im CycloneDX-SBOM modellieren und bei Nichtverfügbarkeit per CPE oder SWID.

Begründung: Bressers (2025) hebt hervor, dass Scanner-Workflows auf korrekte Paketidentifikation angewiesen sind. PURLs liefern die stabilste Zuordnung für Vulnerability-Matching, CPE/SWID dienen als Fallback.

FANF #5 Aggregation/Zusammenführung (MUSS)

Der PoC **muss** aus SBOMs pro Ressource eine konsolidierte Root BOM erzeugen und dabei Identitäten/Referenzen konsistent halten.

Begründung: Bressers (2025) nennt Vulnerability-Scans und den Nachweis bestimmter Pakete als Hauptszenarien, die eine konsolidierte Root BOM erfordern. Springett (2025) weist auf unvollständige Aggregation externer Verweise in Tools hin, weshalb konsistente Identitäten und Referenzen essenziell sind.

FANF #6 Direkte Abbildung, keine Zwischenformate (SOLLTE)

Der PoC **sollte** ein direktes Mapping OCM→CycloneDX verwenden und auf generische Zwischenabstraktionen (z. B. Protobom) verzichten.

Begründung: Springett (2025) und Bressers (2025) warnen vor generischen Zwischenformaten, da sie den kleinsten gemeinsamen Nenner erzwingen, Spezifikationsfortschritt ausbremsen und Informationsverluste verursachen.

FANF #7 Identität & Herkunftsbelege (SOLLTE)

Der PoC **sollte** für jede SBOM-Komponente Identitäts-Evidenz hinterlegen, wie Toolname und Toolversion, Fundort (Pfad/Registry-Ref), eindeutige Kennung (bevorzugt PURL, ggf. CPE/SWID) sowie Integritätsmerkmale (SHA-Hash/Signatur) und zusätzlich die Minimum-Elements-Metadaten zum Erzeugungskontext dokumentieren (Build-/Delivery-Time, Ersteller/Organisation, Erstellzeitpunkt der SBOM).

Begründung: Springett (2025) empfiehlt, die Komponentenidentität mit Evidenz/Attribution zu untermauern. NTIA/CISA benennen „Author of SBOM

Data“ und „Timestamp“ zusätzlich noch als Kernelemente der Minimum Elements.

FANF #8 Übernahme vorhandener SBOMs (SOLLTE)

Der PoC **sollte** bereits existierende SBOMs in einer OCM-Komponente einbetten oder extern referenzieren (BOM-Link), ohne die Bedeutung von Signaturen zu verändern.

Begründung: Springett (2025) rät zur Einbettung/Referenzierung, da reines Re-Serialisieren bestehende Signaturen entwerten kann.

FANF #9 Pedigree/Abstammung (WIRD)

Der PoC **wird** in einer Folgestufe die CycloneDX-Pedigree-Strukturen nutzen, um Abstammung und Veränderungen von Komponenten zu dokumentieren (z. B. Originalquelle, Fork/Variante, angewandte Patches).

Begründung: Springett (2025) beschreibt Pedigree als nützliche Herkunfts- und Änderungshistorie; da viele Tools Pedigree noch kaum automatisiert auswerten, stützt sich die Identität vorerst auf Evidenz (vgl. FANF #7).

FANF #10 Bereitstellung (WIRD)

Der PoC **wird** in das OCM-CLI-Tooling integriert, um SBOMs direkt in Build-/Delivery-Pipelines und für Konsumenten als kommandozeilenbasierten Schritt ausführbar zu machen.

Begründung: Möller (2025a) empfiehlt die Bereitstellung als CLI zusätzlich zur Bibliothek, damit Integration in CI/CD und praktische Rückverfolgbarkeit gewährleistet sind.

Nicht-funktionale Anforderungen (NANF).

NANF #1 Schema-/Format-Konformität (MUSS)

Der PoC **muss** valide CycloneDX-SBOMs in der aktuellen Spezifikationsversion 1.6 erzeugen.

Begründung: Springett (2025) fordert standardkonforme Artefakte, da nur so Interoperabilität mit dem Ökosystem zuverlässig gegeben ist.

NANF #2 Scanner-Kompatibilität (MUSS)

Der PoC **muss** SBOMs liefern, die ohne manuelle Nacharbeit von verbreiteten OSS-Scannern (z. B. Grype, Trivy) verarbeitet werden können.

Begründung: Bressers (2025) betont die Praxistauglichkeit der OCM →CycloneDX Konvertierung für Scanner-Workflows. Da OCM nicht nativ unterstützt wird, müssen Ausgaben als CycloneDX mit PURL-Priorität vorliegen und in Smoke-Tests mit Grype oder Trivy bestehen.

NANF #3 Rückverfolgbarkeit & Round-Trip (MUSS)

Der PoC **muss** jede SBOM-Komponente eindeutig auf ihre OCM-Quelle

rückführbar halten (globale, stabile Identitäten und DNS-konforme OCM-Namenskonventionen).

Begründung: Möller (2025a) verlangt globale, stabile Identitäten, damit OCM und CycloneDX strukturell äquivalent bleiben und eine verlustarme Rückkonvertierung möglich ist.

NANF #4 Semantische Vollständigkeit (MUSS)

Der PoC **muss** relevante OCM-Informationen entweder nativ oder über CycloneDX-*properties* so abbilden, dass Tools sie auslesen können.

Begründung: Springett (2025) und Möller (2025a) plädieren für verlustarme Abbildung; nicht native Inhalte sollen als maschinenlesbare *properties* bereitstehen, damit Scanner sie zuverlässig verarbeiten.

NANF #5 Lizenz & Ökosystem-Kompatibilität (SOLLTE)

Der PoC **sollte** (Hilfs-)Frameworks unter Apache-kompatibler Lizenz nutzen.

Begründung: Möller (2025a) empfiehlt Apache-kompatible Lizenzen, um Nutzung und Weiterverbreitung im OCM-/OSS-Kontext langfristig sicherzustellen.

NANF #6 Wartbarkeit & Software-Bibliotheken (SOLLTE)

Der PoC **sollte** populäre, aktiv gepflegte Go Bibliotheken verwenden und veraltete Middleware meiden.

Begründung: Möller (2025a) fordert breit getragene, gepflegte Abhängigkeiten für langfristige Wartbarkeit. Bressers (2025) und Springett (2025) raten von Zwischenlagen wie *Protobom* ab, da sie Spezifikationsfortschritt bremsen und Pflegeaufwand erhöhen. Daher wird im PoC direkt die offiziellen CycloneDX Go Bibliothek (*cyclonedx-go*) für das Mapping verwendet.

Die Anforderungen wurden in einem ersten Schritt nach muss/sollte/wird Kriterien strukturiert, um die grundlegende Umsetzbarkeit des Proof of Concept sicherzustellen. Eine feinere Priorisierung innerhalb dieser Kategorien wurde bewusst nicht vorgenommen. Theoretisch wäre eine solche Gewichtung etwa mithilfe des etablierten „Kano-Modells“ nach Kano u. a. (1984) möglich gewesen, das zwischen Basis-, Leistungs- und Begeisterungsanforderungen unterscheidet. Für die vorliegende Arbeit war dies jedoch nicht notwendig, da der Anwendungsfokus auf einer prototypischen Validierung lag. In diesem Kontext genügt die Unterscheidung nach muss/sollte/wird Kriterien, um ein konsistentes und umsetzbares Anforderungsprofil zu gewährleisten. Eine detaillierte Priorisierung wäre vor allem in einem weiterführenden Projekt mit produktiver Zielsetzung sinnvoll.

5 Konzeption und prototypische Umsetzung

Dieses Kapitel führt die konzeptionellen Überlegungen und die prototypische Umsetzung des PoC zusammen. Es stützt sich auf die zuvor abgeleiteten technischen Anforderungen. Während der Implementierung wurden zahlreiche Erkenntnisse gewonnen, die den Entwurf geprägt haben. Diese sind im Folgenden dokumentiert.

5.1 Rahmen der Konzeption

Eine OCM-Komponente kann mehrere Artefakte im Ressourceteil enthalten. Im Kontext dieses Prototyps werden der Einfachheit halber ausschließlich Artefakte vom Typ `ociImages` verwendet. Diese müssen jeweils als SBOM erfasst werden. Im Folgenden wird diese Schicht als *Resource-Layer-SBOM* (R-SBOM) bezeichnet. Die einzelnen R-SBOMs sollen anschließend zu einer konsolidierten SBOM zusammengeführt werden, um einen OCM *Component Descriptor* als eine SBOM abzubilden. Diese konsolidierte SBOM wird im Folgenden als *Component-Layer SBOM* (C-SBOM) bezeichnet. Sie bildet noch nicht das gesamte durch OCM modellierte Software Produkt als eine SBOM ab, ermöglicht jedoch bereits die Durchführung eines einheitlichen Vulnerability-Scans auf Komponentenebene und stellt somit den ersten Schritt zu einer *Component-Tree-SBOM* (Root-SBOM) dar. Diese soll die komplette Hierarchie eines in OCM modellierten Produkts abbilden, inklusive aller `componentReferences`[¹]. Also die Baum-Hierarchie, ausgehend von der obersten *Parent-Component*. Dies kann ersten Überlegungen zufolge durch eine Aggregation der SBOMs nach dem Bottom-Up-Prinzip erreicht werden und orientiert sich an der Empfehlung von Springett, 2025, verschachtelte Komponenten (*nested components*) in CycloneDX zur hierarchischen Modellierung des OCM zu verwenden (FANF #3).

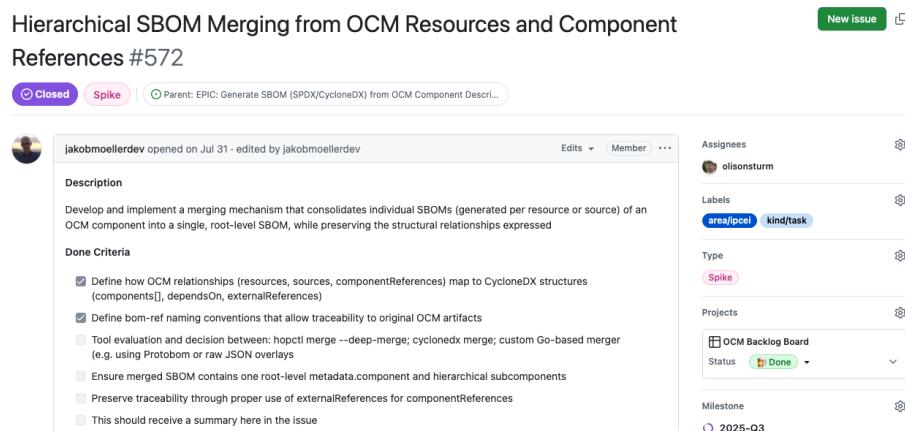


Abb. 5.1: GitHub-Issue: Spike (vgl. Möller, 2025b)

Um Entwicklungszeit zu sparen, Wartungsaufwand zu vermeiden und zugleich eine robuste Interoperabilität gegenüber sich fortentwickelnden SBOM Spezifikation sicherzu-

stellen, wurden zunächst etablierte Open-Source-Werkzeuge zur SBOM-Erzeugung und -Aggregation identifiziert und evaluiert. Die Toolauswahl erfolgte im Rahmen eines Spikes (siehe Abbildung 5.1): In einer zeitlich begrenzten Experimentierphase wurden geeignete SBOM-Werkzeuge pragmatisch erprobt, ohne formale Bewertungskriterien, mit dem Ziel, schnell Unsicherheiten zu reduzieren und den für das Vorhaben tauglichsten Ansatz zu identifizieren.

Auswahl der *Third-Party-Tools*. Für die Erzeugung der R-SBOMs wurden die im Interview genannten, etablierten Open-Source-SBOM-Generatoren *Syft* und *Trivy* getestet. Diese Toolwahl ist nicht von entscheidender Bedeutung, da OCM-Komponenten zwar SBOMs pro Resource (R-SBOMs) benötigen, jedoch die SBOMs auch bereits als *Resource* mit dem SBOM-Mediatype im OCM vorliegen können. Falls nicht, müssen sie generiert werden, um die Aggregation durchzuführen. Deshalb fiel vorab die Wahl auf einen generischen Ansatz dieses CLI-Prototyps. Dieser ermöglicht, dass eine R-SBOM vorliegt, unabhängig davon, wie diese erzeugt wurde. Im weiteren Verlauf wurden verschiedene CLI-Tools zum Generieren und *Mergen* ausprobiert; da diese ihre Eigenheiten mitbrachten, ermöglichte der CLI-Prototyp per Flag (etwa „`--merge-tool <hoppr;cyclonedx-cli>`“) die Auswahl der Tools beim Ausführen des Konvertierungsbefehls eines *Component Descriptors* in die CycloneDX-SBOM. Da die Herkunft der R-SBOM nicht von Bedeutung ist, sondern nur das Abbilden der Informationsstrukturen des OCMs in der SBOM, wurde in der weiteren prototypischen Entwicklung jedoch auf *Syft* gesetzt. *Syft* bot zusätzlich die Möglichkeit, statt des bloßen Aufrufs der *Syft*-CLI über das Go-Paket `exec` (etwa „`cmd := exec.Command("syft", "alpine:3.18", "-o", "cyclonedx-json")`“) die SBOM-Erzeugung über die *Syft*-Go-Bibliothek zu lösen. Da der Prototyp in Go entwickelt wurde, kann so das manuelle Installieren der *Syft*-CLI entfallen. *Syft* dient damit, falls keine vorhandene SBOM existiert, der Generierung von SBOMs für Container-Images und Dateisysteme. Es unterstützt gängige Formate wie CycloneDX und SPDX und liefert detaillierte Informationen zu Paketen und Abhängigkeiten, was für eine spätere Schwachstellenanalyse wesentlich ist.

Für das anschließende Merging der R-SBOMs fiel die Wahl auf das offizielle *CycloneDX CLI*. Die CLI unterstützt sowohl einen flachen Merge, sodass alle Abhängigkeiten (Dependencies) in CycloneDX auf einer Ebene landen, als auch das hierarchische Merging. Letzteres erfordert, dass beim Ausführen des *Merge Command* zusätzlich zu den Eingangs-SBOM Angaben der *Name* und gegebenenfalls *Group* und *Version* einer so genannten *Metadata-Komponente* spezifiziert werden, die die C-SBOM beschreibt und übergeordnet über alle CycloneDX `components[]` (alle Eingangs-SBOMs), den Metadaten des OCM *Component Descriptor* entspricht. Nur dann kann das Werkzeug einen hierarchischen Merge durchführen und die komplette OCM Komponente als eine SBOM

abbilden. Neben dem CycloneDX-CLI wurde auch die Alternative *Hoppr* evaluiert. Hoppr implementiert einen eigenen Merge-Algorithmus und kann verschachtelte SBOMs mittels `--flatten` auflösen und externe Referenzen erweitern. Die Hoppr-Dokumentation beschreibt, dass das Kommando `hopctl merge` einen „deep-merge“ durchführt, der verschachtelte Komponenten auflöst und externe Referenzen in der resultierenden SBOM ersetzt. Für den Prototyp erwies sich jedoch die C# Implementierung der CycloneDX CLI als geeigneter, da es die Hierarchie der OCM Komponenten direkt abbildet und keine zusätzlichen Modellklassen benötigt. Diese Logik wurde im späteren Verlauf noch exakt in einem Go *Interface* nachgebaut (siehe Anhang I), um auch hier das manuelle Installieren der CycloneDX CLI zu erübrigen.

Konzept der hierarchischen Modellierung. Die CycloneDX-Spezifikation erlaubt zwei wesentliche Methoden, um Relationen zwischen mehreren SBOMs auszudrücken: das Einbetten von *Assemblies* in der Liste der `components[]` und die Verwendung von *BOM-Links*. Assemblies beschreiben verschachtelte Komponenten: ein Eltern-BOM enthält weitere BOMs als seine Bestandteile, ähnlich wie ein Armaturenbrett, das einen Instrumententafel-BOM sowie einen Tachometer-BOM umfasst. Diese Verschachtelung bildet die Struktur eines Systems ab, ohne Abhängigkeitsbeziehungen auszudrücken. Bei Bedarf können zudem Dependency-Graphen angegeben werden, die den Verwendungszusammenhang der Komponenten definieren (`dependsOn`). Die CycloneDX-Guidelines erläutern, wie Abhängigkeitsgraphen anhand der eindeutigen Kennung *bom-ref* modelliert werden. Eine Komponente kann über das Attribut `dependsOn` auf andere Komponenten verweisen.

Alternativ ermöglicht die CycloneDX-Spezifikation mit dem *BOM-Link* eine Referenzierung externer SBOMs. Ein BOM-Link ist eine URN, die auf ein anderes SBOM oder ein Objekt innerhalb eines SBOM verweist. Die BOM-Link-Spezifikation beschreibt, dass BOM-Links URN-basierte Verweise auf andere SBOMs oder Objekte innerhalb eines SBOM sind und damit modulare und domänenübergreifende Verlinkungen ermöglichen. Damit lässt sich die Aggregation modular gestalten: einzelne Komponenten behalten ihre eigenen SBOMs, während das übergeordnete SBOM lediglich Verweise enthält. Die Trennung erhöht die Modularität und vereinfacht die Pflege sensibler Daten. Im Rahmen des Prototyps wurde auf die Verwendung von BOM-Links verzichtet, um zunächst eine konsolidierte SBOM zu erzeugen. Für künftige Erweiterungen, etwa um komponentenweise Signaturen beizubehalten oder umfangreiche Komponentensammlungen zu referenzieren, ist der Einsatz von BOM-Links jedoch vorgesehen.

5.2 Architektur und Prototypische Implementierung

Die Implementierung des Prototyps erfolgte überwiegend in Go. Da für CycloneDX bisher keine Go-Bibliothek zum Mergen mehrerer SBOMs existiert, wurde die C# Logik des *CycloneDX CLI Merge* als Go *Interface* implementiert. Diese ist im **Anhang 9.1** einzusehen und verwendet die neueste Version der Bibliothek `cyclonedx-go`, um den Merge nativ in Go zu unterstützen und um Anpassungen zu ermöglichen.

5.2.1 Architekturüberblick

Die Anwendung ist wie folgt strukturiert: `cmd/root.go` ein Cobra-basiertes CLI bildet die oberste Ebene. Darunter liegt ein *Converter*-Modul `cmd/convert.go`, das aus folgenden Komponenten besteht:

CTF-Reader Das Modul `read_from_ctf.go` öffnet ein OCM-Archiv und stellt über das OCM-Go Binding einen `ComponentVersionRepository` bereit.

Component-Processor Der `component_processor.go` verarbeitet eine einzelne Komponente. Für jede im Descriptor referenzierte Ressource (`ociImage`) wird über Syft eine SBOM generiert. Die Methode `ProcessComponent` legt dafür ein temporäres Verzeichnis an, ruft die SBOM-Erzeugung auf und übergibt die Pfade an den *Merger*. Nach dem Merging werden die Metadaten der SBOM an den Namen und die Version der Komponente angepasst.

SBOM-Generator In `generateComponentResourceSboms` wird für jede Ressource mit Typ `ociImage` eine SBOM über den Anchore-Scanner erzeugt. Der Scanner wird über das Syft-Go-Paket konfiguriert. Das Ausgabeformat entspricht dem vom Nutzer gewählten Ziel.

SBOM-Merger Der `component_sbom_merger.go` implementiert die Zusammenführung mehrerer SBOMs. Im „native“-Modus werden die einzelnen SBOM Dateien mit dem Decoder von `cyclonedx-go` eingelesen, auf gültige Metadaten geprüft und anschließend mit dem eigenen Merge-Algorithmus kombiniert. Das Ergebnis wird als CycloneDX 1.6 im JSON-Format serialisiert. Für den Fall, dass alternative Tools erforderlich sind, bietet die API optional Aufrufe an *Hoppr* oder die externe CycloneDX-CLI. Hoppr ist momentan noch nicht implementiert.

Converter Das Modul `convert_impl.go` orchestriert den Prozess: Es traversiert die komplette Komponentenhierarchie, ruft je Komponente den Processor auf und führt die SBOMs bottom-up zusammen. Das Endergebnis ist eine konsolidierte SBOM für die Wurzelkomponente. Ein optionaler Konverter ruft die CycloneDX-CLI nur noch für Formatkonvertierungen auf (etwa zu SPDX falls gewünscht).

5.2.2 Traversierung und *Bottom-Up-Merging*

Der entscheidende Teil der Implementierung und des PoC ist die rekursive Traversierung des Komponentenbaums. Dies bedeutet, dass ausgehend von der Wurzelkomponente alle referenzierten Komponenten systematisch besucht, ihre Beziehungen erfasst und Mehrfachbesuche über eine Besuchsmarkierung verhindert werden, sodass jede Komponente genau einmal verarbeitet und zyklische Referenzen dedupliziert werden. Dies wird erreicht durch die *Bottom-Up*-Aggregation der SBOMs, bei der zunächst für jede Blatt-Komponente eine R-SBOM erzeugt und anschließend die Ergebnisse schrittweise zu den Eltern propagiert werden, bis die C-SBOM der Wurzel vorliegt. Die Methode `processAllComponents` (siehe Listing 5.1) konstruiert dazu zuerst den Graphen aller abhängigen Komponenten per *Breadth-First-Search* (BFS). BFS bedeutet, dass die Knoten *ebenenweise* mithilfe einer Queue besucht werden—zuerst alle direkten Nachfolger der Wurzel, danach deren Nachfolger und so weiter. Wodurch der vollständige Abhängigkeitsgraph ohne Tiefenrekursion aufgebaut und für die anschließende bottom-up-Aggregation vorbereitet wird.

```

1  for len(queue) > 0 {
2      curr := queue[0]
3      queue = queue[1:]
4      currID := fmt.Sprintf("%s:%s", curr.Name, curr.Version)
5      if visited[currID] {
6          continue
7      }
8      visited[currID] = true
9      // Laufzeitdescriptor laden
10     runtimeDesc, err := repo.GetComponentVersion(ctx, curr.Name, curr.
11             Version)
12     // ... Fehlerbehandlung ...
13     // -ResourceSBOMs generieren und Pfad speichern
14     mergedSBOMPath, _ := processor.ProcessComponent(runtimeDesc,
15             outputFormat, mergeTool)
16     resourceSBOMPath[currID] = mergedSBOMPath
17     // Referenzen (Kinder) einreihen
18     for _, ref := range runtimeDesc.Component.References {
19         childID := fmt.Sprintf("%s:%s", ref.Component, ref.Version)
20         childrenOf[currID] = append(childrenOf[currID], childID)
21         if !visited[childID] {
22             queue = append(queue, compKey{Name: ref.Component, Version: ref.
23                 Version})
24         }
25     }
26 }
```

Listing 5.1: BFS zum Erzeugen eines Komponentenbaums und der Resource-SBOMs

Nach Aufbau des Graphen wird für jeden Knoten gezählt, wie viele Kindknoten noch zu verarbeiten sind. Anschließend wird in einer Schleife jeweils ein Blattknoten genommen und sein R-SBOM mit den SBOMs aller Kindknoten verschmolzen. Dies geschieht solange, bis alle Knoten verarbeitet sind:

```

1  for _, nid := range batch {
2      // Liste der -SBOMDateien: -ResourceSBOM des Knotens und alle
3      // bereits zusammengeführten -KindSBOMs
4      files := []string{}
5      if p := resourceSBOMPath[nid]; p != "" { files = append(files, p) }
6      for _, cid := range childrenOf[nid] {
7          if cp := resultPath[cid]; cp != "" { files = append(files, cp) }
8      }
9      // Liegt nur eine Datei vor, ist sie das Ergebnis; sonst wird
10     // ComponentSbomMerge aufgerufen, wobei Name und Version aus
11     // dem Knotenschlüssel abgeleitet werden
12     finalPath := files[0]
13     if len(files) > 1 {
14         compName, compVersion := parse(nid)
15         outPath, err := merger.ComponentSbomMerge(tempDir, files, mergeTool,
16             compName, compVersion)
17         if err == nil { finalPath = outPath }
18     }
19     resultPath[nid] = finalPath
20     // Eltern über Abschluss informieren ...
}

```

Listing 5.2: Bottom-up-Merge der SBOMs über die Komponentenstruktur

Die `ComponentSbomMerge`-Methode (siehe Listing 5.3) entscheidet anhand des `mergeTool`-Parameters, welches Verfahren angewendet wird. Im `native`-Modus werden alle Eingangs-SBOMs mit `cyclonedx-go` eingelesen. Zur Wahl stünde auch der `hoppr`-Modus, dieser unterstützt jedoch keine Hierarchien und erstellt eine flache SBOM mit allen Abhängigkeiten und ermöglicht somit keine Rückreferenzierbarkeit. Nach der Validierung der Metadaten wird der CycloneDX CLI kopierte Merge-Algorithmus `CycloneDXMerge` aufgerufen:

```

1  var boms []cyclonedx.BOM
2  for _, path := range resourceSbomPaths {
3      file, _ := os.Open(path)
4      bom := cyclonedx.BOM{

```

```

5     decoder := cyclonedx.NewBOMDecoder(file, cyclonedx.BOMFileFormatJSON)
6     decoder.Decode(&bom)
7     // Sicherstellen, dass eine Metadatenkomponente vorhanden ist
8     if bom.Metadata == nil || bom.Metadata.Component == nil {
9         return "", fmt.Errorf("invalid BOM: missing metadata")
10    }
11    boms = append(boms, bom)
12 }
13 opts := CycloneDxMergeOptions{
14     BOMs:      boms,
15     Name:      componentName,
16     Version:   componentVersion,
17     MergeMode: MergeModeHierarchical,
18 }
19 mergedBom, err := CycloneDXMerge(opts)
20 // Ergebnis in Datei schreiben und als CycloneDX 1.6 serialisieren

```

Listing 5.3: Lesen & Zusammenführen mehrerer R-SBOMs zur C-SBOM

Der Aufruf `CycloneDXMerge` ist der Kern der Aggregation. Für einen hierarchischen Merge wird zunächst ein *BOM-Subject* aus den übergebenen Parametern (Name, Version, optional Group) gebildet und dann eine tiefe Verschmelzung aller Eingangs-BOMs durchgeführt. Dabei werden BOM-Referenzen mit *Namespaces* versehen, Tool-Informationen zusammengeführt und die *Dependency*-Relationen entsprechend aktualisiert. Die Implementierung orientiert sich an der CycloneDX CLI, ist aber eigenständig realisiert und unterstützt zwei Hierarchie-Szenarien für die hierarchische Zusammenführung.

5.2.3 Zwei Hierarchie-Szenarien

Die native Merging-Logik unterscheidet indirekt zwei Hierarchie-Szenarien:

1. **R-SBOM-Aggregation:** Alle SBOMs, die für die Ressourcen einer Komponente generiert wurden oder vorliegen, werden zu einer *Component-Layer*-SBOM zusammengeführt. Hierbei existiert zunächst kein übergeordneter Parent, denn der Merge selber erzeugt einen neuen CycloneDX SBOM mit Namen und Versionsangaben der OCM Komponente und hängt alle R-SBOMs als `components []`/Kinder darunter an.
2. **C-SBOM-Aggregation:** Sobald Kindkomponenten vorhanden sind, wird beim bottom-up-Merge eine existierende übergeordnete C-SBOM gemeinsam mit den bereits aggregierten C-SBOMs der Kindkomponenten zusammengeführt. Das Ergebnis bildet sowohl die Metadaten des Parents als auch die Hierarchie der Kindkomponenten ab. Die Implementierung ruft hierfür erneut `ComponentSbomMerge` mit allen beteiligten Pfaden auf. Die Merge-Logik sorgt dafür, dass das CycloneDX BOM

Objekt unverändert bleibt und die BOM-Referenzen der Kinder korrekt mit einem *Namespace* versehen werden.

Obwohl beide Fälle den gleichen Algorithmus verwenden, entsteht durch die Bottom-Up-Strukturierung eine natürliche Unterscheidung zwischen dem einmaligen Erzeugen eines Parent-SBOMs für eine Komponente und dem nachträglichen Einfügen weiterer Kind-komponenten in eine bereits bestehende Hierarchie.

5.3 Entwurfsentscheidungen

Die wichtigsten technischen Entscheidungen lassen sich wie folgt zusammenfassen:

Direkter Einsatz der CycloneDX-Go-Bibliothek. Um Informationsverlust durch Zwischenschichten zu vermeiden, wird das Datenmodell von OCM direkt auf die Struktu-ren der CycloneDX-Bibliothek abgebildet. Dadurch lassen sich Metadaten präzise setzen (PURL-First) und spätere Erweiterungen wie Pedigree-Angaben leichter ergänzen.

Natives Merge statt externer CLI. Die frühere Integration der CycloneDX CLI wurde durch eine eigene Go Implementierung ersetzt. Das reduziert externe Abhängigkeiten, vermeidet Installationsaufwand und erlaubt eine feinere Anpassung der Merge-Strategie. Die Implementierung orientiert sich an den Spezifikationen der CycloneDX CLI, erwei-tert sie jedoch um die Möglichkeit, mehrere hierarchische Merges innerhalb eines Baumes auszuführen.

Bottom-up-Aggregation. Anstatt Komponenten wiederholt einzeln zu mergen und anschließend erneut zu traversieren, wird der gesamte Komponentenbaum zuerst aufge-baut und dann von den Blättern zur Wurzel hin aggregiert. Dieses Verfahren garantiert, dass die Wurzel-SBOM die vollständige Hierarchie enthält und erleichtert die spätere Trennung in R-SBOM, C-SBOM und Root-SBOM.

Konfigurierbares Merge-Tool. Obwohl der native Merge der Standard ist, lässt sich über das Flag `--merge-tool` zukünftig auch Hoppr verwenden. Die CLI prüft zur Laufzeit, ob die entsprechenden Binaries vorhanden sind und warnt den Anwender andernfalls und leitet ihn meist durch einen Installationsprozess.

Validierung und Formatkonvertierung. Die Methode `convertFinalSBOM` in An-hang J kümmert sich um die Konvertierung des zusammengeführten SBOMs in das gewünschte Zielformat. Wird ein nicht-JSON-Format gewünscht, ruft sie das Cyclone-DX-CLI mit den Parametern `convert --input-file; --output-format {json|yaml} --output-version {1.6|2.3}` auf. Für SPDX-Exporte wird die Version 2.3 gewählt.

Anschließend wird das Ergebnis zurückgelesen. Eine explizite Schema-Validierung über JSON-Schema-Dateien ist noch offen und könnte als separate Validierungsstufe nachgerüstet werden.

Build und Portabilität. Der PoC ist in Go geschrieben, da das OCM CLI Toolset auch auf einer Cobra-basierten Go Implementierung basiert und so leicht produktiv aufgenommen werden kann. Die Liste der externen Abhängigkeiten ist bewusst klein gehalten, um Lizenzrisiken und potenzieller Schwachstellen zu reduzieren (siehe Tabelle 5.1).

Modul	Zweck
github.com/spf13/cobra	CLI-Framework
ocm.software/open-component-model/bindings/go	Lesen von Komponenten (CTF)
github.com/CycloneDX/cyclonedx-go	Datenmodelle für CycloneDX

Tab. 5.1: Verwendete Go-Module und ihr Zweck

Insgesamt ergibt sich eine modulare und erweiterbare Architektur, die den hierarchischen Aufbau des OCM *Component Trees* entspricht. Die im Prototyp implementierte Merge-Logik kann bei Bedarf um weitere Szenarien ergänzt werden, etwa die Einbindung von BOM-Links oder Integritätsnachweise.

5.4 Anwendungsbeispiel

Der PoC lässt sich über die bereitgestellte CLI demonstrieren. Um die Bedienung und das Ergebnis im Terminal, sowie die Struktur einer generierten SBOM zu veranschaulichen, wird in diesem Abschnitt ein Anwendungsbeispiel ergänzt. Das Beispiel soll zeigen, wie mit folgendem Befehl

```
1 go run main.go convert ./example-ocm/ctf//github.com/acme.org/app \
2   --format cyclonedx-json --output root-sbom.cdx.json --merge-tool
native
```

eine aggregierte Root-SBOM erzeugt werden kann. Nach der Ausführung des Befehls wurden die Tools hoppr und cyclonedx-cli als uninstalled identifiziert. Weshalb in **Abbildung 5.2** und **5.3** die direkte Installierung akzeptiert (`cyclonedx-cli`) oder beispielsweise abgelehnt (`hoppr`) werden kann.

```
I569892@MY3W05J4FT ocm-sbom % go run main.go convert ./example-ocm/ctf//github.com/olison/parent
--format cyclonedx-json --output root-sbom.cdx.json --merge-tool native
2025/09/23 17:32:14 CycloneDX CLI (cyclonedx) not found.
cyclonedx not found. Do you want to install it using 'brew install cyclonedx/cyclonedx/cyclonedx-
cli'? (y/N) y
2025/09/23 17:32:17 Attempting to install cyclonedx using brew install...
==> Auto-updating Homebrew...
```

Abb. 5.2: CycloneDX CLI (cyclonedx) not found

```
2025/09/23 17:25:36 Hoppr (hopctl) CLI not found.
hopctl not found. Do you want to install it using '/usr/bin/python3 -m pip install hoppr'? (y/N)
N
Error: failed to initialize SBOM converter: failed to install hopctl: user declined to install hopctl. Please install manually
Usage:
  ocm-sbom convert [CTF_PATH]//[COMPONENT_NAME] [flags]
```

Abb. 5.3: Hoppr (hopctl) CLI not found

In Abbildung 5.4 hingegen ist das erfolgreiche Generieren einer Root-SBOM im Terminal zu sehen. Hierbei wird jeder einzelne Zwischenschritt, welcher in Kapitel 5.2 beschrieben wurde, sorgfältig in das Terminal geloggt. Die Generierung basierte auf folgendem OCM *Component Tree* (siehe Listing 5.4):

```
1 # yaml-language-server: $schema=https://ocm.software/schemas/
2   configuration-schema.yaml
3
4 components:
5   # Root-Komponente, verweist nur auf die beiden Unterkomponenten
6   - name: github.com/acme.org/app
7     version: 1.0.0
8     provider:
9       name: acme.org
10      componentReferences:
11        - name: backend
12          componentName: github.com/acme.org/app/backend
13          version: 1.0.0
14        - name: frontend
15          componentName: github.com/acme.org/app/frontend
16          version: 1.0.0
17
18      # Child A: zwei OCI-Images
19      - name: github.com/acme.org/app/backend
20        version: 1.0.0
21        provider:
22          name: acme.org
23        resources:
24          - name: api
25            type: ociImage
26            version: 1.0.0
27            access:
28              type: ociArtifact
29              imageReference: docker.io/library/alpine:3.19.1
30          - name: worker
31            type: ociImage
32            version: 1.0.0
33            access:
34              type: ociArtifact
35              imageReference: docker.io/library/busybox:1.36.1
36
37      # Child B: ein OCI-Image
```

```

34   - name: github.com/acme.org/app/frontend
35     version: 1.0.0
36     provider:
37       name: acme.org
38     resources:
39       - name: web
40         type: ociImage
41         version: 1.0.0
42         access:
43           type: ociArtifact
44           imageReference: docker.io/library/nginx:1.25

```

Listing 5.4: Quellcode: Beispiel component-constructor.yaml

Die resultierende CycloneDX SBOM sieht in gekürzter Version in **Abbildung 5.5** wie folgt aus. Die vollständig generierte Root-SBOM kann im Anhang leider nicht nachvollzogen werden, da die Größe, wie in der Abbildung anhand der Zeilen zu sehen ist, zu einem *compile time out* führt.

```

ocm-sbom > {} root-sbom.cdx.json ...
1  {
2    "bomFormat": "CycloneDX",
3    "specVersion": "1.6",
4    "serialNumber": "urn:uuid:2f5b5b9d-b13c-4768-918f-e3f57f75eb28",
5    "version": 1,
6    "metadata": {
7      "tools": {
8        "components": [
9          {
10            "type": "application",
11            "author": "anchore",
12            "name": "syft",
13            "version": "v1.30.0"
14          }
15        ],
16        "component": {
17          "bom-ref": "github.com/acme.org/app@1.0.0",
18          "type": "application",
19          "name": "github.com/acme.org/app",
20          "version": "1.0.0"
21        }
22      },
23      "components": [...]
58499  ],
58500  "dependencies": [...]
59696  ]
59697  ]

```

Abb. 5.5: Auszug der Root-SBOM



```

I569892@MY3W05J4FT ocm-sbom % go run main.go convert ./example-ocm/ctf//github.c
om/acme.org/app --format cyclonedx-json --output root-sbom.cdx.json --merge-tool
native
2025/09/23 17:54:39 Processing OCM component: github.com/acme.org/app from CTF:
./example-ocm/ctf
2025/09/23 17:54:39 Target formats: [cyclonedx-json]
2025/09/23 17:54:39 Output path: root-sbom.cdx.json
2025/09/23 17:54:39 ComponentSbomMerge tool: native
2025/09/23 17:54:39 Generating SBOM for format: cyclonedx-json to root-sbom.cdx.
json
2025/09/23 17:54:39 Processing component (graph build): github.com/acme.org/app:
1.0.0
2025/09/23 17:54:39 INFO resolving descriptor reference=ctf.ocm.software/compo
nt-descriptors/github.com/acme.org/app:1.0.0
2025/09/23 17:54:39 INFO fetching descriptor descriptor.mediaType=application/vn
d.oci.image.manifest.v1+json descriptor.digest=sha256:4680896a3cccd04213a605c3ccf
0765b2272be08b4e16bbbf1e53b3d187a373 descriptor.size=560
2025/09/23 17:54:39 Processing component: github.com/acme.org/app:1.0.0
2025/09/23 17:54:39 No OCI Image resources found or no SBOMs could be generated
for component github.com/acme.org/app:1.0.0
2025/09/23 17:54:39 Processing component (graph build): github.com/acme.org/app/
backend:1.0.0
2025/09/23 17:54:39 INFO resolving descriptor reference=ctf.ocm.software/compo
nt-descriptors/github.com/acme.org/app/backend:1.0.0
2025/09/23 17:54:39 INFO fetching descriptor descriptor.mediaType=application/vn
d.oci.image.manifest.v1+json descriptor.digest=sha256:3cc6a68bd418d38597dee0300c
fb5622ecb10bfe2e2b284becd5d5d0118be421 descriptor.size=568
2025/09/23 17:54:39 Processing component: github.com/acme.org/app/backend:1.0.0
2025/09/23 17:54:39 Generating SBOM with Syft for OCI Image resource: docker.io/
library/alpine:3.19.1
2025/09/23 17:54:41 SBOM generated and saved for resource docker.io/library/alpi
ne:3.19.1 at /var/folders/27/5gt49c0j5sd7tsh_jkq3wd8h0000gn/T/ocm-sbom-conv-9723
39921/ocm-github.com-acme.org-app-backend-sboms-672429911/docker.io-library-alpi
ne-3.19.1-api-sbom.json
2025/09/23 17:54:41 Generating SBOM with Syft for OCI Image resource: docker.io/
library/busybox:1.36.1
2025/09/23 17:54:43 SBOM generated and saved for resource docker.io/library/busy
box:1.36.1 at /var/folders/27/5gt49c0j5sd7tsh_jkq3wd8h0000gn/T/ocm-sbom-conv-972
339921/ocm-github.com-acme.org-app-backend-sboms-672429911/docker.io-library-bus
ybox-1.36.1-worker-sbom.json
2025/09/23 17:54:43 Merging the following 2 SBOMs
2025/09/23 17:54:43 1: /var/folders/27/5gt49c0j5sd7tsh_jkq3wd8h0000gn/T/ocm-sb
om-conv-972339921/ocm-github.com-acme.org-app-backend-sboms-672429911/docker.io-
library-alpine-3.19.1-api-sbom.json
2025/09/23 17:54:43 2: /var/folders/27/5gt49c0j5sd7tsh_jkq3wd8h0000gn/T/ocm-sb
om-conv-972339921/ocm-github.com-acme.org-app-backend-sboms-672429911/docker.io-
library-busybox-1.36.1-worker-sbom.json
2025/09/23 17:54:43 Executing native Go merge...
2025/09/23 17:54:43 SBOMs successfully merged to: /var/folders/27/5gt49c0j5sd7ts
h_jkq3wd8h0000gn/T/ocm-sbom-conv-972339921/ocm-github.com-acme.org-app-backend-s
boms-672429911/merged-component-github.com-acme.org-app-backend.json
2025/09/23 17:54:43 SBOM for component github.com/acme.org/app/backend:1.0.0 at
/var/folders/27/5gt49c0j5sd7tsh_jkq3wd8h0000gn/T/ocm-sbom-conv-972339921/ocm-git
hub.com-acme.org-app-backend-sboms-672429911/merged-component-github.com-acme.or
g-app-backend.json
2025/09/23 17:54:43 Processing component (graph build): github.com/acme.org/app/
frontend:1.0.0
2025/09/23 17:54:43 INFO resolving descriptor reference=ctf.ocm.software/compo
nt-descriptors/github.com/acme.org/app/frontend:1.0.0
2025/09/23 17:54:43 INFO fetching descriptor descriptor.mediaType=application/vn
d.oci.image.manifest.v1+json descriptor.digest=sha256:697ae81d0eec18ab3f3cccd6018
48d6ff7c86c7699ab52a43fa5df336273f30 descriptor.size=569
2025/09/23 17:54:43 Processing component: github.com/acme.org/app/frontend:1.0.0
2025/09/23 17:54:43 Generating SBOM with Syft for OCI Image resource: docker.io/
library/nginx:1.25
2025/09/23 17:54:48 SBOM generated and saved for resource docker.io/library/ning
inx:1.25 at /var/folders/27/5gt49c0j5sd7tsh_jkq3wd8h0000gn/T/ocm-sbom-conv-972339
921/ocm-github.com-acme.org-app-frontend-sboms-4056089721/docker.io-library-ning
inx-1.25-web-sbom.json
2025/09/23 17:54:48 Merging the following 1 SBOMs
2025/09/23 17:54:48 1: /var/folders/27/5gt49c0j5sd7tsh_jkq3wd8h0000gn/T/ocm-sb
om-conv-972339921/ocm-github.com-acme.org-app-frontend-sboms-4056089721/docker.i
o-library-nginx-1.25-web-sbom.json
2025/09/23 17:54:48 Executing native Go merge...
2025/09/23 17:54:48 SBOMs successfully merged to: /var/folders/27/5gt49c0j5sd7ts
h_jkq3wd8h0000gn/T/ocm-sbom-conv-972339921/ocm-github.com-acme.org-app-frontend-
sboms-4056089721/merged-component-github.com-acme.org-app-frontend.json
2025/09/23 17:54:48 SBOM for component github.com/acme.org/app/frontend:1.0.0 at
/var/folders/27/5gt49c0j5sd7tsh_jkq3wd8h0000gn/T/ocm-sbom-conv-972339921/ocm-git
hub.com-acme.org-app-frontend-sboms-4056089721/merged-component-github.com-acme
.org-app-frontend.json
2025/09/23 17:54:48 Executing native Go merge...
2025/09/23 17:54:48 SBOMs successfully merged to: /var/folders/27/5gt49c0j5sd7ts
h_jkq3wd8h0000gn/T/ocm-sbom-conv-972339921/merged-component-github.com-acme.org-
app.json
2025/09/23 17:54:48 Final merged SBOM at: /var/folders/27/5gt49c0j5sd7tsh_jkq3wd
8h0000gn/T/ocm-sbom-conv-972339921/merged-component-github.com-acme.org-app.json
2025/09/23 17:54:48 No format conversion needed; reading merged SBOM directly
2025/09/23 17:54:48 Successfully generated cyclonedx-json SBOM to root-sbom.cdx.
json
2025/09/23 17:54:48 OCM to SBOM conversion process completed.
2025/09/23 17:54:48 Cleaning up temporary directory: /var/folders/27/5gt49c0j5sd
7tsh_jkq3wd8h0000gn/T/ocm-sbom-conv-972339921
I569892@MY3W05J4FT ocm-sbom %

```

Abb. 5.4: Terminal: Erfolgreiche CLI SBoD zu SBOM Konvertierung

6 Evaluation und Ergebnisinterpretation

In diesem Kapitel wird systematisch geprüft, in welchem Maße der umgesetzte Prototyp die in Kapitel 4 definierten funktionalen und nicht-funktionalen Anforderungen erfüllt. Die Bewertung basiert auf kontrollierten Experimenten mit Beispiel OCM Komponenten, der Analyse der erzeugten SBOMs sowie einer kritischen Gegenüberstellung der Implementierung mit den formulierten Zielen. Zur Visualisierung des Erfüllungsgrads werden Harvy-Balls verwendet: Ein vollständig gefüllter Kreis bedeutet *komplett erfüllt*, drei Viertel *überwiegend erfüllt*, halb gefüllt *teilweise erfüllt*, ein Viertel gering erfüllt und ein leerer Kreis nicht erfüllt.

Die Evaluation folgt einem zweistufigen Vorgehen. Zunächst wurden exemplarische OCM *Component Descriptors* mithilfe des Prototyps in CycloneDX SBOMs konvertiert. Diese Ausgaben wurden mit dem offiziellen CycloneDX CLI Validator auf Schema- und Semantik Konformität geprüft. Ergänzend wurde jeder Anforderungseintrag aus Kapitel 4 qualitativ analysiert, indem der implementierte Funktionsumfang und die zugrunde liegende Architektur mit den Muss-, Sollte- und Wird-Kriterien abgeglichen wurden. Diese Gegenüberstellung erlaubt eine differenzierte Einschätzung, ob eine Anforderung voll erfüllt, teilweise erfüllt oder verfehlt wurde. Die eigenen Beurteilungen stützen sich auf die Dokumentation des Projekts. Rückmeldungen Dritter fließen hier nicht ein, um die Bewertung konsistent anhand der realisierten Funktionalität vorzunehmen.

6.1 Bewertung der funktionalen Anforderungen

Tabelle 6.1 fasst die zehn funktionalen Anforderungen (FANF #1–#10) zusammen und ordnet den erreichten Erfüllungsgrad zu. Die Beschreibungen orientieren sich an den Formulierungen der Anforderungsanalyse, wurden jedoch aus Gründen der Übersichtlichkeit leicht gekürzt.

ID	Anforderung	Erfüllungsgrad
FANF #1	Transformation OCM → CycloneDX (MUSS). Der Prototyp muss OCM-Komponentendeskriptoren einschließlich Komponenten, Relationen und Metadaten verlustarm in eine CycloneDX-SBOM überführen.	
FANF #2	Ressourcen → Komponenten (MUSS). OCM Ressourcen sollen als CycloneDX-Komponenten modelliert werden; Herkunft und Zugriffsinformationen werden in die Komponentendefinition übertragen.	
FANF #3	Hierarchie und Verknüpfungen (MUSS). Die Komponentenhierarchie (<code>componentReferences</code>) ist durch verschachtelte Komponenten und externe Referenzen (BOM-Links) abzubilden.	
FANF #4	Paketidentität via PURL (MUSS). Die Identität der Pakete wird primär mithilfe von Package URLs (PURL) modelliert. Bei fehlenden PURLs dienen CPE oder SWID als Fallback.	
FANF #5	Aggregation und Zusammenführung (MUSS). Aus SBOMs einzelner Ressourcen ist eine konsolidierte Root-BOM zu erzeugen; Identitäten und Referenzen müssen konsistent bleiben.	
FANF #6	Direktes Mapping ohne Zwischenformate (SOLLTE). Die Abbildung OCM → CycloneDX soll ohne generische Zwischenabstraktionen erfolgen und direkt die offizielle <code>cyclonedx-go</code> -Bibliothek nutzen.	
FANF #7	Identität und Herkunftsbelege (SOLLTE). Für jede SBOM-Komponente sollen Identitätsevidenzen (u. a. Toolname, Version, Fundort, Integritätsmerkmale) und Kontexteinträge nach den <i>Minimum Elements</i> hinterlegt werden.	
FANF #8	Übernahme vorhandener SBOMs (SOLLTE). Bereits existierende SBOMs sind in OCM-Komponenten einzubetten oder extern zu referenzieren, ohne Signaturen zu verändern.	

Tab. 6.1: Erfüllungsgrad der funktionalen Anforderungen.

ID	Anforderung	Erfüllungsgrad
FANF #9	Pedigree/Abstammung (WIRD). Die CycloneDX-Pedigree-Strukturen sollen zur Dokumentation der Herkunft und Veränderungen von Komponenten genutzt werden.	
FANF #10	Bereitstellung (WIRD). Der Prototyp wird in das OCM-CLI-Tooling integriert, sodass die Konvertierung als Kommandozeilen schritt in Build- und Delivery-Pipelines ausführbar ist.	

Tab. 6.1: Erfüllungsgrad der funktionalen Anforderungen (Fortsetzung).

Diskussion. Die Muss-Anforderungen FANF #1 bis FANF #5 bilden die essenzielle Kernfunktionalität des Projekts. Die Ergebnisse zeigen, dass die grundlegende Umwandlung von OCM-Beschreibungen in CycloneDX-SBOMs funktioniert: Das Werkzeug erzeugt valide SBOMs und spiegelt die wichtigsten Komponenten und Beziehungen wider. Einige informationsreiche Felder bleiben jedoch unberücksichtigt, wodurch die Abbildung nur überwiegend erfüllt ist (FANF #1). Die Abbildung von Ressourcen zu Komponenten (FANF #2) beschränkt sich bislang auf den Ressourcentyp `ociImage`. Tiefer verschachtelte Hierarchien und externe Referenzen werden nur teilweise verarbeitet (FANF #3). Die Paketidentifikation mittels PURL (FANF #4) ist voll umfänglich automatisch durch das generieren der R-SBOMs via Syft erreicht. Jedoch momentan nur für `ociImages`. Die Arbeit kümmert sich um diesen Teil nicht selber. Auch die Aggregation mehrerer SBOMs zu einer konsolidierten Root-BOM (FANF #5) erfordert noch Feinschliff, da Identitäten über Dokumentgrenzen hinweg nicht immer konsistent bleiben.

Die Soll-Anforderungen zeigen ein gemischtes Bild: Die direkte Nutzung der offiziellen CycloneDX Bibliothek (FANF #6) ist vollständig realisiert, womit der Prototyp ohne informationsverlustige Zwischenformate (Mittelware, wie `Protobom`) auskommt. Demgegenüber fehlen weitgehend die optionalen Nachweise zur Komponentenidentität (Anfänge durch `properties[]` gemacht) (FANF #7), die Möglichkeit des Konsums bereits eingebletteter existierender SBOMs als Ressourcen (FANF #8) und die Nutzung der Pedigree-Strukturen (FANF #9). Letztere beiden Punkte sind für eine umfassende Historie und Wiederverwendbarkeit von Daten wesentlich. Schließlich steht die Integration in das OCM Toolset (FANF #10) noch aus und ist bisher nur als experimenteller Befehl im Rahmen des Prototyps vorhanden. Dies ist der letzte Schritt und wird erst nach erreichen aller vorhergegangenen Anforderungen in Betracht gezogen.

6.2 Bewertung der nicht-funktionalen Anforderungen

Neben der Funktionalität spielen Qualitätseigenschaften wie Konformität, Kompatibilität und Wartbarkeit eine entscheidende Rolle. Tabelle 6.2 bewertet die sechs nicht-funktionalen Anforderungen (NANF #1–#6).

ID	Anforderung	Erfüllungsgrad
NANF #1	Schema-/Format-Konformität (MUSS). Der Prototyp erzeugt CycloneDX SBOMs gemäß der Spezifikation 1.6.	
NANF #2	Scanner-Kompatibilität (MUSS). Die SBOMs müssen ohne Nacharbeit von verbreiteten OSS-Scannern wie Grype oder Trivy verarbeitet werden können.	
NANF #3	Rückverfolgbarkeit und Round-Trip (MUSS). Jede SBOM-Komponente soll eindeutig auf ihre OCM-Quelle rückführbar sein, um eine verlustarme Rückkonvertierung zu ermöglichen.	
NANF #4	Semantische Vollständigkeit (MUSS). Relevante OCM-Informationen sollen nativ oder über CycloneDX-Properties abgebildet werden, sodass Werkzeuge sie auslesen können.	
NANF #5	Lizenz- & Ökosystem-Kompatibilität (SOLLTE). Der Prototyp soll Apache-kompatible Lizenzen nutzen. Das Repository verwendet die Apache 2.0 Lizenz, wodurch die Wiederverwendung unproblematisch ist.	
NANF #6	Wartbarkeit & Software-Bibliotheken (SOLLTE). Es sollen populäre, aktiv gepflegte Go-Bibliotheken zum Einsatz kommen und veraltete Middleware gemieden werden.	

Tab. 6.2: Erfüllungsgrad der nicht-funktionalen Anforderungen.

Diskussion. Hinsichtlich der Konformität zur CycloneDX-Spezifikation (NANF #1) erfüllt der Prototyp die grundlegenden Validierungsregeln. Jedoch müssen noch Pedigree-Strukturen implementiert werden. Die generierten SBOMs bestehen den offiziellen Validator, weisen jedoch bei komplexen Strukturen vereinzelt Modellierungslücken auf. Die

Scanner-Kompatibilität (NANF #2) ist momentan eingeschränkt: Die Ursache liegt jedoch nicht beim Prototyp, sondern bei den noch unausgereiften OSS-Scannern. Während die erzeugten CycloneDX-SBOMs verschachtelte Komponenten und weitere Spezifikationsmerkmale korrekt abbilden, verarbeiten Scanner, wie Gripe und Trivy diese nur teilweise. Die Rückverfolgbarkeit (NANF #3) ist grundsätzlich gegeben, weil die OCM Komponentennamen auf DNS-konformen Identitäten basieren und diese als `bom-ref` nutzen. Doch bislang ist diese noch nicht implementiert. Die semantische Vollständigkeit (NANF #4) leidet unter fehlenden Lizenz- und Signaturinformationen. Nicht native OCM Felder werden bislang nur sporadisch als Properties hinterlegt. Positiv hervorzuheben ist die Lizenz kompatibilität (NANF #5), da das Projekt unter der Apache 2.0-Lizenz steht und somit diese Anforderung komplett erfüllt. Zudem verwendet der Prototyp etablierte Bibliotheken wie `cyclonedx-go`, sodass eine solide Wartbarkeit gewährleistet ist (NANF #6). Allerdings fehlt eine umfassende Testabdeckung, und die Dokumentation der internen API ist knapp, was spätere Erweiterungen erschwert.

6.3 Erkenntnisse und Limitationen

Die Evaluation zeigt, dass der Prototyp ein tragfähiges Fundament für die Umwandlung von OCM SBoDs in CycloneDX SBOMs bildet. Die Kernfunktionalitäten werden größtenteils erfüllt und die Entscheidung für eine direkte Abbildung ohne Zwischenformate erwies sich als vorteilhaft. Außerdem kann auch klar behauptet werden, dass die vollständige Interoperabilität erreicht werden kann, jedoch noch einiges an Entwicklungsaufwand erbracht werden muss. Dennoch offenbart die Analyse mehrere Schwachstellen:

Lückenhafte Metadaten. Wichtige Informationen wie Lizzenzen, Signaturen, vollständige Abhängigkeitsbäume oder Build-Zeiten werden nur unvollständig übernommen. Dies schränkt die Nutzbarkeit der SBOMs für Compliance- und Security-Analysen bisher noch ein.

Begrenzte Hierarchie- und Aggregationsunterstützung. Die Abbildung tief verschachtelter Komponenten und die konsistente Zusammenführung mehrerer SBOMs bedürfen weiterer Entwicklung, um die Struktur des OCM *Component Trees* vollständig zu erhalten.

Fehlende Unterstützungen für optionale Erweiterungen. Funktionen wie die Einbettung bestehender SBOMs, die Nutzung von Pedigree-Strukturen oder die Hinterlegung von Identitäts-Evidenz sind bislang nicht realisiert, obwohl sie für Transparenz und Auditierbarkeit entscheidend sind. Jedoch steht diesen Anpassungen auch nichts im Weg.

Integrationsreife. Die derzeitige Integration als eigenständige CLI ermöglicht erste Experimente, genügt aber nicht den Anforderungen automatisierter CI/CD-Pipelines. Hier sind stabile Schnittstellen und ein durchgängiger Round-Trip zwischen OCM und Cyclo-
neDX erforderlich.

Dokumentation und Testbarkeit. Eine ausführliche technische Dokumentation, automatisierte Tests und Beispiele für den praktischen Einsatz fehlen. Diese Defizite erschweren die Nachvollziehbarkeit und Wartung des Codes.

Insgesamt unterstreicht die Evaluation den Nutzen des Prototyps als Ausgangspunkt für die automatische Erzeugung von SBOMs aus OCM-Artefakten. Die identifizierten Defizite dienen als Leitfaden für die Weiterentwicklung, um in einem nächsten Schritt die genannten Lücken zu schließen und die Einsatzreife und Sicherheit für produktive Software Lieferketten zu erhöhen.

7 Fazit und Ausblick

Die Arbeit verfolgte das Ziel, einen verlustarmen Konvertierungsansatz vom Open Component Model in den CycloneDX SBOM Standard zu entwerfen und zu evaluieren. Der PoC zeigte, dass eine direkte Abbildung der SBoD als SBOM grundsätzlich tragfähig ist und mit Hilfe der offiziellen `cyclonedx-go` Bibliothek umgesetzt werden kann. Im Rahmen der Evaluation (siehe Kapitel 6) wurden die zehn funktionale und sechs nicht-funktionale Anforderungen geprüft. Die Muss-Anforderungen zur Transformation von Komponenten, Abbildung von Ressourcen, Hierarchie, Paketidentität und Aggregation wurden überwiegend erfüllt, während optionale Aspekte wie die Pedigree-Strukturen, die Einbettung bestehender SBOMs und umfassende Identitätsevidenz weitgehend offen bleiben. Somit erfüllt der Prototyp den Großteil der funktionalen Kernanforderungen. Nicht-funktional überzeugt der Prototyp durch schema konforme SBOMs und eine Apache-kompatible Lizenzierung, weist jedoch Schwächen bei der Scanner-Kompatibilität, Rückverfolgbarkeit und semantischen Vollständigkeit auf. Der Einsatz von *Vulnerability Scannern* wie Grype und Trivy offenbart insbesondere die Herausforderung, verschachtelte Komponenten in CycloneDX zu verarbeiten. Dies dekte sich mit den Interviewbefunden. Während Grype `nested components` erkennt, gibt die Grype Ausgabe zum Beispiel per VEX flache Strukturen zurück und Trivy ignoriert sogar tieferliegende Ebenen komplett. Scanner konsumieren SBOMs oft unterschiedlich und mit format- bzw. tiefenabhängigen Divergenzen (vgl. Bressers, 2025). Diese Erkenntnisse verdeutlichen, dass eine reine Konvertierung in CycloneDX noch keine durchgängige Supply-Chain-Transparenz garantiert. Eine verlustfreie Überführung bleibt jedoch stets im Zielbild erreichbar, wenn Referenzen auf OCM-Ursprungsressourcen konsistent mitgeführt werden (vgl. Möller, 2025a).

Beantwortung der Forschungsfrage Die Forschungsfrage lautete, wie die Informationsstruktur des Open Component Models ohne Informationsverluste in den CycloneDX SBOM Standard überführt werden kann, um Kompatibilität mit etablierten Software-Supply-Chain-Security-Tools zu gewährleisten. Die erarbeitete Lösung basiert auf der direkten Überführung der OCM Modellelemente auf CycloneDX-Konzepte: OCM `resources` werden als CycloneDX `components` repräsentiert, die Komponentenhierarchie wird durch verschachtelte Komponenten und in Zukunft auch durch BOM-Links abgebildet, und Package URLs dienen als primärer Identifikator, der Software Artefakte. Durch den Verzicht auf intermediäre Formate bleibt der Informationsgehalt weitgehend erhalten und die resultierenden SBOMs sind mit gängigen Sicherheitswerkzeugen nutzbar. Gleichzeitig zeigt die Evaluation, dass vollständige Informationsverlustfreiheit nicht erreicht wurde: Metadaten zu Lizenzen, Build-Kontext oder Pedigree werden bislang nur rudimentär oder gar nicht übernommen, wodurch die Rückverfolgbarkeit eingeschränkt ist. Zudem offenbarten die Tests, dass gängige Scanner `nested components` unzureichend auswerten. Die

Antwort auf die Forschungsfrage fällt daher zweigeteilt aus: Es konnte gezeigt werden, dass eine verlustarme Überführung möglich ist und zu verwertbaren SBOMs führt, jedoch sind ergänzende Maßnahmen und Verbesserungen auf Ebene der Datenmodellierung sowie der Werkzeuge erforderlich, um die Kompatibilität mit den meisten Security- und Compliance-Tools zu gewährleisten.

Ausblick und weitere Forschungsschritte Die Arbeit eröffnet verschiedene Anschlussfragen, deren Bearbeitung zur Reife des Konvertierungsansatzes beiträgt:

- **Erweiterung der Metadatenerfassung.** Künftig sollten Lizenzinformationen, kryptographische Signaturen, Build- und Delivery-Zeiten sowie vollständige Abhängigkeitsbäume vollständig in die SBOMs übertragen werden. Dies erfordert eine Erweiterung des Mappings und eine Anreicherung des OCM-Schemas.
- **Unterstützung für bestehende SBOMs und Pedigree.** Die Einbettung und Referenzierung vorhandener SBOMs, ohne deren Signaturen zu verändern, sowie die Nutzung der Pedigree-Strukturen zur Dokumentation von Forks und Patches sind wichtige nächste Schritte, um einen nahtlosen Rundflug von Software-Artefakten zu ermöglichen.
- **Integration in CI/CD-Pipelines.** Eine robuste CLI Integration sollte durch eine Programmierschnittstelle ergänzt werden, damit SBOMs automatisiert in Build- und Delivery-Pipelines generiert, angereichert und gespeichert werden können.
- **Verbesserung der Scanner-Kompatibilität.** Die Tests haben gezeigt, dass aktuelle Scanner nur unzureichend verarbeiten. Hier besteht Forschungsbedarf, sowohl in der Weiterentwicklung der Scanner (durch Kollaboration zu Grype und Trivy) als auch in der Gestaltung von SBOM Strukturen, die von den Tool besser unterstützt werden.
- **Vergleich mit anderen SBOM-Standards.** Eine Untersuchung, wie das Mapping auf SPDX oder andere Standards realisiert werden kann, wäre sinnvoll, um die Übertragbarkeit der entwickelten Prinzipien zu überprüfen und Multiformat-SBOMs zu ermöglichen.

Persönliche Reflexion Die Arbeit vereinte theoretische Auseinandersetzungen mit Supply-Chain-Security-Standards und die praktische Entwicklung eines Prototyps. Besonders eindrücklich war der Einblick in die Komplexität moderner Lieferketten und die Erkenntnis, dass Transparenz kein Selbstzweck ist, sondern aktives Engagement erfordert. Die Konzeption des Mappings schärfte mein Verständnis für die feinen Unterschiede zwischen liefer- und stücklistenorientierten Modellen. Die technischen Herausforderungen bei der Umsetzung, angefangen bei der Modellierung komplexer Komponentenbäume bis hin zur Interaktion mit realen Scannern, machten deutlich, dass Interoperabilität

mehr umfasst als das Einhalten eines Schemas. Ermutigend war die Unterstützung der Open-Source-Community und deren Community-Calls. Gleichzeitig wurde klar, dass viele Tools noch nicht auf die neusten SBOMs Spezifikationen und Fähigkeiten vorbereitet sind. Für mich persönlich ist die Arbeit daher nicht mit dem Abgabetermin abgeschlossen. Ich beabsichtige, meine Erkenntnisse in die Weiterentwicklung und Kollaboration mit *Grype*, um nested components besser zu unterstützen und die prototypische Implementierung zu einem reifen Tool auszubauen. Diese Reflexion bestärkt mich in der Überzeugung, dass Forschung und praktische Open Source Beiträge Hand in Hand gehen müssen, um die Sicherheit und Transparenz von Software zu verbessern.

Anhang A: Interviewleitfaden

Introduction

- Welcome and thank you for participating and for taking the time.
- First of all, I want to apologise because I am right now really nervous and stressed..so that means my english tends to suffer a bit. I appreciate your patience
- Short self-introduction: My name is Olison Sturm -> Oli. I've been working at SAP for three years now, since yesterday. I'm conducting this interview as part of my bachelor's thesis at the Cooperative State University of Baden-Württemberg (DHBW), in collaboration with SAP and the Open Component Model (OCM) (NeoNephos Foundation as part of the Linux Foundation Europe).
- Brief explanation of the topic:
 - I'm investigating transformation strategies for converting complete OCM component descriptors into standardised CycloneDX SBOM.
- Important notes:
 - This is a **semi-structured** interview.
 - It will last about **40 minutes**.
 - Your participation is voluntary, and anonymity is ensured upon request.
 - **Recording permission:**
“*Do I have your permission to record this session?*”
- **Interview objective:**
The goal of this interview is to identify technical requirements for transforming OCM component descriptors into CycloneDX SBOMs. We are especially interested in the necessary content, structures, modelling strategies, tool expectations, and best practices. The insights will help derive concrete requirements for a structured and semantically meaningful mapping between both formats – ultimately contributing to answering the following research question:

Research Question

How can the information structure of the Open Component Model (OCM) be converted into the CycloneDX SBOM format with minimal information loss to enable compatibility with established software supply chain tools?

Optional:

Give a short overview of a prepared OCM descriptor (including componentReferences, extraIdentity, and other fields), as visual context will help the expert answer more concretely.

Background Questions (Expertise Context)

- H1:** How long have you been working in the field of Software Supply Chain Security?
- H2a:** What is your current role or responsibility in this field?
- H2b:** What role do SBOMs play in your work?
- H3:** In what scenarios do you see the need to transform OCM-like structures into a standardised SBOM format such as CycloneDX?
- H4:** Which real-world use cases do you foresee for SBOMs generated from OCM components – e.g., audits, policy enforcement, deployment, vulnerability scanning?

Core Questions (General, for All Participants)

- K1:** What tools or technologies do you know or use for generating, merging, or transforming SBOMs – particularly in the context of CycloneDX?

K2:

- *For OCM experts:* Which OCM structures do you consider especially important so that external systems, like scanner, can process the generated SBOMs?

- *For others:* Which CycloneDX structures are most critical for ensuring downstream tool compatibility?
- K3:** Which elements or structures from OCM can, in your experience, be mapped directly and without data loss to CycloneDX?
- K4:** Which OCM concepts are difficult or impossible to represent in CycloneDX?
- **Follow-up:** Do you know of any workarounds?
- K5:** Which options exist for modelling component hierarchies from OCM in CycloneDX (e.g., dependencies, bom-links, embedded components)? Which option do you consider most suitable?
- K6:** From your perspective, what are essential quality criteria that a SBOM generated from OCM should fulfil?
- K7:** Based on your experience, are there best practices for designing such a mapping?
- K8:** What are potential risks or limitations of pursuing a direct OCM-to-CycloneDX mapping?

Group-Specific Follow-Up Questions

OCM Experts

- O1:** Which OCM concepts do you consider critical for downstream processing in other formats like SBOMs?
- O2:** How do you typically handle metadata in OCM (e.g., labels, identities, repositoryContexts) and which of these should be passed on to SBOMs?
- O3:** Are there conventions in OCM that others should be aware of when transforming descriptors to other formats?
- O4:** What kind of tooling would help streamline a compliant OCM-to-SBOM mapping?
- O5:** At which stage in the lifecycle of an OCM component do you think it makes sense to capture the SBOM-relevant data (e.g., license info, package manifests, build metadata)?

CycloneDX Experts

- C1:** Which CycloneDX fields and relationships are especially important for SBOM usability in practice (e.g., for analysis, security, compliance tools)?
- C2:** Do you have a preferred approach to modelling hierarchies in CycloneDX – among bom-links, dependencies, or embedded components?
- C3:** What are common mistakes or misunderstandings you see in how other projects use CycloneDX?
- C4:** How flexible is CycloneDX in your opinion – e.g., when extending structures via properties or custom fields?
- C5:** Are there any planned changes or extensions to CycloneDX that could be relevant for this mapping effort?
- C6:** Do you think using an intermediate abstraction like **Protobom** is a better approach than mapping directly to CycloneDX or SPDX?

SBOM Scanner Expert

- S1:** Which CycloneDX fields does your tool rely on for detecting vulnerabilities, licenses, or policy violations?
- S2:** Are there SBOM fields or formats you find problematic – e.g., too vague, redundant, or underspecified?
- S3:** What conditions must a CycloneDX SBOM meet to be accepted or correctly parsed by your tool? (e.g., how does a Syft SBOM behave in Trivy?)
- S4:** How important is proper hierarchy modelling for your tool's analysis processes?
- S5:** Do you know of limitations when processing dependencies, bom-links, or nested components?
- S6:** Could you envision support for OCM descriptors in your tool someday?
- S7:** How are SBOMs typically used in practice? How are the results of your tool further utilised? Why was it acceptable in your project not to model component hierarchies?

Wrap-Up & Closing

- Brief summary of key takeaways (optional)

- Closing questions:
 - **A1:** Do you believe a unified SBOM format will eventually emerge as the de facto standard?
 - **A2:** What do you consider to be non-negotiable requirements for a reliable OCM-to-SBOM mapping?
 - **A3:** Is there anything else you would like to add that we haven't discussed?
- Info about next steps: Transcription & qualitative content analysis (following Mayring)
- Optionally: Offer review/fact-check of their transcript
- **Final note from OCM team:**

“On behalf of the OCM team, I'd also like to invite you to collaborate with us. We'd greatly appreciate your perspective and future contributions.”
- Thank you very much for your time and valuable input!

Anhang B: Experteninterview mit Jakob Möller

- 1 [0:00:00.0]
- 2 [0:00:07.4] Olison Sturm: Okay, danke für deine Geduld, wir können jetzt anfangen.
- 3 [0:00:10.4] Jakob Möller: Super, kein Problem.
- 4 [0:00:13.2] Jakob Möller: Dann leg mal los mit deinem Fireflies.AI Notetaker, Olison.
- 5 [0:00:17.2] Olison Sturm: Ja, der ist super.
- 6 [0:00:18.7] Olison Sturm: Also der macht es wirklich top auf Englisch zumindest auf Deutsch muss ich sagen, habe ich es noch nicht ausprobiert, aber.
- 7 [0:00:23.4] Jakob Möller: Ich denke, wie gesagt, also wir können ohne Probleme auf Englisch switchen.
- 8 [0:00:27.2] Jakob Möller: Also das musst du mir sagen.
- 9 [0:00:28.4] Jakob Möller: Ich kann auch jederzeit auf Englisch, das ist für mich gar kein Problem.
- 10 [0:00:32.6] Olison Sturm: Wir machen es einfach Deutsch.
- 11 [0:00:34.0] Olison Sturm: Okay, also ich würde einfach kurz anfangen nochmal.
- 12 [0:00:39.7] Olison Sturm: Also wir würden ein semi strukturiertes Interview machen, heißt, ich habe ein paar Nachfragen auch zu gewissen Fragen, wenn ich es nicht ganz verstehe oder wenn ich denke, es wäre wichtig für die Arbeit, dass es noch mal ein bisschen genauer ausgeführt wird. Ansonsten kann ich gerne auch alles anonym veröffentlichen in der Arbeit.
- 13 Jakob Möller: Brauchst du nicht.
- 14 [0:00:55.0] Olison Sturm: Okay, also alles, was du sagst, ist auch öffentlich zugänglich.
- 15 [0:01:00.8] Jakob Möller: Ja, verstanden.
- 16 [0:01:03.9] Olison Sturm: Wunderbar. Ansonsten muss ich dich noch einmal fragen, ob ich das Ganze transkribieren darf?
- 17 [0:01:07.0] Jakob Möller: Ja.
- 18 [0:01:09.1] Olison Sturm: Und dann würde ich anfangen mit dem Ziel für dieses Interview ist eigentlich schlussendlich einfach die technischen Anforderungen für mein Proof of Concept zu erlangen, dann eben die Transformation von OCM Component Beschreiben eben auf CycloneDX zu übertragen, auf eine CycloneDX SBOM.
- 19 [0:01:32.2] Olison Sturm: Und wichtig wäre dabei halt irgendwie rauszufinden,

- was für Inhalte, welche Strukturen, was vielleicht eine gewisse Modellierungsstrategie wäre und welche Tools hilfreich wären.
- 20 [0:01:45.7] Olison Sturm: Genau, dann würde ich noch einmal die Forschungsfrage vorlesen, weil ich glaube, die hast du bisher noch nicht gehört.
- 21 [0:01:51.2] Olison Sturm: Wie lässt sich die Informationsstruktur des Open Component Models möglichst verlustfrei in das CycloneDX SBOM Format konvertieren, um die Nutzung von etablierten Software Lieferkettentools zu ermöglichen?
- 22 [0:02:05.3] Jakob Möller: Ist klar.
- 23 [0:02:07.0] Olison Sturm: Genau, bei den anderen Interviewen habe ich jetzt immer noch Kontext zum OCM gegeben, das würde ich jetzt einfach skippen.
- 24 [0:02:13.2] Olison Sturm: Super.
- 25 [0:02:14.3] Olison Sturm: Genau, dann würde ich schon direkt mit Hintergrundfragen anfangen.
- 26 [0:02:17.3] Olison Sturm: Zu deiner Expertise, da wäre jetzt mir ganz wichtig, seit wann arbeitest du denn schon im Kontext der Software Supply Chain Security?
- 27 [0:02:26.7] Jakob Möller: Also das erste Mal habe ich mit Supply Chain Security angefangen vor sieben Jahren und habe dort damit gearbeitet, SBOMs dynamisch auszulesen, dort Security Vulnerabilities für ein Großunternehmen zu identifizieren und dann dynamisch zu patchen.
- 28 [0:02:46.4] Jakob Möller: Und das erste Mal angefangen zu arbeiten, konkret mit einem SBOM Framework habe ich über das OCM das ist jetzt fünf Jahre her, da habe ich es als Verwender verwendet und das erste Mal wirklich im OCM mitgearbeitet, würde ich sagen, habe ich so vor zwei Jahren.
- 29 [0:03:03.2] Olison Sturm: Okay, und was ist deine derzeitige Rolle in deinem Team?
- 30 [0:03:12.1] Jakob Möller: Ja, also ich bin momentan der Lead Architect vom OCM und habe damit auch die Verantwortung des OCM Projekt, ich sag mal, in der Open Source Schirmherrschaft unter NeoNephos zu begleiten.
- 31 [0:03:27.6] Jakob Möller: Dort bin ich Technical Advisory Council und Technical Steering Committee Member.
- 32 [0:03:32.1] Jakob Möller: Das bedeutet, ich bin praktisch die Ansprechperson, wenn es darum geht, neue Projektinitiativen zu starten, die auch in die Community zu bringen, Abstimmungen zu organisieren, solche Sachen.
- 33 [0:03:43.9] Jakob Möller: Und wenn es Produktvisionen geht, dann bin ich natürlich auch einer der Maintainer dort.

Expertise & Kontext

- Expertise & Kontext
- 34 [0:03:48.1] Jakob Möller: Das heißt, ich bin praktisch auch beteiligt an der Mitgestaltung der OCM Kernspezifikation.
 - 35 [0:03:57.7] Olison Sturm: Okay, und wo genau oder wie genau spielen jetzt SBOMs eine Rolle beim OCM?
 - 36 [0:04:03.2] Jakob Möller: Also OCM beschreibt sich ja selber als Software Bill of Delivery.
 - 37 [0:04:07.7] Jakob Möller: Das bedeutet, OCM beschreibt sich in einem größeren Scope als ein klassisches SBOM.
 - 38 [0:04:12.6] Jakob Möller: Ein SBOM beschreibt ja einen Zustand einer Software an einem Zeitpunkt X in seiner vollen Detaillierung, ist ja Bill of Materials, also welche Materialien enthält jetzt eine Software, richtig dumm gesprochen.
- OCM-spezifische Strukturen
- 39 [0:04:25.4] Jakob Möller: Und eine Bill of Delivery nimmt eben bereits existierende Softwareartefakte, sie von A nach B zu schieben.
 - 40 [0:04:32.3] Jakob Möller: Das bedeutet, wir sehen SBOMs und SBoDs als sehr nah verwandt, aber als nicht äquivalent.
 - 41 [0:04:38.2] Jakob Möller: Und der Grund, warum OCM auch existiert, ist, weil ein klassisches SBOM nicht ausreicht, eine Softwarelieferung komplett zu beschreiben.
- Konkrete Anforderungen
- Mapping-Kompatibilität
- 42 [0:04:47.0] Jakob Möller: Natürlich haben wir trotzdem Interesse daran, so nah wie möglich an SBOM als Konzept dranzubleiben, weil wir natürlich auch Kompatibilität mit existierenden Ökosystemen ermöglichen wollen.
 - 43 [0:05:00.4] Olison Sturm: OK, wunderbar.
 - 44 [0:05:01.8] Olison Sturm: Und schlussendlich, in welchem Verhältnis stehst du zu meiner Arbeit, also zu meiner Thesis?
 - 45 [0:05:10.8] Jakob Möller: Also ich bin ja Teil des OCM Teams, was dich ja auch mit betreut und als solcher habe ich natürlich auch schon Einsichten in deine Arbeit gehabt, aber ich habe bis jetzt keine Einfluss genommen auf die Kontrolle der Lieferung deiner, ich sag mal, deiner Arbeitsresultate.
- Expertise & Kontext
- 46 [0:05:28.6] Jakob Möller: Das bedeutet, ich bin praktisch Consulting, würde ich sagen.
 - 47 [0:05:31.8] Jakob Möller: Also wenn du Fragen hast zum OCM, dann haben wir öfter schon mal zusammen miteinander gesprochen, aber du hast größtenteils in Eigenregie gearbeitet.
 - 48 [0:05:41.7] Olison Sturm: Ja.
 - 49 [0:05:43.3] Olison Sturm: Und dann jetzt schon die erste Frage, die jetzt so konkret meiner Forschung beiträgt: Wo siehst du denn den Nutzen, also den

	Use Case von SBOMs tatsächlich für OCM?
50	[0:05:59.1] Olison Sturm: Also dass man den quasi, dass man aus einer OCM Komponente eine SBOM machen kann in Bezug auf Schwachstellenanalyse, Policy Checks und so weiter.
51	[0:06:12.0] Jakob Möller: Also es ist ja allgemein bekannt, dass die meisten Open Source Toolings, die momentan auf dem Markt existieren, mit SBOMs interagieren, weil sie diesen Snapshot in Time eines SBOMs verwenden um Scans durchzuführen oder Analysen durchzuführen.
52	[0:06:28.8] Jakob Möller: Eine SBoD ist natürlich etwas größer gefasst, weil sie natürlich den ganzen Transfer abdeckt von einer Umgebung in die andere.
53	[0:06:36.2] Jakob Möller: Nichtsdestotrotz sehe ich einen extremen Mehrwert da drin, ein Delivery Artefakt, was dann einmal in einem Cloud Environment oder in einem On Premise Environment liegt, zu konvertieren in so einen, ich sag mal Snapshot in so ein Statement of Time, das dann wieder repräsentiert werden kann über eine SBOM.
54	[0:06:56.0] Jakob Möller: Das bedeutet, der Scope der SBoD ist zwar größer, aber es ist ja nur dann relevant, wenn wir tatsächlich von einer Umgebung in die nächste wollen.
55	[0:07:05.3] Jakob Möller: Und in dem Moment, wo wir in einer Umgebung sind, können wir zumindest aus meiner Sicht gerade wahrscheinlich auch relativ verlustfrei von einem Format in das andere konvertieren, ohne Informationsverlust darzustellen.
56	[0:07:18.3] Olison Sturm: Okay, danke.
57	[0:07:19.9] Olison Sturm: Das war schon alles zum Hintergrund und zur Einordnung deiner Expertise.
58	[0:07:25.2] Olison Sturm: Jetzt würde ich zu den Kernfragen kommen.
59	[0:07:27.1] Olison Sturm: Die Kernfragen sind extra so gestellt, dass die vergleichbar sind mit allen anderen Experten.
60	[0:07:31.8] Olison Sturm: Also ich habe drei Expertengruppen.
61	[0:07:33.8] Olison Sturm: Einmal die Scanner Experten, also die am Ende eine SBOM konsumieren.
62	[0:07:39.4] Olison Sturm: Dann einmal die CycloneDX Experten und einen OCM Experten.
63	[0:07:45.3] Olison Sturm: Genau, die sind nur ganz wenig angepasst, aber geben am Ende die gleiche Antwort.
64	[0:07:50.6] Olison Sturm: Also genau.

- Konkrete Anforderungen
- [65] [0:07:53.2] Olison Sturm: Fangen wir mit der ersten Frage an.
 - [66] [0:07:54.7] Olison Sturm: Welche Tools oder Technologien kennst du und verwendest du oder würdest du zur Erstellung, zum Mergen und zum Transformieren von SBOMs verwenden, wenn du deine Erfahrung nutzen müsstest?
 - [67] [0:08:09.8] Jakob Möller: Ja, also da würde ich natürlich darauf zurückgreifen, was bereits im Open Source Ökosystem vorhanden ist.
 - [68] [0:08:16.2] Jakob Möller: Da kommt es jetzt natürlich darauf an, welche Form von SBOM wir uns anschauen.
 - [69] [0:08:20.1] Jakob Möller: Wenn wir uns jetzt auf eine Spezifikation, zum Beispiel CycloneDX konzentrieren wollen, dann gibt es ja etabliertes Tooling auch von CycloneDX selbst, also dieselben Menschen, die praktisch die Spezifikation stellen ja auch Referenz Tooling zur Verfügung.
 - [70] [0:08:34.8] Jakob Möller: Das könnte man nutzen, damit zu interagieren.
- Tool- und Technologieeinsatz
- [71] [0:08:36.9] Jakob Möller: Es gibt auch Community Toolings, die Konvertierungen anbieten.
 - [72] [0:08:40.5] Jakob Möller: Da gibt es Toolings, die ich glaube auch schon von dir erforscht wurden, wie Protobom.
 - [73] [0:08:46.4] Jakob Möller: Allgemein sind diese Spezifikationen offen und das heißt, es ist auch meiner Meinung nach sehr einfach und ich sag mal, leicht zugänglich, sein eigenes Tooling zu schreiben.
- Best Practices
- [74] [0:08:56.6] Jakob Möller: Das heißt, wenn ich jetzt was bauen müsste, was auf diesen SBOMs aufbaut, dann würde ich erstmal schauen, gibt es das schon auf dem offenen Markt in irgendeiner Form, zum Beispiel Security Scanner würde ich jetzt nicht neu bauen.
 - [75] [0:09:06.7] Jakob Möller: Da gibt es ja etablierte Tools wie Trivy beispielsweise oder Insert SBOM Aware Security Tool hier.
- Konkrete Anforderungen
- [76] [0:09:15.2] Jakob Möller: Aber wenn es Custom Tooling geht, beispielsweise wenn du interne Prozesse modellieren willst, dann bleibt dir ja oft nichts anderes übrig, als direkt mit der Spezifikation zu arbeiten.
 - [77] [0:09:26.3] Olison Sturm: Okay, die zweite Frage wäre: Welche OCM Strukturen sind aus deiner Erfahrung besonders wichtig für SBOMs am Ende?
 - [78] [0:09:38.2] Olison Sturm: Also damit die dann eben in weiteren Tools wie Vulnerability Scannern genutzt werden?
- OCM-spezifische Strukturen
- [79] [0:09:43.4] Olison Sturm: Also welche OCM Strukturen sind wirklich signifikant wichtig und müssen überführt werden?
 - [80] [0:09:48.9] Jakob Möller: Also im OCM unterscheiden wir grundsätzlich zwischen Artefakttypen, Ressourcen und Sourcen.

	81	[0:09:55.5] Jakob Möller: Bei uns sind Ressourcen aktive Lieferartefakte, die von einer Umgebung in die nächste übertragen werden müssen und Sourcen sind Source Code Artefakte, aus denen Ressourcen generiert werden.
CycloneDX-spezifische Struktur	82	[0:10:07.1] Jakob Möller: Das bedeutet, die wichtigsten Artefakte, die wir haben, sind Ressourcen und diese Ressourcen können auch von einem Standpunkt in den nächsten, hatte ich ja bereits erwähnt, umgemappt werden in bei CycloneDX zum Beispiel Komponenten, die dann wiederum einen SBOM in sich behalten.
OCM-spezifische Strukturen	83	[0:10:23.9] Jakob Möller: Wir haben in den Ressourcen dann einen sogenannten Access und dieser Access beschreibt, wo das Lieferartefakt herkommt und die Kombination von Metainformation, was diese Ressource beinhaltet, plus Access, wo diese Ressource herkommt, die kann meiner Meinung nach gut umgemappt werden in eine SBOM Komponente, die dann auch sich beschreiben lässt.
Mapping-Kompatibilität	84	[0:10:48.9] Olison Sturm: Okay, das ist die nächste Frage tatsächlich.
	85	[0:10:57.0] Olison Sturm: Wie siehst du die Möglichkeit, Hierarchien vom OCM, also die hierarchische Struktur vom OCM im CycloneDX abzubilden?
	86	[0:11:07.7] Jakob Möller: Wie du sicher weißt, gibt es ja auch im OCM die Möglichkeit, Beschreiber untereinander zu referenzieren.
OCM-spezifische Strukturen	87	[0:11:14.8] Jakob Möller: Wir verwenden das unter anderem, verschiedene Softwareprodukte miteinander zu koppeln und Abhängigkeiten zu modellieren.
Modellierungsstrategien	88	[0:11:21.4] Jakob Möller: Wir nennen das ganze Component Referenzen, weil eine Komponente referenziert eine andere Komponente, die dann wiederum Ressourcen enthalten kann.
CycloneDX-spezifische Struk	89	[0:11:30.5] Jakob Möller: Und das Schöne ist praktisch, dass diese Komponentenmodellierung, diese Referenzierung 1 zu 1 in CycloneDX übertragen werden kann, weil auch CycloneDX über Referenzfähigkeiten verfügt, andere BOMs, also andere Bill of Materials aus anderen etablierten, schon bereits vorhandenen Boms referenzieren kann.
Mapping-Kompatibilität	90	[0:11:51.3] Jakob Möller: Und die Möglichkeit von einer globalen Identität ist sowohl bei OCM als auch bei CycloneDX vorhanden, zumindest spezifiziert.
Konkrete Anforderungen	91	[0:12:00.3] Jakob Möller: Das heißt, solange man diese globalen Identitäten erhält, erhält man sogar eine semantisch äquivalente Struktur.
Modellierungsstrategien	92	[0:12:07.8] Olison Sturm: Ja, also ganz konkret wäre das dann in CycloneDX, was du meinst, External References und dann einen BOM Link, BOMs zu verlinken oder hast du da was?
	93	[0:12:19.4] Jakob Möller: Da gibt es verschiedene Möglichkeiten.
	94	[0:12:21.0] Jakob Möller: Jetzt kommt es natürlich ins Implementierungsdetail.

	95	[0:12:23.9] Jakob Möller: Man kann solche Bäume natürlich jetzt sowohl asynchron als auch synchron betrachten.
	96	[0:12:28.7] Jakob Möller: Ich meine damit, dass man den Baum komplett traversieren kann oder man behandelt nur einen Knoten und behandelt die Referenzen als gegeben.
	97	[0:12:36.7] Jakob Möller: Und da gibt es natürlich Unterschiede.
CycloneDX-spezifische Strukturen	98	[0:12:38.8] Jakob Möller: Wenn wir Softwareartefakte ausliefern, dann wollen wir meistens den kompletten Baum ausliefern und dementsprechend können wir natürlich auch den ganzen Baum abtraversieren.
	99	[0:12:47.8] Jakob Möller: Äquivalent kann man das natürlich auch bei einem SBOM machen.
	100	[0:12:50.9] Jakob Möller: Du kannst natürlich alle Referenzen, die in deinem SBOM spezifiziert sind, auflösen und in einem, ich sag mal, abgeflachten Oberbaum darstellen.
Modellierungsstrategien	101	[0:13:00.4] Jakob Möller: Du kannst aber genauso gut sagen, du gehst nur auf eine Ebene N und lässt alles andere nur referenziert, weil du sagst, die hast du vielleicht schon in deinem System.
	102	[0:13:10.1] Jakob Möller: Da gibt es dann Unterschiede.
	103	[0:13:11.7] Jakob Möller: Ich glaube, dass da der Wert davon abhängig ist und wie du es auch umsetzen sollst, davon abhängig ist, was du gerade genau machen willst.
Tool- und Technologieeinsatz	104	[0:13:20.1] Jakob Möller: Also wenn du jetzt beispielsweise Verwundbarkeiten identifizieren willst, dann reicht es meistens nicht, den kompletten Baum nur zu analysieren, sondern du hast ja oft Vulnerability Datenbanken, die dann wiederum Referenzen auf bereits gescannte Komponenten haben.
	105	[0:13:36.1] Jakob Möller: Das heißt, es macht dann oft Sinn, deinen obersten Baum sich anzuschauen, dann zu gucken, okay, gibt es jetzt für meine Referenzen vielleicht schon Scan Result, die du einbinden kannst?
	106	[0:13:47.2] Jakob Möller: Und wenn nein, dann musst du die halt auch scannen.
	107	[0:13:50.2] Olison Sturm: Ja, verstehe.
	108	[0:13:52.9] Olison Sturm: Und wie würdest du sagen, dass wir es hinkriegen, aus diesen SBoD Strukturen dann ohne Informationsverlust das in CycloneDX zu verführen?
	109	[0:14:05.3] Olison Sturm: Du sagst einfach, also wenn ich deine Antwort davor richtig verstanden habe, sagst du ja, dass das alles überführbar ist, weil es immer irgendwie was in CycloneDX gibt, die Strukturen zu überführen.

	110	[0:14:17.3] Olison Sturm: Würdest du sagen, dass das alles verlustfrei möglich ist?
Konkrete Anforderungen	111	[0:14:20.3] Jakob Möller: Also ich glaube tatsächlich, wenn man komplett rekursiv auflöst, dann kann man komplett verlustfrei auflösen unter der Annahme, dass wir eine Rückführung ermöglichen.
Mapping-Kompatibilität	112	[0:14:30.3] Jakob Möller: Also es ist natürlich wichtig, im SBOM Referenzen auf die ursprünglichen Ressourcen im OCM zu halten, weil ansonsten kannst du diese Rückkonvertierung nicht mehr sicher durchführen oder diese Rückreferenzierung.
Best Practices	113	[0:14:42.5] Jakob Möller: Aber allgemein ist es, glaube ich, schon möglich, auch selbst mit einem nicht voll rekursiv aufgelösten Baum eine verlustfreie Konvertierung zu ermöglichen, weil sowohl SBOM als auch, also beziehungsweise CycloneDX als Implementierung von einem SBOM Ansatz, als auch OCM in dem Fall die Möglichkeit erlauben, Gegenreferenzierungen aufzubauen, zum Beispiel über Labels oder Annotationen oder wie auch immer man das nennen will, also arbiträre Set an Metadaten.
Qualitätskriterien	114	[0:15:10.8] Olison Sturm: Okay, ja, verstehe.
	115	[0:15:18.2] Olison Sturm: Okay, die Frage haben wir mehr oder weniger schon beantwortet.
	116	[0:15:21.2] Olison Sturm: Ich würde dann einfach mal die Frage überspringen.
	117	[0:15:28.8] Olison Sturm: Was sind für dich essentielle Kriterien an die Qualität meinens Proof of Concepts bzw.
	118	[0:15:35.8] Olison Sturm: Generell beim Generieren einer SBOM aus einer SBoD?
	119	[0:15:43.0] Jakob Möller: Ich glaube, das wichtigste Kriterium für mich ist die Verwendbarkeit von dem resultierenden SBOM OCM Artefakt.
Qualitätskriterien	120	[0:15:50.9] Jakob Möller: Das bedeutet, wenn wir starten mit einem OCM Komponentenbeschreiber, dann muss dieser Komponentenbeschreiber in eine Art und Weise konvertiert werden, in der er auch verwendbar von anderen Tools ist, die bereits mit SBOMs umgehen können.
Konkrete Anforderungen	121	[0:16:03.3] Jakob Möller: Das bedeutet, es hilft uns nicht, in einen SBOM zu konvertieren, aber dann die Felder so darzustellen, dass kein Tool damit arbeiten kann.
Best Practices	122	[0:16:11.3] Jakob Möller: Das heißt, wir müssen etablierte Konventionen nutzen, beispielsweise Globalität sicherzustellen.
	123	[0:16:16.2] Jakob Möller: Das hatten wir ja gerade eben angesprochen.
Konkrete Anforderungen	124	[0:16:18.8] Jakob Möller: Und wenn wir das sichergestellt haben, dann haben wir vorausgesetzt, okay, dieses SBOM, was dann aus der Konvertierung rauskommt, das kann jetzt für andere Tools verwendet werden, zum Beispiel
Tool- und Technologieeinsatz		

Konkrete Anforderungen	[125] einen Verwundbarkeitsscan Umgekehrt ist es aber auch wichtig, dass dieses SBOM wieder komplett rückreferenzierbar ist.
	[126] [0:16:38.1] Jakob Möller: Wenn wir es nicht schaffen, eine Rückreferenzierung auch möglich zu machen, dann hilft uns das Scan Resultat aus dem SBOM vielleicht in dem Moment gar nichts, weil ich ja gar nicht mehr weiß, von welcher Ressource das abhängig ist.
	[127] [0:16:51.1] Jakob Möller: Das heißt, ein richtiger Mehrwert entsteht nur dann, wenn ich tatsächlich eine Übertragung der Informationen vom SBOM wieder zurück in die OCM Ressourcen ermögliche.
	[128] [0:17:02.2] Jakob Möller: Das bedeutet, wenn ich es schaffe, die semantische Verbindung aufzubauen von dieser Ressource, die gerade in diesem Environment deployed ist, hat diese Verwundbarkeiten und muss entsprechend geprüft werden.
Qualitätskriterien	[129] [0:17:14.8] Jakob Möller: Wenn ich diesen Link aufbauen kann, dann ist für mich ein signifikanter Mehrwert gegeben.
Tool- und Technologieeinsatz	[130] [0:17:19.0] Jakob Möller: Und natürlich, wenn wir andere Tools betrachten, dann ist der Mehrwert entsprechend anders.
	[131] [0:17:23.6] Jakob Möller: Das heißt, ich habe eigentlich zwei Kriterien für mich.
	[132] [0:17:26.5] Jakob Möller: Das eine ist Rückverfolgbarkeit nach der Konvertierung, dass ich auch wieder rückkonvertieren kann oder zumindest rückreferenzieren kann.
Konkrete Anforderungen	[133] [0:17:33.3] Jakob Möller: Und das andere Kriterium ist semantische Vollständigkeit und auch Sinnhaftigkeit im SBOM.
	[134] [0:17:39.2] Jakob Möller: Das heißt, es müssen alle Felder auch wirklich korrekt referenziert sein, auch korrekt übertragen worden sein und müssen auch im SBOM, im CycloneDX SBOM in dem Fall von anderen Tools auch auslesbar sein.
	[135] [0:17:52.2] Jakob Möller: Das heißt, wir dürfen keine neue Zwischenspezifikation schaffen, die dann wiederum von anderen Tools nicht ausgelesen werden kann.
	[136] [0:18:00.3] Olison Sturm: Verstehe.
Best Practices	[137] [0:18:01.0] Olison Sturm: Hast du konkret jetzt, die zwei Kriterien zu erreichen, irgendwelche Best Practices oder irgendwelche Erfahrungen, die mir dabei helfen könnten?
	[138] [0:18:08.5] Jakob Möller: Also ich glaube, was wir auf jeden Fall haben, das hatte ich ja auch schon angesprochen, ist die globale eindeutige Identität.
OCM-spezifische Strukturen	[139] [0:18:16.1] Jakob Möller: Das bedeutet, eine Komponente im OCM muss eine

		DNS Compliant Domain sein.
OCM-spezifische Strukturen	140	[0:18:20.8] Jakob Möller: Und das machen wir deshalb, weil wir ja gesagt haben, wir haben Referenzen auf andere Komponenten und wir müssen immer sicherstellen können, dass diese Referenzen auch global eindeutig auflösbar sind.
Best Practices	141	[0:18:30.7] Jakob Möller: Und im CycloneDX SBOM gibt es da äquivalente Strukturen, die dir auch ermöglichen, über diverseste Identifikationsstrukturen auch so eine global eindeutige Identität zuzuweisen.
CycloneDX-spezifische Strukturen	141	
Tool- und Technologieeinsatz	142	[0:18:42.9] Jakob Möller: Das können dann Scanner wieder benutzen, die Scan Resultate für diese Identität zu speichern.
Best Practices	143	[0:18:48.5] Jakob Möller: Und damit könnten wir beispielsweise ein Scannernetzwerk aufbauen.
	144	[0:18:53.9] Olison Sturm: Okay, wunderbar.
	145	[0:18:55.4] Olison Sturm: Und jetzt noch zum Abschluss der Kernfragen.
	146	[0:18:59.0] Olison Sturm: Was spricht deiner Meinung komplett bzw. gegen diesen Ansatz überhaupt SBoDs in SBOMs zu übertragen?
	147	[0:19:08.2] Jakob Möller: Also das erste, was wir natürlich auch am Anfang angesprochen haben, ist, dass SBoDs ja eigentlich ein transientes Verhalten beschreiben sollen, also ein Verhalten in einer Zielumgebung, wohingegen eine SBOM ein statisches Artefakt ist, was sich nie ändert.
CycloneDX-spezifische Strukturen	148	[0:19:23.0] Jakob Möller: Das bedeutet, ein SBOM beschreibt ja eigentlich nur einen Status Quo von einem Software Artefakt, zum Beispiel in einem gewissen Zustand.
	149	[0:19:31.2] Jakob Möller: Das enthält ja Abhängigkeiten und die ändern sich nicht.
Kritik & Gegenargumente	150	[0:19:33.7] Jakob Möller: Also wenn du einmal eine Version von einer Software released hast, dann wird die auf ewig so bleiben.
	151	[0:19:38.4] Jakob Möller: Da gibt es dann nicht auf einmal Unterschiede.
	152	[0:19:40.7] Jakob Möller: Bei einem SBoD ist das anders.
	153	[0:19:42.4] Jakob Möller: Ein SBOM verändert sich in dem Moment, wo du die Zielumgebung anpasst.
	154	[0:19:47.9] Jakob Möller: Wenn du jetzt eine Konvertierung durchführst, dann ist genau genommen diese Boundary nicht mehr gegeben, dass du ein statisches Artefakt hast, weil es ist ja dann nicht mehr statisch, sondern du hast ja dann angepasste Ziellooker.
	155	[0:19:59.8] Jakob Möller: Ich glaube, das wäre, ich sag mal, konzeptionell

		zumindest schwammig.
Kritik & Gegenargumente	156	[0:20:04.1] Jakob Möller: Da gibt es kein abgedecktes.
	157	[0:20:05.6] Jakob Möller: Okay, ist das jetzt dasselbe SBOM, ja oder nein?
	158	[0:20:08.5] Jakob Möller: Ist das jetzt dieselbe Identität?
	159	[0:20:10.7] Jakob Möller: Weil nach SBOM wäre das eigentlich eine andere Identität.
	160	[0:20:14.8] Olison Sturm: Okay, wunderbar.
	161	[0:20:15.7] Olison Sturm: Ich glaube, damit spielt ja auch so der Lebenszyklus von SBoDs rein.
	162	[0:20:20.3] Olison Sturm: Also wir haben ja irgendwie, keine Ahnung, gewisse Zustände von unseren, also keine Ahnung, wir haben z.B. die Build Time.
	163	[0:20:31.6] Olison Sturm: In welchen Lebenszyklus wird das am besten reinpassen?
	164	[0:20:34.2] Olison Sturm: Also wann würde die Generierung von der SBOM Sinn ergeben?
Konkrete Anforderungen	165	[0:20:38.3] Jakob Möller: Also da gibt es zwei Möglichkeiten.
	166	[0:20:40.9] Jakob Möller: Du kannst entweder zur Build Time oder zur Delivery Time, nennt sich das bei uns so ein SBOM, generieren.
OCM-spezifische Strukturen	167	[0:20:47.0] Jakob Möller: Build Time würde bedeuten, das erste Mal, wo du einen Beschreiber bei uns im OCM erstellst, zum Beispiel aus deinem Continuous Integration System, aus deiner Pipeline.
	168	[0:20:57.1] Jakob Möller: In dem Moment könntest du ja alle Artefakte als Ressourcen beschreiben.
Tool- und Technologieeinsatz Best Practices	169	[0:21:01.4] Jakob Möller: Die wären dann da, die könntest du auch scannen und dann könntest du dieses Scan Resultat anhängen an den Beschreiber.
	170	[0:21:06.8] Jakob Möller: Alternativ kannst du natürlich auch hingehen und kannst sagen, du lässt den Beschreiber erstmal so und lieferst den aus und ein Konsument, der praktisch diese Komponenten dann auch wieder referenziert.
Konkrete Anforderungen	171	[0:21:18.7] Jakob Möller: Der hat dann eine Live Analyse zum Beispiel in seiner Umgebung, in der er beispielsweise die Software liefert und kann dann praktisch live auf diesem Komponentenbeschreiber Konvertierung ausführen und dann den Scan durchführen.
	172	[0:21:31.4] Jakob Möller: Beides hat Vor und Nachteile.

- Kritik & Gegenargumente
- 173 [0:21:33.2] Jakob Möller: Im ersten Moment hast du natürlich einen Scan gemacht, der direkt aus deinem CI System kommt, Der ist dann statisch, der ist dann ein Snapshot in Time, sage ich mal.
 - 174 [0:21:41.3] Jakob Möller: Das Problem ist, wenn im Nachhinein dann Scan Resultate anfallen, dann müsste man die nochmal dynamisch miteinander verlinken.
- Best Practices
- 175 [0:21:48.8] Jakob Möller: Die SBOM würde sich vielleicht nicht stark ändern, aber die Resultate, die dann daraus gescannt werden.
- Mapping-Kompatibilität
- 176 [0:21:55.2] Jakob Möller: Genau, es gibt schon die Möglichkeit im OCM über einen spezifizierten Artefaktyp einen SBOM anzuhängen, Also das geht schon heute, aber du verlierst natürlich damit die dynamische Scan Möglichkeit und du müsstest jedes Mal, wenn du den Komponentenbaum anpasst, natürlich auch das SBOM aktualisieren.
 - 177 [0:22:12.2] Jakob Möller: Das passiert dann nicht automatisch.
- Modellierungsstrategien
- 178 [0:22:14.4] Olson Sturm: Aber der Scope von meiner Arbeit ist ja eigentlich nicht genau, woher ich die SBOMs kriege, ob die jetzt als Typ angehängt sind in der Ressource in OCM oder ob die generiert werden von Syft oder von irgendeinem Open Source Scanner, sondern schlussendlich ist es ja eigentlich das Aggregieren und das Herstellen einer globalen Beschreibung von dem ganzen OCM in SBOM, richtig?
 - 179 [0:22:41.4] Jakob Möller: Ja, natürlich.
 - 180 [0:22:42.4] Jakob Möller: Also grundsätzlich, egal wo du den Scan durchführst, egal wo du das SBOM aggregierst, die Aggregationsproblematik bleibt natürlich.
- OCM-spezifische Strukturen
- 181 [0:22:49.7] Jakob Möller: Komponentenbeschreiber sind keine voll aufgelösten Bäume.
 - 182 [0:22:53.0] Jakob Möller: Das heißt, egal was du machst, du musst, wenn du einen kompletten Baum traversieren willst, immer eine komplette Auflösung möglich machen und das ändert sich nicht, egal ob du das zur Build Time oder zur Delivery Time auflöst.
 - 183 [0:23:07.9] Olson Sturm: OK, machen wir weiter mit der nächsten Frage.
 - 184 [0:23:10.0] Olson Sturm: Ich muss ganz kurz noch mal lesen, ob wir die nicht schon gleich beantwortet haben.
 - 185 [0:23:20.2] Olson Sturm: Okay.
 - 186 [0:23:21.0] Jakob Möller: Genau.
 - 187 [0:23:21.2] Olson Sturm: Jetzt noch einmal zu gewissen OCM typischen Metadaten würde ich sagen.

	188	[0:23:27.6] Olison Sturm: Also sowas wie Label, dann haben wir die extra Identity, wir haben, ich glaube, irgendwas mit Repository Context, richtig.
	189	[0:23:39.7] Olison Sturm: Wie würdest du das Mapping vollziehen?
	190	[0:23:43.4] Olison Sturm: Also hast du da Ideen, welche Felder da am besten in CycloneDX zur Option stehen?
Konkrete Anforderungen	191	[0:23:51.8] Jakob Möller: Ja, also ich hatte es ja schon ein paar Mal angesprochen, ich glaube, das semantisch korrekteste Mapping wäre das Mapping auf einer Komponentenidentität, die global eindeutig und rückreferenzierbar ist, weil die sich nie ändert.
Best Practices	192	[0:24:07.3] Jakob Möller: Felder wie Repository Kontexte sind eigentlich nur für den Delivery Aspekt gedacht und sind deswegen nicht stabil und sollten meiner Meinung nach deswegen auch nicht verwendet werden, in einen SBOM zurückzumappen.
	193	[0:24:20.8] Jakob Möller: Man kann diese verwenden, zusätzlichen Kontext zu stellen, aber die ändern sich nach jeder Delivery.
Mapping-Kompatibilität	194	[0:24:28.2] Jakob Möller: Also der Repository Kontext ändert sich natürlich nach jedem Transport in ein anderes Zielsystem zu den Ressourcen mit der extra Identity.
OCM-spezifische Strukturen	195	[0:24:37.2] Jakob Möller: Dadurch, dass wir dann voraussetzen, dass wir eine globale Komponentenidentität haben, ist jede Ressourcenidentität auch global, weil die Kombination aus Komponentenidentität plus Ressourcenidentität auch wieder eindeutig ist.
Konkrete Anforderungen	196	[0:24:49.9] Jakob Möller: Und das kann man dann wiederum nutzen, zusammen mit dem Access, der ja eigentlich dynamisch ist, eine SBOM Referenz aufzubauen, die statisch ist und die auch sich nie ändert, die also immutabel ist und gleichzeitig eine SBOM-Referenz auf einen Access.
	197	[0:25:05.5] Jakob Möller: Zu haben, die auch statisch ist.
	198	[0:25:07.2] Jakob Möller: Das heißt, wir können eigentlich einen SBOM generieren, was auch lokationsunabhängig immer funktioniert.
	199	[0:25:13.4] Olison Sturm: OK, wunderbar.
OCM-spezifische Strukturen	200	[0:25:17.1] Olison Sturm: Gibt es irgendwelche weiteren OCM Konventionen, die mir bekannt sein müssten, über die hinweg, worüber wir gerade gesprochen haben?
Modellierungsstrategien	201	[0:25:28.8] Jakob Möller: Ich glaube, das Wichtigste ist natürlich die OCM Kernspezifikation, die wir für unsere Laufzeit verwenden.
	202	[0:25:34.8] Jakob Möller: Über die haben wir bereits geredet.
Mapping-Kompatibilität	203	[0:25:37.2] Jakob Möller: Wenn wir Spezifikationen anlegen, dann machen wir das über eine Zusatzspezifikation, die nennt sich Konstruktorspezifikation.

		[0:25:45.0] Jakob Möller: Diese Konstruktorspezifikation ist eine Sonderform des Beschreibers, den wir schon angesprochen hatten, und die ist nur dann relevant, wenn wir neue Komponenten anlegen.
Mapping-Kompatibilität	204	[0:25:45.0] Jakob Möller: Diese Konstruktorspezifikation ist eine Sonderform des Beschreibers, den wir schon angesprochen hatten, und die ist nur dann relevant, wenn wir neue Komponenten anlegen.
OCM-spezifische Strukturen	205	[0:25:53.7] Jakob Möller: Und da wäre es beispielsweise wichtig, wie man eine SBOM in eine Komponente hineinlegt.
Modellierungsstrategien	206	[0:26:01.1] Jakob Möller: Aber wir hatten ja schon darüber geredet, deine Arbeit dreht sich vor allen Dingen die Aggregation.
	207	[0:26:05.4] Jakob Möller: Das heißt, du gehst ja von bereits fertig erstellten und transportierten Komponenten aus und damit ist das eher im Hintergrund und vielleicht gar nicht so relevant.
	208	[0:26:14.2] Olison Sturm: Okay, verstehe.
	209	[0:26:15.2] Olison Sturm: Danke.
Tool- und Technologieeinsatz	210	[0:26:18.7] Olison Sturm: Wir haben vorher schon einmal ganz kurz über Tools gesprochen.
	211	[0:26:22.3] Olison Sturm: Gibt es irgendwelche Tools, die aus OCM Sicht relevant werden oder gibt es nur die Richtlinie, dass es einfach Open Source Tools sein sollten, weil wir in Open Source Kontext arbeiten.
	212	[0:26:33.1] Jakob Möller: Also OCM ist natürlich ein Public Domain Projekt.
	213	[0:26:37.8] Jakob Möller: Das heißt, alles was wir in OCM verwenden können, sind auch nur Projekte, die auch in der Public Domain vorhanden sind und auch in der Public Domain verwendbar sein dürfen, abhängig von unseren License Restriktionen.
Konkrete Anforderungen	214	[0:26:51.6] Jakob Möller: Bei OCM ist das die Apache Lizenz.
	215	[0:26:54.3] Jakob Möller: Das bedeutet grundsätzlich sind alle Projekte theoretisch erstmal OK, die innerhalb von einer Apache Lizenz wiederverwendet werden können.
Best Practices	216	[0:27:02.9] Jakob Möller: Das heißt, wenn wir jetzt beispielsweise ein Hilfsframework aufbauen, dann sollte das nur mit Tools geschrieben sein, die auch unter Apache wieder weiterverbreitet werden können.
Qualitätskriterien	217	[0:27:15.7] Jakob Möller: Natürlich haben wir auch noch die Restriktion, dass wir als Unternehmen sicherstellen müssen, dass die Projekte, an denen wir mitarbeiten, über lange Zeit verwaltet werden können.
	218	[0:27:26.0] Jakob Möller: Das heißt, Sekundärkriterien für Auswahl von Abhängigkeiten sind Dinge wie beispielsweise die Maintainability, also wie oft kommen neue Features oder Fixes in die Abhängigkeiten rein und wie gefragt und wie populär sind die Tools natürlich auch im Open Source Kontext.
	219	[0:27:44.6] Jakob Möller: Es bringt niemandem was, wenn wir ein Tool

Qualitätskriterien		repräsentieren oder integrieren mit etwas, was eh niemand verwendet.
Best Practices		[0:27:50.5] Jakob Möller: Also da sind wir natürlich interessiert daran, möglichst in die Ökosysteme, die schon da sind, zu integrieren.
Konkrete Anforderungen	220	[0:27:57.3] Olison Sturm: Okay, danke.
	221	[0:27:58.5] Olison Sturm: Gibt es noch weitere nicht funktionale Anforderungen, die mein Prototyp vor allem erfüllen sollte?
Konkrete Anforderungen	222	[0:28:07.9] Jakob Möller: Ich glaube, das Wichtigste ist, dass wir zeigen können, dass die SBOMs, die wir generieren, komplett zurückführbar sind.
	223	[0:28:14.6] Jakob Möller: Das habe ich ja bereits gesagt.
Qualitätskriterien	224	[0:28:16.3] Jakob Möller: Diese Zurückführbarkeit sollte idealerweise in einer Art und Weise stattfinden, in der wir das sowohl als Code Abhängigkeit als auch als Tooling, beispielsweise in Form eines Command Line Interfaces anbieten können.
Konkrete Anforderungen	225	[0:28:29.7] Jakob Möller: Das heißt, es muss zeigen, dass wir das sozusagen als kleine Library verwenden können, im Endeffekt diese Konvertierung durchzuführen.
	226	[0:28:37.7] Jakob Möller: Wie wir dann diese konvertierten SBOMs wieder injecten, das ist für mich nicht Teil des Prototypen, weil das für mich etwas ist, was wir bereits mit etabliertem Tooling lösen können.
Konkrete Anforderungen	227	[0:28:46.7] Jakob Möller: Also für mich geht es praktisch nur die tatsächliche Generierung, Abtraversierung des SBOM Graphen.
	228	[0:28:54.7] Olison Sturm: Verstehe, da sind wir schon am Ende von meinen vorbereiteten Fragen.
	229	[0:29:00.9] Olison Sturm: Ich würde dich jetzt einfach noch gerne fragen, wie siehst du es denn in Zukunft?
	230	[0:29:04.1] Olison Sturm: Wird es vielleicht irgendwann in Zukunft einheitliches SBOM Format geben oder irgendwas Standard der sich durchsetzt oder werden noch viel mehr auf den Markt kommen als die zwei größten bekanntesten SPDX.
Zukunftsperspektiven	231	[0:29:14.7] Olison Sturm: Und CycloneDX.
	232	[0:29:16.3] Olison Sturm: Wie siehst du das da?
	233	[0:29:17.9] Jakob Möller: Also meine Sichtweise auf die Dinge ist, dass ich glaube, dass wir einen sehr etablierten Markt haben, was das angeht.
Best Practices	234	[0:29:24.4] Jakob Möller: Und ich persönlich glaube nicht, dass wir ohne große Markt neue Frameworks sehen werden, die auch in großer Weite angefangen werden, von allen verwendet zu werden.

	236	[0:29:35.8] Jakob Möller: Das heißt, wir haben jetzt schon den Zustand, dass Menschen oft Probleme damit haben, dass wir zwei etablierte Standards haben mit CycloneDX und mit SPDX.
	237	[0:29:44.2] Jakob Möller: Und ich glaube persönlich, dass wir wahrscheinlich eher sehen werden, dass viel mehr Tools die etablierten zwei Frameworks, die da sind, nutzen werden, genauso wie wir das ja auch machen wollen.
CycloneDX-spezifische Struktur	238	[0:29:55.3] Jakob Möller: Unser Ziel ist es ja auch nicht CycloneDX oder SPDX zu ersetzen, sondern unser Ziel ist es ja nur zu ergänzen und zu integrieren.
	239	[0:30:03.7] Jakob Möller: Und ich glaube, dass viele Schnittstellen in diesen Spezifikationen auch bereits gegeben sind.
	240	[0:30:09.4] Jakob Möller: Also gerade CycloneDX bietet ja explizit Erweiterungsmöglichkeiten in ihren Spezifikationen, über die das auch abbildbar ist.
Zukunftsperspektiven Best Practices	241	[0:30:17.7] Olison Sturm: Was hältst du davon, wenn wir gar kein SBOM Format brauchen würden, Dadurch, dass vielleicht in Zukunft Scanner einfach direkt die OCM Beschreibung, die Beschreiber unterstützen würden, Hältst du das für eine Option, die man erreichen kann?
	242	[0:30:33.1] Jakob Möller: Ich halte das für extrem unrealistisch, einfach aus dem Grund, weil Scannerentwickler sich auf einen Implementierungshintergrund beschränken müssen, der wohl definiert ist.
	243	[0:30:44.5] Jakob Möller: Das bedeutet, wir können ja nicht jedes Mal, wenn wir ein neues Framework von irgendwoher haben, alle Scanner Anbieter anfragen, ein neues Framework auch in ihren Scannern zu unterstützen.
	244	[0:30:55.5] Jakob Möller: Das heißt, es muss einen Standard geben, beispielsweise jetzt beim Vulnerability Scanning, auf den man sich industrieweit zumindest de facto einigt.
	245	[0:31:03.3] Jakob Möller: Das führt dann automatisch dazu, dass die Anforderung eher ist, dass wenn man sich eine Ergänzung überlegt, dann müsste mit etablierten Frameworks kompatibel sein und darf nicht yet another Standard sein.
	246	[0:31:15.8] Olison Sturm: Verstehe.
	247	[0:31:18.4] Olison Sturm: Kennst du schon CycloneDX, was da der Plan dahinter ist?
	248	[0:31:23.7] Jakob Möller: Du kannst mir gerne erklären.
	249	[0:31:25.0] Olison Sturm: Ja genau.
	250	[0:31:25.8] Olison Sturm: Also CycloneDX will weg von dem ganzen Bill of Materials, also unterstützen ja ganz viele Bill of Materials, nicht nur Software,

	Bill of Material.
251	[0:31:35.3] Olison Sturm: Und in Zukunft soll CycloneDX, die ersten Steps werden schon gemacht in 1.7, das Ende Dezember wahrscheinlich released werden soll in Zukunft zu einem Transparency Standard werden, also viel offener für fast alles nutzbar.
252	[0:31:52.5] Olison Sturm: Das ist so die Vision von CycloneDX in der Zukunft.
253	[0:31:57.7] Jakob Möller: Also meine Sichtweise auf das Ganze ist, dass ich nicht glaube, dass das verwendet werden wird, zumindest gegen Anfang.
254	[0:32:03.9] Jakob Möller: CycloneDX ist am Anfang durch Tools repräsentiert worden, die CycloneDX verwendet haben und weil sie CycloneDX verwendet haben, ist CycloneDX auch größer geworden.
255	[0:32:14.7] Jakob Möller: Und ich glaube, wenn es keine Frameworks gibt und keine Tools, die CycloneDX in diesem Transparenzformat auch verwenden wollen und es auch keine Frameworks gibt, die CycloneDX Transparenzstandard nutzen wollen, dann wird es auch keine große Adoption finden.
256	[0:32:29.1] Jakob Möller: Das ist aber nur meine Sichtweise aus dem aktuellen Standpunkt.
257	[0:32:33.3] Jakob Möller: Also da müsste man jetzt auch mal abwarten, wie so die Community reagiert, bevor man dann sagen kann, ja, okay, das macht ja total Sinn.
258	[0:32:41.0] Jakob Möller: Was ich sagen kann, ist, dass auch OCM schon interessiert daran ist, zu überlegen, können wir SBOM als Standard Speicherformat beispielsweise verwenden und die OCM Beschreiber nur als, ich sag mal, Laufzeitmodell nutzen und immer wenn wir persistieren und immer wenn wir mit Speicherformaten umgehen, dann automatisch umkantieren in beispielsweise einen CycloneDXSBOM.
259	[0:33:05.9] Jakob Möller: Wenn das jetzt wegfallen würde, also wenn wir jetzt sagen würden, es gibt kein SBOM mehr, was vorgegeben ist und es gibt ein Transparenzformat, dann frage ich mich natürlich, wo ist dann noch der Mehrwert für Integratoren, wenn wir kein Einheitsformat mehr haben, auf das wir uns einigen können, vielleicht, weil es eine Library dafür gibt, dann die man verwenden kann?
260	[0:33:26.2] Jakob Möller: Ich weiß es nicht.
261	[0:33:27.2] Olison Sturm: Denkst du, dass CycloneDX einfach immer ein paar Steps weit voraus ist und alle ein bisschen hinterherhinken, alle Spezifikationen irgendwie zu verstehen und eher so langsam mitrücken und nicht alle Funktionen wirklich richtig genutzt werden oder eher nicht?
262	[0:33:46.4] Jakob Möller: Also für mich ist CycloneDX in einer ganz schwierigen Position, weil sie natürlich ein Konsortium sind und auch

		getrieben von verschiedenen Firmen und Interessenhaltern.
	263	[0:33:55.8] Jakob Möller: Also würde ich das vielleicht mit der Politik vergleichen.
	264	[0:33:58.2] Jakob Möller: Natürlich versucht man abzuschätzen, was jetzt die nächste große Entwicklung ist und wie man die Firmen dabei unterstützen kann, ich sag mal, Software Transparenz weiterzutreiben und weiterzubringen.
	265	[0:34:12.1] Jakob Möller: Gleichzeitig glaube ich aber, dass das automatisch dazu führt, dass man in manchen Gebieten einfach zu viel investiert und in manchen zu wenig, weil man arbeitet ja immer automatisch.
Zukunftsperspektiven Best Practices Expertise & Kontext	266	[0:34:19.9] Jakob Möller: Das ist ja der Auftrag von so einer Entwicklungsgruppe, so ein bisschen vorauszuschauen.
	267	[0:34:24.7] Jakob Möller: Ich persönlich glaube, das ist aber okay, weil am Ende des Tages gibt es ja immer so einen Mittelpunkt aus de facto und de jure.
	268	[0:34:32.7] Jakob Möller: Das bedeutet, wenn die CycloneDX Spezifikation da was Neues vorgibt und es stellt sich raus, das wird von niemandem adopted, dann wird vermutlich diese De jure Spezifikation irgendwann aufgegeben und wird ersetzt durch etwas, was mit den Learnings kombiniert wird, die aus der De facto Implementierung kommt.
	269	[0:34:48.6] Jakob Möller: Also ich persönlich sehe da jetzt keine Schwierigkeiten in der Adoption oder dass man zu weit voraus ist.
	270	[0:34:54.7] Olison Sturm: Okay, danke.
	271	[0:34:57.8] Olison Sturm: Das wäre es tatsächlich auch schon.
	272	[0:34:59.5] Olison Sturm: Gibt es noch irgendwelche Ergänzungen, irgendwas, was wir nicht angesprochen haben oder eine unverzichtbare Anforderung für mich.
	273	[0:35:08.4] Olison Sturm: Alles klar, dann danke ich dir für die Zeit und das war es dann auch schon mit unserem Gespräch.
	274	[0:35:13.2] Olison Sturm: Ich habe Gedanken genau, was passiert ist jetzt am Ende, dass ich einfach nach Mayring eine Inhaltsanalyse mache mit den vergleichbaren Fragen, schlussendlich auch noch mit einer Dokumentanalyse die Spezifikation einfließen zu lassen, meine funktionalen und nicht funktionalen Anforderungen zu stellen.
	275	[0:35:31.4] Olison Sturm: In meiner Arbeit.
	276	[0:35:33.2] Olison Sturm: Und natürlich wirst du am Ende die Arbeit sehen, ist ja klar.
	277	[0:35:36.9] Olison Sturm: Und auch das Transkript dann in der Arbeit.

- 278 [0:35:40.3] Olison Sturm: Genau.
- 279 [0:35:41.2] Jakob Möller: Ansonsten danke für die Teilnahme, danke fürs Organisieren.
- 280 [0:35:45.2] Jakob Möller: Tschüssi.
- 281 [0:35:46.2] Olison Sturm: Ciao.

Anhang C: Experteninterview mit Steve Springett

- 1 [0:00:00.0]
2 [0:00:07.3] Olison Sturm: So usually it should start a recording and we should hear it...
3 [0:00:22.2] Steve Springett: Taking notes right now.
4 [0:00:25.7] Olison Sturm: It does?
5 [0:00:26.0] Steve Springett: You know what, this is probably my meeting.
6 [0:00:30.9] Steve Springett: Was this my meeting?
7 [0:00:33.1] Olison Sturm: Yeah, it's.
8 [0:00:33.7] Olison Sturm: It's your meeting.
9 [0:00:34.8] Steve Springett: Okay.
10 [0:00:47.3] Olison Sturm: Because the recording is not on I thought.
11 [0:00:52.6] Steve Springett: Did you want me to record?
12 [0:00:54.3] Olison Sturm: Yeah, would be perfect.
13 [0:01:02.1] Steve Springett: Okay, I think it's recording now.
14 [0:01:05.0] Olison Sturm: Okay, perfect.
15 [0:01:11.7] Olison Sturm: Yeah, I think it will work hopefully. Afterwards you could send me the recording if you have it on your.
16 [0:01:24.9] Olison Sturm: Okay, perfect.
17 [0:01:25.7] Olison Sturm: Thanks.
18 [0:01:26.8] Olison Sturm: So I wanna start with a brief explanation about my topic.
19 [0:01:34.7] Olison Sturm: So I'm investigating the transformation strategy for converting to complete OCM component tree to one standardized SBOM..
20 [0:01:50.0] Olison Sturm: So in this context CycloneDX because I'm focusing completely on CycloneDX because it was easier to understand for me in the beginning and I had a lot of struggle because I read the whole specification of CycloneDX and I started with SPDX but it was so hard so I thought it might better to focus one SBOM format.
21 [0:02:14.1] Olison Sturm: So I did it.
22 [0:02:16.3] Olison Sturm: So yeah, the goal for this interview is to basically identify the technical requirements for transferring for transforming OCM components to CycloneDX.

- 23 [0:02:32.9] Olison Sturm: So are you like kind a bit familiar with the OCM, the open component model?
- 24 [0:02:39.3] Steve Springett: I am not, no.
- 25 [0:02:40.7] Olison Sturm: Okay, so yeah, I've been working with it for a month now but generally I'm working for SAP since three years while my whole studies and the OCM got donated from SAP to the NeoNephos Foundation.
- 26 [0:03:03.3] Olison Sturm: It's part of the Linux Foundation I think and maybe I can give you a short introduction and share my screen to show you like the structure a bit of OCM so maybe you can answer the questions later on a bit better.
- 27 [0:03:22.4] Olison Sturm: I'll share my screen.
- 28 [0:03:31.6] Olison Sturm: So first of all I want to show you our website so I think it explains it the best.
- 29 [0:03:42.7] Olison Sturm: We call it at SAP the Software Bill of Delivery.
- 30 [0:03:45.8] Olison Sturm: So it's made for packaging all the artifacts and signing them with digits and then deploying the artifacts in another system or anywhere else.
- 31 [0:03:56.9] Olison Sturm: Or for example if you have all the artifacts in one registry, you can upload it somewhere else in other registry that's not like something like that.
- 32 [0:04:06.2] Olison Sturm: So but I'm not that deep into it because I'm working with it for a month so I'm struggling a bit still.
- 33 [0:04:12.4] Olison Sturm: But yeah and I could show you the structure.
- 34 [0:04:18.8] Olison Sturm: So to get started with the open component model we have a component constructor Is it called and you specify in a YAML file all your components.
- 35 [0:04:30.3] Olison Sturm: So a component can have first of all the name and it should be have like a namespace like a domain at the beginning and also like that it is unique and then you have resources and resources are basically OCI images, helm charts, blobs and so on.
- 36 [0:04:49.3] Olison Sturm: There are many different types.
- 37 [0:04:51.1] Olison Sturm: I focused mostly on OCI images by now, but in the future it should support everything hopefully.
- 38 [0:04:59.3] Olison Sturm: And yeah, you can have multiple components and for example you can have components.
- 39 [0:05:06.3] Olison Sturm: I will show you in my IDE.

- 40 [0:05:09.7] Olison Sturm: This is one component constructor that I'm using for my proof of concept.
- 41 [0:05:14.4] Olison Sturm: And we have here a parent component.
- 42 [0:05:17.5] Olison Sturm: I named it like that because it's easier to understand a parent OCM component and we have component references so we have kind of OCM component tree with like there's no end to be honest.
- 43 [0:05:31.4] Olison Sturm: If you every component could have like another child component.
- 44 [0:05:37.0] Olison Sturm: And so the main thing are the resources and there you can have like for example an OCI image or like many different other types as well as SBOMs and so on.
- 45 [0:05:52.4] Olison Sturm: And if I have like for example child one as a child of the parent component, this one can have as well multiple resources and as well sources.
- 46 [0:06:07.4] Olison Sturm: That's another attribute would say.
- 47 [0:06:11.5] Olison Sturm: And sources are basically the source code for the binaries that we have here and yeah, and so on.
- 48 [0:06:20.0] Olison Sturm: And so you have your whole tree and many different fields like there as well the access type, OCI artifacts.
- 49 [0:06:28.8] Olison Sturm: But you could have also an input type like a local type for example UTF 8 or as a plain text basically and many other input types.
- 50 [0:06:42.5] Olison Sturm: Maybe I can show you now I can't but yeah, that's.
- 51 [0:06:48.1] Olison Sturm: I don't know if you understand what OCM is doing.
- 52 [0:06:52.5] Olison Sturm: Yeah, it makes sense.
- 53 [0:06:53.8] Steve Springett: Okay, perfect.
- 54 [0:06:57.6] Olison Sturm: So I will stop sharing my screen and we can go to questions.
- 55 [0:07:04.9] Olison Sturm: So yeah, I have a research question and I will like read it out loud.
- 56 [0:07:11.5] Olison Sturm: How can the information structure of the open component model be converted into the CycloneDX SBOM format with minimal information loss and to enable compatibility and
- 57 [0:07:25.5] Olison Sturm: interoperability with established software supply chain tools.

- 58 [0:07:32.1] Olison Sturm: Yeah, so I have.
- 59 [0:07:35.4] Olison Sturm: My first questions are like background questions about your expertise context.
- 60 [0:07:42.2] Olison Sturm: So I saw a lot of videos online of you and I know you are the creator of Dependency Track.
- 61 [0:07:49.4] Olison Sturm: Right.
- 62 [0:07:50.7] Olison Sturm: And but it would be really helpful if you would tell me like how long you're working in this field.
- 63 [0:07:59.1] Olison Sturm: What's your actual focus.
- 64 [0:08:01.1] Steve Springett: Yeah, so 30 plus years as a software developer, 20 of those years in software security software supply chain since 2012 and physical supply chain since 2008.
- 65 [0:08:19.1] Olison Sturm: Okay.
- 66 [0:08:21.3] Olison Sturm: And what's your current role or like your responsibility in this field especially CycloneDX.
- 67 [0:08:30.7] Steve Springett: I am for Cyclone.
- 68 [0:08:33.7] Steve Springett: I'm the original author and currently serve as the chair of the core working group and the chair of ECMA TC54, which is the group that's standardizes the specification.
- 69 [0:08:51.5] Olison Sturm: That's the group that gave the minimal elements that the SBOM should have, right?
- 70 [0:08:59.7] Steve Springett: No, no, ECMA.
- 71 [0:09:01.8] Steve Springett: Yeah, ECMA is the, is an international standards organization.
- 72 [0:09:07.1] Steve Springett: So like JavaScript for example is.
- 73 [0:09:10.4] Olison Sturm: Yeah, thanks.
- 74 [0:09:11.4] Olison Sturm: Okay, so the first question that's now about OCM, in what scenarios do you see the need to transform OCM like structures into a standardized SBOM format?
- 75 [0:09:28.3] Olison Sturm: So CycloneDX.
- 76 [0:09:30.0] Steve Springett: Yeah, the need...I think just interoperability is likely the biggest motivator.
- 77 [0:09:46.7] Steve Springett: I mean we have hundreds of tools in the CycloneDX tool center and we know that there are thousands of tools that

Expertise & Kontext

- Expertise & Kontext
- 78 [0:09:57.3] Steve Springett: So it's, you know, in order to leverage this rich ecosystem of tools, it's, you know, I think there's going to be a need to convert it to that to take advantage of that.
 - 79 [0:10:13.7] Olison Sturm: Yeah.
 - 80 [0:10:15.6] Olison Sturm: And which real world use case do you foresee for SBOMs generated from OCM components?
 - 81 [0:10:26.3] Olison Sturm: So for example, policy enforcements or vulnerability scanning and so on?
 - 82 [0:10:35.0] Steve Springett: Honestly, in any of the use cases supported by OCM.
- Expertise & Kontext
- 83 [0:10:41.1] Steve Springett: CycloneDX originally started out as a full stack bill of material specification and over the last couple of years, and especially going into next year has transformed into a transparency expression language.
- Zukunftsperspektiven
- 84 [0:11:00.5] Steve Springett: And for example, the majority of features that went into CycloneDX 1.7 and pretty much all of the features going into CycloneDX 2.0, which will be next year, have nothing to do with bills and materials.
 - 85 [0:11:18.7] Steve Springett: It's all about transparency, being able to express end to end from the moment that software is conceived as an idea to the point that it's actually destroyed throughout its entire life cycle is essentially what CycloneDX can do.
 - 86 [0:11:44.8] Steve Springett: And we're moving more and more into the direction of less bill of material type use cases and more just holistic transparency.
 - 87 [0:11:57.6] Olison Sturm: Okay.
 - 88 [0:11:58.8] Olison Sturm: And yeah, amazing.
- Tool- und Technologieeinsatz
- 89 [0:12:02.9] Olison Sturm: So I would start now with the core questions to get my technical requirements and the first question would be what tools or technologies do you know, or use for generating, merging or transforming SBOMs, I mean like CycloneDX SBOMs in this case.
 - 90 [0:12:27.1] Steve Springett: Yeah, we've got a couple within the CycloneDX community.
 - 91 [0:12:31.5] Steve Springett: There's the CycloneDX CLI and then there's the web version of the CLI which is the CycloneDX web tool that actually uses the exact same code but runs in the browser.
 - 92 [0:12:49.0] Steve Springett: It's based on WASM, so it runs really fast.

	Tool- und Technologieeinsatz		
93		[0:12:55.7] Steve Springett: So no JavaScript.	
94		[0:12:57.1] Olison Sturm: Yeah, it's amazing.	
95		[0:12:58.1] Olison Sturm: I looked into the code and what I did is like building almost the same emerging tool that CycloneDX CLI has.	
96		[0:13:06.7] Olison Sturm: I built it in Go.	
97		[0:13:08.3] Olison Sturm: Go language.	
98		[0:13:09.1] Olison Sturm: Yeah.	
99		[0:13:09.7] Steve Springett: And because I need to be in Go because OCM and all these scanners that I'm using by now for the proof of concept is in Go.	
100		[0:13:18.6] Olison Sturm: So there wasn't really a way except using the CLI and like execute the command from Go.	
101		[0:13:25.3] Olison Sturm: But I did it like that.	
102	..Middleware (Protobom)	[0:13:28.2] Steve Springett: So yeah, there's also the SBOM utility within the CycloneDX community that was originally contributed by IBM and outside of CycloneDX the only thing that I really know of is protobuf, but honestly it doesn't do that great of a job.	
103	..Herausforderungen	[0:13:47.1] Olison Sturm: Okay, that's another question later on.	
104	Tool- und Technologieeinsatz	[0:13:49.1] Olison Sturm: Yeah, yeah, that's what I wanted to ask.	
105		[0:13:51.8] Olison Sturm: Okay, perfect, thanks.	
106	Modellierungsstrategien	[0:13:53.4] Olison Sturm: And which CycloneDX structures are most critical for ensuring downstream like tools?	
107		[0:14:03.9] Steve Springett: It really depends on the use case.	
108	Modellierungsstrategien	[0:14:11.3] Steve Springett: The basic idea for Cyclone was that weren't trying to create the perfect model, were trying to create something that could be easily automatable.	
109		[0:14:24.7] Steve Springett: And there are very few primitives in CycloneDX and those minimum amount of primitives can be used for a wide range of use cases.	
110	Best Practices	[0:14:39.6] Steve Springett: So if you're in the hardware business.	
111		[0:14:43.0] Steve Springett: Right.	
112	CycloneDX-spezifische Struktur	[0:14:43.6] Steve Springett: Using the CycloneDX component object, AI again, and you know, SBOM and stuff.	
	Modellierungsstrategien		

	113	[0:14:56.0] Steve Springett: Component objects.
Kritik & Gegenargumente	114	[0:14:57.6] Steve Springett: If you're into services, we do have a services object as well and there are many commercial and open source tools that support that today.
	115	[0:15:09.8] Steve Springett: I would say at a very minimum the metadata section of the CycloneDX specification components and services.
Best Practices	116	[0:15:22.0] Steve Springett: Some of the advanced use cases, most of the tools don't support for example the pedigree part of a component.
CycloneDX-spezifische Struk	117	[0:15:31.8] Steve Springett: For example.
Modellierungsstrategien	118	[0:15:33.3] Steve Springett: Very few tools support that because it's, it can't be automated today.
	119	[0:15:41.4] Steve Springett: So maybe with component without that and other than that, maybe just a dependency and the external references.
	120	[0:15:52.5] Olson Sturm: Yeah.
	121	[0:15:52.8] Steve Springett: So if you have those primitives and you should realistically be able to model pretty much 95% of what people want to model.
	122	[0:16:03.2] Olson Sturm: Okay, yeah, thanks.
	123	[0:16:10.4] Olson Sturm: Which elements or structures from the OCM?
	124	[0:16:14.1] Olson Sturm: So we saw like a kind of the OCM would be in your experience or that can be directly mapped without any data loss to CycloneDX.
..Verlustfreie Konvertierung	125	[0:16:27.8] Steve Springett: I think pretty much all of them, I think I, I, I haven't seen anything in OCM that would actually.
Mapping-Kompatibilität	126	[0:16:37.4] Steve Springett: Let me, let me go take a look at it but I didn't see anything in OCM that would actually result in data loss from OCM to cycle.
	127	[0:16:48.6] Olson Sturm: So you mean there's always like a field or it could be always mapped to any of.
	128	[0:16:54.8] Olson Sturm: In a CycloneDX field.
	129	[0:16:56.4] Steve Springett: Yeah, either natively.
	130	[0:16:59.3] Steve Springett: Either natively, right.
Best Practices	131	[0:17:00.8] Steve Springett: Or if the field just doesn't exist, you can use CycloneDX properties which are value pairs.
CycloneDX-spezifische Struktur	132	[0:17:09.5] Olson Sturm: Yeah, I heard of it and I, I did an issue opened an issue for a namespace for OCM for the properties already because we have a

- lot of like in OCM we have extra identities and another like operation system and so on and to have the interoperability so to map backwards if we map to OCM we need some.
- 133 [0:17:37.6] Olison Sturm: Yeah, some informations and I didn't find like any field that are like the same in CycloneDX.
- 134 [0:17:46.6] Olison Sturm: So for example the external identity and so on, I put it in Properties.
- 135 [0:17:52.3] Steve Springett: So that's the way describe external identity.
- 136 [0:17:56.4] Olison Sturm: That's hard to be honest always if I didn't really got it I used the extra identity not the extra identity properties in CycloneDXs.
- 137 [0:18:10.0] Olison Sturm: So yeah, I could look it up, search it up if you want.
- 138 [0:18:16.0] Steve Springett: Yeah, I mean it might be covered with the CycloneDX evidence.
- 139 [0:18:25.9] Olison Sturm: What do you mean by that?
- 140 [0:18:27.4] Steve Springett: Well, a CycloneDX has this concept so a component, when you actually build the software it only has a single identity.
- 141 [0:18:41.1] Steve Springett: Now it can be referred to as a couple other things in the real world but when you actually build the software it's a single thing.
- 142 [0:18:56.0] Steve Springett: What we have is a way using evidence in Cyclone to substantiate the methods and techniques used to derive the identity of a component and your confidence of each one of those methods and techniques.
- 143 [0:19:19.7] Steve Springett: There are ways to have lower confidence identities and higher confidence identities based on the level of confidence of you know, what you did tooling wise.
- 144 [0:19:34.8] Olison Sturm: Okay, get it.
- 145 [0:19:35.8] Olison Sturm: So in OCM, for example, external identities are.
- 146 [0:19:44.1] Olison Sturm: You can freely specify them like you could like in you have the extra identity and then you can like give every name as a key and then every string as a value.
- 147 [0:19:54.7] Olison Sturm: So the most use cases, for example the arch, so the architecture, ARM64 or OS Linux.
- 148 [0:20:06.0] Olison Sturm: So such things are extra identities.
- 149 [0:20:10.6] Olison Sturm: Yeah and as well we have labels that can be specified completely free with the name, value pair and such things I would say because they're not real like you can.

CycloneDX-spezifische Strukturen

OCM-spezifische Strukturen
Mapping-Kompatibilität

		[0:20:24.6] Olson Sturm: Everyone can specify them on their own if they're using OCM would be like probably mapped into properties.
	150	[0:20:30.8] Steve Springett: Yeah, that's right.
Mapping-Kompatibilität	151	[0:20:32.6] Olson Sturm: Okay, let's move on which OCM concept are difficult or impossible to represent in CycloneDX.
	152	[0:20:45.2] Olson Sturm: I would say, like, especially I want to know about the hierarchical structure that we have in the OCM.
	153	[0:20:55.4] Olson Sturm: Because if we really want to have one SBOM that describes the whole OCM component tree, so many different OCM components.
	154	[0:21:06.7] Olson Sturm: How would.
	155	[0:21:08.7] Olson Sturm: Would you say, how would be their approach there?
Best Practices	156	[0:21:11.5] Steve Springett: Yeah, it's fairly easy.
	157	[0:21:14.3] Steve Springett: You know, Cyclone has nested components, and so components can include other components.
	158	[0:21:23.1] Steve Springett: And then using external references, we also can reference the BOM of a component via BOM link.
Mapping-Kompatibilität	159	[0:21:33.5] Olson Sturm: Right, with the serial id.
	160	[0:21:38.4] Steve Springett: Yes, yes.
CycloneDX-spezifische Struk	161	[0:21:40.8] Steve Springett: And in doing so, you can then specify like, well, this is the BOM, and then specify the lifecycle for operations.
..Hierarchische Strukturen	162	[0:21:50.9] Steve Springett: Okay, so like, this is your hardware, this is your operating system.
	163	[0:21:55.7] Steve Springett: Maybe you have another BOM for your configuration.
	164	[0:21:58.3] Steve Springett: Right, because that changes all the time.
	165	[0:22:01.1] Steve Springett: BOM for your application and your application libraries, and they all can kind of link to one another.
Modellierungsstrategien	166	[0:22:07.7] Olson Sturm: Okay, so are those two different approaches using external identities and nested components, or would you say you would use both of them within one whole SBOM?
	167	[0:22:21.6] Steve Springett: You could use both.
	168	[0:22:22.9] Steve Springett: Right?
	169	

	170	[0:22:23.3] Steve Springett: I mean, it's very common, especially for like enterprise software where you might have the marketing ver.
Best Practices	171	[0:22:32.6] Steve Springett: The marketing label of a particular thing.
Mapping-Kompatibilität	172	[0:22:35.9] Steve Springett: And then the reality is that particular enterprise solution incorporates like five or six different, like actual pieces of software or in the case of like an MRI machine.
Modellierungsstrategien	173	[0:22:48.8] Steve Springett: Right, where you've got a physical piece of hardware, but you got three different software stacks.
	174	[0:22:54.1] Olson Sturm: Right, okay, yeah, I understand.
	175	[0:22:57.4] Olson Sturm: Okay, so that's what I've thought as well, but it's nice to hear it from you as well, that I am on the right way.
	176	[0:23:06.5] Olson Sturm: So the next question, we already answered this by now, from your perspective, what are essential quality criterias that SBOM generated from OCM but or in general, an SBOM should fulfill.
	177	[0:23:28.8] Steve Springett: Obviously the minimum elements, you know, as a starting point.
	178	[0:23:35.4] Steve Springett: But I'm a big fan of using evidence to substantiate how you came up with the component identity in the first place.
Qualitätskriterien	179	[0:23:47.7] Steve Springett: So I would definitely recommend that there's a section of the CycloneDX Authoritative guide to SBOM that actually has a quality spider graph and the methods and techniques and the confidence for those techniques.
	180	[0:24:07.0] Steve Springett: It's all part of the quality.
Qualitätskriterien	181	[0:24:09.4] Steve Springett: But the other two aspects of the quality are derived from SCVS, which is the software component verification standard.
Best Practices	182	[0:24:20.4] Steve Springett: They have a BOM maturity model, and the BOM maturity model describes essentially any field that could theoretically be represented in a bom.
	183	[0:24:32.0] Steve Springett: And, and the level of difficulty it is to actually do that based on the availability of tools or if human intervention is actually required.
	184	[0:24:42.9] Steve Springett: There's, there's a rating system and so those five things kind of make up what we believe is quality.
	185	[0:24:54.8] Steve Springett: Use that as a guide.
	186	[0:24:57.5] Steve Springett: However, also using SCVS, different stakeholders can create their own profiles.

	187	[0:25:12.4] Steve Springett: So like a person that is maybe in the legal team that does open source licensing, he might have a profile and another person in legal that does M and A he will might have a profile for both open source licensing and patents.
Qualitätskriterien	188	[0:25:30.2] Steve Springett: You know, your application security folks might have a profile.
Best Practices	189	[0:25:33.8] Steve Springett: Your, your infrastructure security folks might have a profile, et cetera.
	190	[0:25:39.2] Steve Springett: And it really is the concept of putting weight against the types of data that you actually care about for the type of analysis that you want to perform form.
	191	[0:25:51.8] Steve Springett: So quality is really in the eyes of the beholder.
	192	[0:25:55.2] Olison Sturm: Right.
	193	[0:25:55.5] Steve Springett: It really depends on what you want to do with it.
	194	[0:25:58.1] Olison Sturm: Yeah, that's the whole problem.
	195	[0:26:00.1] Olison Sturm: A bit of my research, to be honest, because I, I've never spoke to someone that is using really OCM and want to have a whole SBOM for the whole OCM.
	196	[0:26:15.0] Olison Sturm: So it's kind of an idea of the OCM team and that's why I joined the team and said yeah, of course I will do research there and like figure it out because I'm really interested in the software security topics and stuff.
	197	[0:26:31.2] Olison Sturm: Yeah, okay.
Modellierungsstrategien	198	[0:26:32.7] Olison Sturm: And I don't know if I showed it to you, but OCM also signs the artifacts, I would say and also the whole OCM component with digits.
	199	[0:26:47.0] Olison Sturm: Would you say there's any way to transform them directly into OCM?
	200	[0:26:51.4] Olison Sturm: Because I think you have also like a way to sign in CycloneDX, right?
Best Practices	201	[0:26:56.2] Steve Springett: We do, yeah, we support envelope signatures in Cyclone.
CycloneDX-spezifische Strukturen	202	[0:27:01.8] Steve Springett: There likely is not a way to you know, convert the signing from one serialization format to another because when you do that the signature is going to be invalid for that particular blob.
Kritik & Gegenargumente		
..Konzeptuelle Inkompatibilität		
	203	[0:27:14.8] Olison Sturm: Yeah, get it.

	204	[0:27:16.2] Olison Sturm: So you have to do it again with the.
	205	[0:27:18.7] Steve Springett: Yeah.
	206	[0:27:22.3] Olison Sturm: Okay, let's move on to the next question.
	207	[0:27:25.6] Olison Sturm: What are potential risks or limitations of pursuing a direct OCM to CycloneDX mapping?
Best Practices	208	[0:27:34.4] Olison Sturm: Would you say it's not even practical.
	209	[0:27:37.6] Olison Sturm: Practical.
	210	[0:27:38.1] Olison Sturm: And you should.
	211	[0:27:39.1] Olison Sturm: We shouldn't follow this approach and like.
	212	[0:27:49.1] Steve Springett: No, I think it's a viable path forward.
	213	[0:28:00.2] Steve Springett: I don't know.
..Verlustfreie Konvertierung Mapping-Kompatibilität	214	[0:28:00.8] Steve Springett: I think just getting the taxonomy right for the things that can't be represented in CycloneDX natively.
	215	[0:28:09.9] Steve Springett: Yeah, that which I think you said that you are trying to register.
	216	[0:28:14.8] Olison Sturm: Yeah, I tried but it's Two weeks ago and I never got a master.
	217	[0:28:18.1] Olison Sturm: Oh really?
	218	[0:28:18.5] Steve Springett: Okay.
	219	[0:28:19.5] [0:28:20.4] Steve Springett: Jan is been.
	220	Steve Springett: Jan is my other co lead or one of the other co leads for Cyclone.
	221	[0:28:25.0] Steve Springett: He's been on vacation so apologies.
	222	[0:28:27.3] Olison Sturm: No worries.
	223	[0:28:30.8] Steve Springett: Haha. I think he's coming back.
	224	[0:28:32.0] Steve Springett: I think he's coming back this week though.
	225	[0:28:34.0] [0:28:37.2] Olison Sturm: Okay.
Modellierungsstrategien Best Practices	226	Steve Springett: So getting that right and just making sure that there's consistency between the tools is likely the, you know, the thing that should be I, I think focused on for ensuring that this actually works.

	227	[0:28:55.3] Steve Springett: Right.
Modellierungsstrategien	228	[0:28:55.8] Olison Sturm: Yeah.
Best Practices	229	[0:28:57.6] Steve Springett: Yeah.
	230	[0:28:57.8] Steve Springett: Other than that I, I think it's, it should be fairly simple actually.
	231	[0:29:02.6] Olison Sturm: Okay, perfect.
	232	[0:29:04.4] Olison Sturm: Nice to hear.
	233	[0:29:06.5] Olison Sturm: So now I have like a few more questions.
	234	[0:29:08.7] Olison Sturm: I know we're like running out of time.
	235	[0:29:11.1] Olison Sturm: I don't know what, how long do you have time?
	236	[0:29:13.8] Steve Springett: Yeah, I can go on for.
	237	[0:29:15.1] Steve Springett: Let me just check real quick, see if.
	238	[0:29:16.8] Olison Sturm: Yeah.
	239	[0:29:19.7] Steve Springett: I'm actually.
	240	[0:29:20.2] Steve Springett: Okay.
	241	[0:29:21.1] Olison Sturm: Okay, perfect.
	242	[0:29:21.8] Olison Sturm: Because I have like five more questions to be honest.
	243	[0:29:24.8] Olison Sturm: And yeah, that would be it because now we had like the core questions that every expert.
	244	[0:29:29.6] Olison Sturm: Because I'm asking scanner experts as well as OCM experts now I have group specific follow up questions and yeah, would be perfect to get them but most of them are already answered.
	245	[0:29:43.4] Olison Sturm: I have to see.
	246	[0:29:46.7] Olison Sturm: Yeah.
	247	[0:29:54.3] Olison Sturm: So what are common mistakes or misunderstandings you see in other projects that using Cyclone the X.
Best Practices	248	[0:30:03.7] Olison Sturm: So what are they missing for example?
	249	[0:30:05.8] Olison Sturm: Or.

	250	[0:30:09.8] Steve Springett: Let's see the.
Modellierungsstrategien	251	[0:30:16.1] Steve Springett: I think one common thing is the difference between assemblies which are the nested components in CycloneDX.
	252	[0:30:25.1] Olson Sturm: Yeah.
	253	[0:30:25.7] Steve Springett: And dependencies.
	254	[0:30:27.7] Steve Springett: Some people just don't get it.
	255	[0:30:29.5] Olson Sturm: Yeah, that's what I figured out.
	256	[0:30:32.1] Olson Sturm: I had the problem with Grype, I don't know if you know of scanner and I had it with Trivy as well.
	257	[0:30:38.7] Olson Sturm: Trivy don't even scans it and Grype does but gives out CycloneDX that is like flatten and not even not have persisted the structures anymore.
	258	[0:30:51.6] Steve Springett: Yeah, yeah.
	259	[0:30:53.7] Steve Springett: So some folks don't get that.
Best Practices	260	[0:30:59.0] Steve Springett: I think the other complicated thing which we're going to try to uncomplicate is formulation.
CycloneDX-spezifische Strukturen	261	[0:31:08.6] Steve Springett: There's a few tools that support formulation today and it's one of these really simple concepts but the way it's implemented is really, really prescriptive.
Zukunftsperspektiven	262	[0:31:25.7] Steve Springett: So we're coming up with an authoritative guide on actually how to do that.
..Zukunft der SBOM-Standards	263	[0:31:32.4] Steve Springett: Other than that, I mean.
	264	[0:31:35.9] Steve Springett: Oh, the AI folks get confused around.
	265	[0:31:41.1] Steve Springett: Well we call them base models or foundational models.
	266	[0:31:44.5] Steve Springett: And my response is always, well, quit making up words.
	267	[0:31:48.2] Steve Springett: Right.
	268	[0:31:48.5] Steve Springett: It's pedigree.
	269	[0:31:49.6] Steve Springett: Look in the dictionary.
	270	[0:31:50.5] Steve Springett: This is what it's called.
	271	[0:31:53.7] Steve Springett: So you know, we've got different industries that

		kind of make up names for something that already exists because Cyclone is, you know, software, hardware services, you know, AI and you know, some of these industries use different words for the exact same thing.
CycloneDX-spezifische Strukturen	272	[0:32:14.7] Steve Springett: So that's always a challenge.
	273	[0:32:16.4] Olison Sturm: Yeah, I would have another question about licensing.
	274	[0:32:21.6] Olison Sturm: I, if I remember correctly, secondly is using SPDX licensing, right?
	275	[0:32:27.2] Steve Springett: It can, yeah, that's.
Tool- und Technologieeinsatz	276	[0:32:28.8] Steve Springett: It uses, it uses SPDX license IDs and those are strictly validated.
	277	[0:32:37.5] Steve Springett: So it has to be a valid ID uses SPDX expressions which are not validated, but you can use SPDX tools to do that.
	278	[0:32:49.5] Steve Springett: It uses just the name of a license.
CycloneDX-spezifische Strukturen	279	[0:32:54.8] Steve Springett: The unfortunate reality is that when the SPDX specification was created for licensing, they assumed that everybody would use the, what is it, license ref or something like that to specify things that fall outside of, you know, an SPDX license identifier.
Best Practices	280	[0:33:16.6] Steve Springett: The reality is that nobody does like the entire procurement process.
	281	[0:33:22.4] Steve Springett: Nothing, nothing uses the license ref.
	282	[0:33:26.5] Steve Springett: So it was really important for us to be able to have just an unresolved license name as well as all the commercial licensing support so that you can actually bring in procurement teams to the specification.
	283	[0:33:43.6] Olison Sturm: Okay.
	284	[0:33:44.2] Olison Sturm: And you saw the OCM structure and we have for example our OCI references, so an OCI image and binary, like an artifact.
	285	[0:33:53.6] Olison Sturm: And they have, for example I'm using right now, for example Syft, but you can also specify another scanner, Trivy to create the SBOM and what I'm doing then is like merging them together using the hierarchical merge from CycloneDX.
	286	[0:34:12.6] Olison Sturm: I had to adapt it a bit because if I merge and merge and merge, we have different merges.
	287	[0:34:25.7] Olison Sturm: So for example, if it is a parent merge or it's a child merge, that's what I implemented as well.

	288	[0:34:31.0] Olison Sturm: And how would you say, how can I, like while the life cycle of creating OCM, how can I get the.
	289	[0:34:44.9] Olison Sturm: Or do the scanners get the license as well and transfer to CycloneDX?
	290	[0:34:52.2] Olison Sturm: I don't think so, but please.
Tool- und Technologieeinsatz	291	[0:34:54.8] Steve Springett: Yeah, I mean some of them do a decent job, you know, scan code, for example, does a really good job of licensing the ORT from here.
	292	[0:35:10.3] Steve Springett: That actually does a fairly decent job at licensing as well.
	293	[0:35:14.7] Steve Springett: Most of the tools, however, they look at the manifest of whatever it is and that's what they use.
	294	[0:35:22.1] Steve Springett: Including all the CycloneDX tools for the most part we actually do have a licensing utility that does a much better job.
	295	[0:35:29.5] Steve Springett: But in most cases most of the tools are going to use the license.
	296	[0:35:35.5] Steve Springett: That's declared in the manifest, which may or may not be.
	297	[0:35:38.5] Steve Springett: Right.
	298	[0:35:40.8] Olison Sturm: Okay, that would lead to another question.
	299	[0:35:43.4] Olison Sturm: Is there one tool that would you use on your own to like what is right now the best tool to create from an OCI image from an artifact?
	300	[0:35:53.9] Olison Sturm: A CycloneDX SBOM?
	301	[0:35:56.0] Steve Springett: Oh, I, I don't know.
	302	[0:35:57.2] Steve Springett: There's so many.
Best Practices	303	[0:36:00.4] Steve Springett: I mean, I would not actually use one of the open source tools.
	304	[0:36:05.4] Steve Springett: I would likely use one of the commercial tools.
	305	[0:36:12.8] Steve Springett: The reason for that is that the data science teams of commercial tools, they do a really good job not just with vulnerabilities and stuff like that, but licensing as well, even commercial licensing.
	306	[0:36:30.1] Steve Springett: Because just because you have an OCI image doesn't mean everything's going to be open source.

Best Practices	307	[0:36:35.7] Steve Springett: So some of the data science teams, I'm not going to mention any vendors, but some of the data science teams at some of the vendors are really good.
	308	[0:36:46.0] Olison Sturm: Okay, so you will not tell me like which company or which tool is.
	309	[0:36:51.0] Steve Springett: I would totally find.
	310	[0:36:52.3] Olison Sturm: You don't, you don't have.
	311	[0:36:53.2] Steve Springett: I would recommend investigating them.
	312	[0:36:55.3] Olison Sturm: Yeah.
	313	[0:36:55.8] Olison Sturm: Okay.
	314	[0:36:56.2] Steve Springett: We did a, we did a bake off of SCA and container analysis tools and the bake off was done in two parts.
	315	[0:37:09.0] Steve Springett: Can you identify the component and its metadata accurately and if so, then what is the accuracy of the vulnerability intelligence on the first part?
Best Practices	316	[0:37:22.6] Steve Springett: Most tools failed.
Qualitätskriterien	317	[0:37:24.4] Olison Sturm: Okay.
Tool- und Technologieeinsatz	318	[0:37:25.9] Steve Springett: So that's the component identity and all the metadata associated about that component.
	319	[0:37:31.5] Steve Springett: Most tools failed, including all the open source tools.
	320	[0:37:34.2] Olison Sturm: Okay, good to know.
	321	[0:37:35.7] Olison Sturm: But I think I thought I have to stay at open source because open.
	322	[0:37:41.6] Olison Sturm: The open component model is open source, so we can only use open source tools.
Expertise & Kontext	323	[0:37:46.2] Olison Sturm: And that's why I never figured out anything else.
OCM-spezifische Strukturen	324	[0:37:50.5] Olison Sturm: But you can't even recommend an open source tool.
Best Practices	325	[0:37:57.1] Steve Springett: I haven't seen one that's actually, I mean.
	326	[0:38:03.5] Olison Sturm: No, it's fine if there's.
Qualitätskriterien	327	[0:38:04.8] Steve Springett: They all get.
Expertise & Kontext	328	[0:38:05.3] Steve Springett: They all get them wrong in different ways.

	Expertise & Kontext	
		329 [0:38:07.3] Olson Sturm: Okay.
		330 [0:38:07.8] Olson Sturm: Yeah, so I have to figure it out on my own.
		331 [0:38:10.7] Olson Sturm: What I almost did, but yeah, good, thanks.
		332 [0:38:14.3] Olson Sturm: And so you also talked a bit about protobom.
		333 [0:38:17.4] Olson Sturm: That's my last question.
		334 [0:38:19.3] Olson Sturm: Like for the.
		335 [0:38:20.4] Olson Sturm: Yeah, I have like a closing question, but yeah, never mind.
		336 [0:38:24.1] Olson Sturm: And do you think using an interminate abstraction like Protobom, like having the Proto Buff protocol is a better approach than directly map from the OCM to CycloneDX or would you say maybe for like future purposes it's better to map to like a standard in between spdx, CycloneDX to.
		337 [0:38:51.8] Steve Springett: I would actually map it directly.
		338 [0:38:53.9] Steve Springett: Protobom is only going to support the things that are Common between the specifications.
Tool- und Technologieeinsatz		339 [0:39:00.7] Steve Springett: Yeah.
..Middleware (Protobom)		340 [0:39:01.1] Steve Springett: So if you have something in a specification that is not in the other specification, protobom doesn't support it.
Modellierungsstrategien		341 [0:39:10.1] Steve Springett: Additionally, they seem to be lagging in terms of support for different things that Cyclone, for example, already supports, but Protobom does not.
Best Practices		342 [0:39:22.8] Steve Springett: So we've had situations where people have assumed that CycloneDX does not do something simply because Protobom didn't support it.
		343 [0:39:35.8] Steve Springett: And the reality is that Cyclone had supported those things for years.
		344 [0:39:39.4] Steve Springett: It's just a protobom never adopted it.
Zukunftsperspektiven		345 [0:39:43.0] Steve Springett: So I would actually map directly personally we are working on.
		346 [0:39:55.6] Steve Springett: There's a number of specifications coming out of TC54 that we're working on and the Transparency Exchange API is one of the big ones.

	347	[0:40:09.6] Steve Springett: The API is a way to not only exchange the supply chain artifacts like Sboms and C boms and H boms and everything else, but it's.
	348	[0:40:19.7] Steve Springett: And more importantly it's a way to ask questions on from vendors in a completely autonomous manner without having to worry about a bill of material format.
	349	[0:40:35.2] Steve Springett: For example, the normalized model for that is CycloneDX.
Best Practices Zukunftsperspektiven	350	[0:40:42.0] Steve Springett: It doesn't matter what and that's version 2.0, by the way.
CycloneDX-spezifische Struktur	351	[0:40:46.4] Steve Springett: CycloneDX 2.0 not only has the file based format we all know, but it also has an API format that API format is the basis for the Transparency Exchange API, specifically Insights.
	352	[0:41:07.4] Steve Springett: So coming into the API, it can be any BOM format, Cyclone, spdx, cifs, proprietary format doesn't really matter.
	353	[0:41:20.8] Steve Springett: But when you use Insights and you ask questions, that normalized model is essentially the CycloneDX 2.0 API.
	354	[0:41:30.4] Olison Sturm: That's nice.
Zukunftsperspektiven	355	[0:41:31.4] Olison Sturm: So will there be any other versions in between 1.6 and 2.0 or will.
	356	[0:41:36.5] Steve Springett: The next 1.7, which is currently we're trying to get it past the finish line.
Zukunftsperspektiven	357	[0:41:42.0] Steve Springett: We expect that to be ratified in December.
	358	[0:41:46.0] Steve Springett: That's going to be the last of the One X.
	359	[0:41:48.7] Olison Sturm: Okay.
	360	[0:41:49.5] Steve Springett: Everything else is based on 2.0, so yeah, maybe also simultaneously in development.
	361	[0:41:57.5] Olison Sturm: Okay, get it.
Zukunftsperspektiven	362	[0:41:59.7] Olison Sturm: So would you say that there's like in the future only one SBOM standard, like for example, your 2.0 because it's the best one because it sounds really nice but you.
Best Practices Mapping-Kompatibilität	363	[0:42:15.6] Steve Springett: Yeah, I mean in terms of bill of material formats because you have the normalized model and an API, rather than having to deal with file formats, the bill of material format itself becomes a lot less relevant.
Tool- und Technologieeinsatz	364	[0:42:39.9] Steve Springett: Things like protobuf become a lot less necessary.

Mapping-Kompatibilität	365	[0:42:49.7] Steve Springett: Realistically, I don't know if that would.
	366	[0:42:54.6] Steve Springett: One of three things are going to happen.
	367	[0:42:56.6] Steve Springett: One, people will choose CycloneDX just because it's the normalized model.
	368	[0:43:01.6] Steve Springett: Two things will continue on just as they are with multiple BOM formats.
Zukunftsperspektiven Best Practices	369	[0:43:07.5] Steve Springett: And because we have normalization at the API level, the format becomes irrelevant, making everybody's lives easier.
	370	[0:43:19.3] Steve Springett: Or three, it actually encourages more variants in future BOM formats because now they might be able to do things that they couldn't do before because of the constraints of the formats, and you might actually have some innovation actually happening there.
	371	[0:43:36.9] Steve Springett: I don't know what's going to happen.
	372	[0:43:38.7] Steve Springett: Okay, one of those three things are going to happen.
	373	[0:43:42.1] Olison Sturm: Nice.
	374	[0:43:46.0] Olison Sturm: Okay, that's.
	375	[0:43:46.7] Olison Sturm: That's all my.
	376	[0:43:47.6] Olison Sturm: My last question would be, is there anything else you would like to add or.
	377	[0:43:51.2] Olison Sturm: That we haven't talked about yet?
	378	[0:43:57.6] Steve Springett: Not really, other than to, you know, if you.
	379	[0:44:04.1] Steve Springett: If you come up with a tool, you know, make sure you get that listed in the CycloneDX tool center.
	380	[0:44:11.5] Steve Springett: So that way people can discover and find it.
	381	[0:44:15.3] Steve Springett: And if you have any questions, the Slack workspace is a really good place to kind of reach out to folks.
	382	[0:44:22.7] Olison Sturm: Okay, perfect.
	383	[0:44:24.7] Olison Sturm: So that's all.
	384	[0:44:27.0] Olison Sturm: So what I'm doing afterwards is like doing the transcription and like a qualitative analysis of all the interviews I will do.
	385	[0:44:37.9] Olison Sturm: And if you want, optionally, I could give you the

transcript if you not even have it, because maybe on the link you can see it or like, I have to finish my thesis in a month, so I could also.

386 [0:44:53.7] Olison Sturm: Because probably I'm allowed to share it because it's all open source, to send it to you.

387 [0:44:59.4] Steve Springett: I'd be happy to see it.

388 [0:45:00.2] Olison Sturm: Maybe translate it, because I don't write it in English.

389 [0:45:03.2] Olison Sturm: I will write it in German.

390 [0:45:04.2] Steve Springett: German?

391 [0:45:05.0] Olison Sturm: Yeah.

392 [0:45:06.7] Steve Springett: One of the.

393 [0:45:07.0] Steve Springett: One of the guys on our team can translate it for me.

394 [0:45:09.1] Steve Springett: Yeah, Yeah.

395 [0:45:10.7] Steve Springett: I mean, like, I mean, on dependency track side, Nicholas is German.

396 [0:45:17.2] Steve Springett: Jan, on the CycloneDX side, he's German.

397 [0:45:20.7] Steve Springett: So.

398 [0:45:21.0] Steve Springett: Yeah.

399 [0:45:22.3] Olison Sturm: Okay, perfect.

400 [0:45:23.1] Steve Springett: I think we're covered there.

401 [0:45:24.9] Olison Sturm: Nice.

402 [0:45:25.7] Olison Sturm: Yeah.

403 [0:45:26.4] Olison Sturm: And otherwise I should ask on behalf of the team to.

404 [0:45:31.9] Olison Sturm: Yeah.

405 [0:45:32.6] Olison Sturm: To invite you to collaborate with us, the OCM and the SAP team, if you want to.

406 [0:45:39.9] Olison Sturm: Yeah, I. I don't have really an email, I think, or something else, but we would greatly appreciate, like.

407 [0:45:46.5] Olison Sturm: Sure.

- 408 [0:45:46.8] Olison Sturm: Having some talks in the future.
- 409 [0:45:48.4] Steve Springett: Yeah.
- 410 [0:45:48.8] Steve Springett: Feel free to reach out.
- 411 [0:45:49.7] Olison Sturm: Feel free.
- 412 [0:45:52.3] Steve Springett: You did.
- 413 [0:45:52.7] Steve Springett: Your English was great, man.
- 414 [0:45:54.0] Steve Springett: You had no.
- 415 [0:45:55.1] Steve Springett: No, no reason to be nervous about anything.
- 416 [0:45:57.7] Olison Sturm: Thanks.
- 417 [0:45:59.9] Olison Sturm: Yeah.
- 418 [0:46:00.8] Olison Sturm: Thanks for the valuable input.
- 419 [0:46:03.0] Steve Springett: Cheers.
- 420 [0:46:03.8] Steve Springett: Take care.
- 421 [0:46:04.5] Olison Sturm: All right, bye.
- 422 [0:46:05.0] Olison Sturm: Yeah, you too.
- 423 [0:46:05.6] Olison Sturm: Thank you.
- 424 [0:46:06.1] Olison Sturm: Bye.

Anhang D: Experteninterview mit Josh Bressers

- 1 [0:00:00.0]
2 [0:00:00.1] Olison Sturm: Is it okay to take a record of this conversation?
3 [0:00:04.7] Josh Bressers (Anchore): For sure.
4 [0:00:05.6] Olison Sturm: Okay, perfect.
5 [0:00:07.0] Olison Sturm: So first of all, thank you for your time.
6 [0:00:09.6] Olison Sturm: And then yeah, I'm a bit nervous interviews, so I tend to.
7 [0:00:16.7] Olison Sturm: Well, my English tends to suffer a bit maybe, but never mind.
8 [0:00:22.6] Olison Sturm: So we know each other from the community call, right?
9 [0:00:26.9] Josh Bressers (Anchore): Yep, yep, for sure.
10 [0:00:28.0] Olison Sturm: Perfect.
11 [0:00:28.9] Olison Sturm: And yeah, I, I told you I'm working at SAP since yesterday for three years and I'm doing right now at the end of my three years bachelor, my bachelor thesis in.
12 [0:00:43.7] Olison Sturm: In this OCM context in this CycloneDX SBOM context.
13 [0:00:48.8] Olison Sturm: So a brief explanation of the topic is I'm investigating transformation strategies between OCM component descriptors like component trees basically into a standardized CycloneDX SBOM format. I also have a research question.
14 [0:01:08.1] Olison Sturm: I will read it out loud.
15 [0:01:11.3] Olison Sturm: How can the information structure of the open component model be converted into the CycloneDX SBOM format with minimal information loss to enable compatibility.
16 [0:01:25.2] Olison Sturm: Compatibility and established software supply chain tools like Grype and Syft and so on.
17 [0:01:32.5] Olison Sturm: Yeah, so it should be open source tools.
18 [0:01:35.0] Olison Sturm: And first of all, I have some background questions about your expertise context.
19 [0:01:42.1] Olison Sturm: I would start with them.
20 [0:01:43.2] Olison Sturm: How long have you been working in this field of software supply chain security?

Expertise & Kontext

- 21 [0:01:52.0] Josh Bressers (Anchore): Longer than most people have been alive.
- 22 [0:01:54.3] Josh Bressers (Anchore): So I would.
- 23 [0:01:57.0] Josh Bressers (Anchore): The, the.
- 24 [0:01:57.7] Josh Bressers (Anchore): The TLDR answer is since 2004.
- 25 [0:02:01.3] Josh Bressers (Anchore): But the.
- 26 [0:02:02.4] Josh Bressers (Anchore): My.
- 27 [0:02:02.7] Josh Bressers (Anchore): My correct answer is maybe a little more complicated.
- 28 [0:02:07.0] Josh Bressers (Anchore): So I started working at red hat in 2004 and it was on a team we used to call the Security Response Team.
- 29 [0:02:13.0] Josh Bressers (Anchore): And fundamentally our job was to understand everything Red Hat was shipping and then track vulnerabilities in that stuff.
- 30 [0:02:21.0] Josh Bressers (Anchore): And then you can imagine that role evolved over the course of the decade I spent there where were paying attention to many more products, many more technologies.
- 31 [0:02:31.3] Josh Bressers (Anchore): We were starting to ask questions about like is this thing trustworthy?
- 32 [0:02:36.1] Josh Bressers (Anchore): Whereas back in the early days no one cared.
- 33 [0:02:37.9] Josh Bressers (Anchore): We just ran whatever and then I moved to a company called Elastic where I was hired to be a product manager doing security features in the product.
- 34 [0:02:50.2] Josh Bressers (Anchore): But they didn't have the supply chain stuff under control at all.
- 35 [0:02:53.3] Josh Bressers (Anchore): And I was like, I'll just fix it for you.
- 36 [0:02:55.0] Josh Bressers (Anchore): Right?
- 37 [0:02:55.3] Josh Bressers (Anchore): My new job is I, I think I called myself product security there.
- 38 [0:03:00.0] Josh Bressers (Anchore): And that's what I did, right, was get a handle on the things were shipping, excuse me, and then the vulnerabilities in the components and what was trustworthy, what wasn't trustworthy and all that.
- 39 [0:03:11.8] Josh Bressers (Anchore): And then I've made my way to anchor

Expertise & Kontext

- now where I would say prior to this role it was a little more ad hoc than I think this space has become where now we have things like SBOMs, we have very formal tooling for scanning vulnerabilities, tracking vulnerabilities, things like that.
- 40 [0:03:29.2] Josh Bressers (Anchore): And so it's been, I've been in this space a really long time, but it depends how you want to carve it up right where it's been.
- 41 [0:03:35.2] Josh Bressers (Anchore): It's a very formal.
- 42 [0:03:36.8] Josh Bressers (Anchore): I've been in a core since 2001 and then a little more Wild west prior to that.
- 43 [0:03:43.1] Olison Sturm: Okay.
- 44 [0:03:43.5] Olison Sturm: And then what's your current role? And what are your responsibilities in this topic of SBOMs.
- 45 [0:03:51.3] Josh Bressers (Anchore): So I am the vice president of security at Ankor and I have a weird job because I mean we're a small company so I do many things.
- 46 [0:03:59.8] Josh Bressers (Anchore): One of the things I do is I work with the community running things like open SSF meetings, giving interviews, podcasts, that kind of stuff.
- 47 [0:04:08.6] Olison Sturm: Okay.
- 48 [0:04:09.1] Josh Bressers (Anchore): I also am responsible for the supply chain security of the actual Anchor software, the stuff we ship like Syft and Grype being the open source tools as well as Anchor Enterprise and a bunch of other commercial offerings.
- 49 [0:04:22.2] Josh Bressers (Anchore): Like in fact we have a release going out in two days and I spent my morning going through all of our components making sure like everything is correct.
- 50 [0:04:30.7] Josh Bressers (Anchore): We're not going to ship a bunch of vulnerabilities.
- 51 [0:04:32.7] Josh Bressers (Anchore): Like some weird crap didn't show up that no one expected that kind of thing.
- 52 [0:04:36.8] Josh Bressers (Anchore): Right.
- 53 [0:04:37.4] Olison Sturm: Okay.
- 54 [0:04:38.0] Josh Bressers (Anchore): And then I also kind of for the third piece of it is I'm responsible for our information security at the company.
- 55 [0:04:47.1] Josh Bressers (Anchore): But for your purposes that probably

- Expertise & Kontext ↴
- 56 [0:04:49.1] Olison Sturm: Matter as much, but it's nice to know.
- 57 [0:04:54.7] Olison Sturm: So the first background question now about like a bit of OCM, but I could give you maybe first of all the OCM questions, maybe a short introduction into OCM or are you like a bit familiar with the OCM by now?
- 58 [0:05:12.1] Josh Bressers (Anchore): I would say I'm, I, I read the website.
- 59 [0:05:15.0] Josh Bressers (Anchore): So before you sent your email, I'd never heard of OCM before and that's the website.
- 60 [0:05:19.6] Josh Bressers (Anchore): So yes, anything you can tell me would be lovely.
- 61 [0:05:22.2] Olison Sturm: Yeah, perfect.
- 62 [0:05:23.2] Olison Sturm: So you saw the website.
- 63 [0:05:24.9] Olison Sturm: It's basically like packaging all the artifacts and OCM components and like also leverage components between as a.
- 64 [0:05:32.8] Olison Sturm: Like have child components and root components and so on.
- 65 [0:05:36.8] Olison Sturm: So you have a component tree.
- 66 [0:05:38.9] Olison Sturm: I could briefly share my screen to show you a bit in the IDE.
- 67 [0:05:46.7] Olison Sturm: So we have here a YAML file and that's how you.
- 68 [0:05:52.4] Olison Sturm: Wait.
- 69 [0:05:52.8] Olison Sturm: I have the overlay of zoom here.
- 70 [0:05:58.0] Olison Sturm: So to start with an OCM component or with many components.
- 71 [0:06:02.7] Olison Sturm: You need to specify a component constructor and it means you have different components.
- 72 [0:06:10.5] Olison Sturm: This is one component, this one is in this case my parent component.
- 73 [0:06:14.3] Olison Sturm: But it could be every name, but it should be unique.
- 74 [0:06:16.8] Olison Sturm: That's important.
- 75 [0:06:17.8] Olison Sturm: So you have your domain, a namespace and so on and your version and as well who's providing it and stuff.

- 76 [0:06:25.6] Olison Sturm: And then you can have multiple resources.
- 77 [0:06:28.2] Olison Sturm: That's basically or mostly OCI artifacts or anything else where they can show you what we can have here.
- 78 [0:06:37.7] Olison Sturm: So it can be OCI images so we access it from registry or we can have still a blob or helm charts so local objects as well or like binaries basically.
- 79 [0:06:52.6] Olison Sturm: And yeah, as well SBOMs can be included and so on.
- 80 [0:06:57.8] Olison Sturm: So that are the resources.
- 81 [0:06:59.2] Olison Sturm: There are many different types of resources and you can like specify it with access.
- 82 [0:07:04.9] Olison Sturm: So we are going on for example the Docker registry and get it and then we have this component and every component can have multiple like I think there's no limit component references.
- 83 [0:07:18.4] Olison Sturm: So other OCM components.
- 84 [0:07:20.3] Olison Sturm: In this case we have the component name child2, that's only example.
- 85 [0:07:26.6] Olison Sturm: So if we scroll down to child2, we specified it here.
- 86 [0:07:31.4] Olison Sturm: This one has again like multiple resources, multiple artifacts and yeah, child one for example has one with an external identity as well.
- 87 [0:07:42.6] Olison Sturm: Like there are some information like architecture or an operating system or you can specify your own labels for your own use case.
- 88 [0:07:53.6] Olison Sturm: And yeah, that's what you do if you want to ship or deliver your artifacts, your whole system.
- 89 [0:08:01.6] Olison Sturm: With ocm there are not only resources you have in here in a component, there are also source.
- 90 [0:08:09.2] Olison Sturm: It means like basically source code that can be shipped as well.
- 91 [0:08:14.6] Olison Sturm: And yeah, I, to be honest, I'm not so deep in it.
- 92 [0:08:21.0] Olison Sturm: I'm working with it like a bit more than a month as well.
- 93 [0:08:25.0] Olison Sturm: So that's all what I know.

- 94 [0:08:27.9] Olison Sturm: I focused mainly on OCI images, so turning them into SBOMs and merging them, aggregating them to getting a whole SBOM that represents our OCM component tree.
- 95 [0:08:44.9] Olison Sturm: I would say yeah.
- 96 [0:08:46.8] Olison Sturm: Do you have any question?
- 97 [0:08:50.8] Josh Bressers (Anchore): So I, I do have a question.
- 98 [0:08:52.4] Josh Bressers (Anchore): I'm assuming these YAML files and this.
- 99 [0:08:56.0] Josh Bressers (Anchore): I. I couldn't find a good answer to this on the website, but I assume the YAML files are constructed by.
- 100 [0:09:03.0] Olison Sturm: Humans essentially they are right now.
- 101 [0:09:05.5] Olison Sturm: Okay, so they're like plans that make it easier.
- 102 [0:09:09.0] Olison Sturm: But right now the component constructor is constructed by a human so built for their system with all the important resources and sources and so on and what it does then you can like make a component version out of it.
- 103 [0:09:27.4] Olison Sturm: So you get all your blob files here or you like such files and each of them is one component OCM component.
- 104 [0:09:37.7] Olison Sturm: For example, a component descriptor as it called.
- 105 [0:09:41.0] Olison Sturm: So it has mainly the same structure, almost the same structure as a component in the constructor has, except like having some sign fields like digits to sign the OCM or the resources and so on.
- 106 [0:09:59.3] Olison Sturm: So we could see it on the website as well.
- 107 [0:10:02.3] Olison Sturm: So we have here create a component version, we're doing it with the constructor to make it easy for the user and then we get all our component descriptors out of it.
- 108 [0:10:11.0] Olison Sturm: So for every component we have specified in a constructor we get a component descriptor.
- 109 [0:10:17.6] Olison Sturm: And what I'm doing right now is like consuming them and like map all the fields and put them in any way, like in any good way with best practices into a cycloneDX SBOM.
- 110 [0:10:29.2] Olison Sturm: And of course we have like many SBOM per artifact so I have to aggregate them as well.
- 111 [0:10:37.7] Olison Sturm: So that's in my opinion.
- 112 [0:10:39.9] Olison Sturm: But that's why I have all the calls right now.

- 113 [0:10:44.0] Olison Sturm: Before you, I had a call with Stephen Springett, I don't know if you know him, and we had a really good talk about everything and he was like sounds nice your approach and had like some input as well.
- 114 [0:10:57.4] Olison Sturm: And that's why I have all the expert talks as well with the OCM expert tomorrow to make a nice proof of concept.
- 115 [0:11:06.7] Olison Sturm: Yeah and like having a good better thesis.
- 116 [0:11:09.8] Olison Sturm: So that's the context I think. Are there any questions?
- 117 [0:11:14.2] Josh Bressers (Anchore): No, no.
- 118 [0:11:14.9] Josh Bressers (Anchore): And Steve is the best you're going to find for this discussion.
- 119 [0:11:18.3] Josh Bressers (Anchore): So that's all.
- 120 [0:11:19.5] Olison Sturm: Okay, so let's switch back to the questions.
- 121 [0:11:26.2] Olison Sturm: I will stop sharing.
- 122 [0:11:31.7] Olison Sturm: So now you know like kind of OCM.
- 123 [0:11:34.8] Olison Sturm: But in what scenarios do you see the need to transform OCM like structures so the whole OCM tree into a standardized SBOM format such as CycloneDX.
- 124 [0:11:47.8] Olison Sturm: So is it like using scanners like yours or are there many other use cases?
- 125 [0:11:54.3] Olison Sturm: Where do you see use cases?
- 126 [0:11:56.6] Josh Bressers (Anchore): So here's what I would envision and this is from dealing with our customers at Anchor as well as just community conversations and things I've seen is I suspect scanners wouldn't find this data particularly useful because most of the scanners today need to like for example Grype.
- 127 [0:12:19.9] Josh Bressers (Anchore): Grype struggles to scan any SBOM that wasn't created by Syft.
- 128 [0:12:25.6] Josh Bressers (Anchore): And this is like being worked on but it's kind of a long way away.
- 129 [0:12:29.5] Josh Bressers (Anchore): The use case we're seeing is, and this is very SAP adjacent is in heavily regulated industries there.
- 130 [0:12:39.0] Josh Bressers (Anchore): The regulators are asking for information on how this was built, where did it come from, what's going on, and so what's happening.

Expertise & Kontext

Kritik & Gegenargumente

	131	[0:12:47.4] Josh Bressers (Anchore): And this is intriguing to me is what's the SBOM world doesn't keep track of the information like OCM is doing.
Kritik & Gegenargumente	132	[0:12:57.2] Josh Bressers (Anchore): I feel like this is interesting, but also only in certain environments because today we're telling people if you need to track the lifecycle of your software, you need to create a source SBOM and then create a build SBOM before, during and after build.
	133	[0:13:14.3] Josh Bressers (Anchore): And then you need to create an artifact SBOM and then you need to create an SBOM of the thing you're shipping and then an S bomb of the thing you're running.
Expertise & Kontext	134	[0:13:20.9] Josh Bressers (Anchore): And it's kind of like, this is kind of nuts.
	135	[0:13:24.0] Josh Bressers (Anchore): You know, this is a lot of S bombs.
	136	[0:13:26.1] Josh Bressers (Anchore): Yeah, that's the only way we can solve it today.
	137	[0:13:28.5] Josh Bressers (Anchore): Whereas this is a technology that puts a bunch of these pieces together.
	138	[0:13:33.2] Josh Bressers (Anchore): And so it is plausible that this could provide the evidence necessary for regulators who are asking for this kind of stuff saying, where did you get this artifact from?
	139	[0:13:45.3] Josh Bressers (Anchore): Is the artifact you're handing out over here, the artifact you think you got over here?
	140	[0:13:50.5] Josh Bressers (Anchore): I mean, that's one of my favorites, right?
	141	[0:13:51.9] Josh Bressers (Anchore): Is when people are like, where this, where the container come from?
	142	[0:13:55.0] Josh Bressers (Anchore): I can't find this anymore.
	143	[0:13:55.8] Josh Bressers (Anchore): Like, I don't know.
	144	[0:13:56.5] Olison Sturm: So we have to make it traceable, like to go back.
	145	[0:13:59.6] Olison Sturm: Like if we make a CycloneDX SBOM out of OCM, we need to map it as well back to see which artifact is it exactly from which OCM component.
	146	[0:14:10.5] Olison Sturm: That's really important.
Best Practices Mapping-Kompatibilität	147	[0:14:13.8] Josh Bressers (Anchore): For sure.
	148	[0:14:14.5] Josh Bressers (Anchore): And I think the other piece of this though is that I do not know of any systems that can process and do something with OCM documents, at least not yet.

	149	[0:14:26.0] Josh Bressers (Anchore): And SBOMs on the other hand, have become, I think, an accepted standard in the industry.
Best Practices Mapping-Kompatibilität	150	[0:14:31.2] Josh Bressers (Anchore): So it's easy to distribute them.
	151	[0:14:32.9] Josh Bressers (Anchore): There are lots of tools for parsing them and processing them.
	152	[0:14:36.5] Josh Bressers (Anchore): And so that I think would be another advantage is if I gave someone something like an OCM document, they're going to be like, what the hell is that?
	153	[0:14:42.2] Olison Sturm: That's exactly what my team, OCM team is like thinking about to make it like standardized because no one is working right now with SAP.
	154	[0:14:51.8] Olison Sturm: Sorry for.
	155	[0:14:52.5] Olison Sturm: I got a call.
	156	[0:14:53.4] Josh Bressers (Anchore): Oh, you're fine.
	157	[0:14:55.4] Olison Sturm: Okay.
	158	[0:14:56.8] Olison Sturm: Yeah, that's why I got this reach research topic and that's why I'm one of figuring out if.
	159	[0:15:06.4] Olison Sturm: If it is possible to.
	160	[0:15:08.2] Olison Sturm: To map between them.
	161	[0:15:11.1] Josh Bressers (Anchore): Yeah, yeah.
	162	[0:15:12.3] Olison Sturm: Okay.
	163	[0:15:13.1] Olison Sturm: So I have a couple of core questions that every expert from CycloneDX, from OCM and from these scanners get.
	164	[0:15:21.1] Olison Sturm: So are like around seven questions.
	165	[0:15:25.2] Olison Sturm: So to compare them and like make my, like the.
	166	[0:15:28.7] Olison Sturm: The real goal is to identify technical requirements for my proof of concept of this interview.
Tool- und Technologieeinsatz	167	[0:15:35.7] Olison Sturm: Yeah, yeah.
..SBOM-Generatoren	168	[0:15:36.4] Olison Sturm: So the first question would be what tools or technologies do you know or use for generating, merging or transforming SBOMs in particular with CycloneDX?
..Herausforderungen	169	[0:15:49.0] Josh Bressers (Anchore): Yeah, so I mean, I use SIF mostly, you know, unsurprisingly, and that can convert between formats.

- 170 [0:15:57.8] Josh Bressers (Anchore): But what Syft does is we actually have an SBOM format Syft spits out by default called the Syft.
- 171 [0:16:03.3] Josh Bressers (Anchore): SBOM format is not an SBOM format we ever want anyone to use.
- 172 [0:16:06.9] Josh Bressers (Anchore): It's really funny.
- 173 [0:16:08.0] Josh Bressers (Anchore): Alan Friedman was the guy at CISO who is like kind of the.
- 174 [0:16:11.3] Josh Bressers (Anchore): The.
- 175 [0:16:11.6] Josh Bressers (Anchore): He did a lot of work for SBOM community building and he found out about our format and he's like, what the hell is this?
- 176 [0:16:17.1] Josh Bressers (Anchore): I'm like, never use that, Alan.
- 177 [0:16:18.8] Josh Bressers (Anchore): Like, so the prop.
- 178 [0:16:20.0] Josh Bressers (Anchore): One of the challenges we have is you got SPDX, Cyclone DX.
- 179 [0:16:22.5] Josh Bressers (Anchore): Right.
- 180 [0:16:22.7] Josh Bressers (Anchore): You can't convert between the two without losing data.
- 181 [0:16:25.7] Josh Bressers (Anchore): And that's why Anchore just created a third format that is the Venn diagram is those two things.
- 182 [0:16:31.1] Olison Sturm: It's so demo.
- 183 [0:16:33.3] Josh Bressers (Anchore): Right?
- 184 [0:16:34.1] Olison Sturm: Is it the Syft Format?
- 185 [0:16:35.8] Josh Bressers (Anchore): Yeah, yeah.
- 186 [0:16:37.2] Josh Bressers (Anchore): It only exists so we can output either CycloneDX or SPDX.
- 187 [0:16:42.3] Olison Sturm: Okay.
- 188 [0:16:42.7] Josh Bressers (Anchore): And whatever version we want.
- 189 [0:16:44.8] Josh Bressers (Anchore): Right.
- 190 [0:16:45.1] Josh Bressers (Anchore): Because like we can change our format, whatever.

..Herausforderungen

Tool- und Technologieeinsatz

	191	[0:16:47.5] Olison Sturm: So you.
	192	[0:16:48.0] Olison Sturm: You built basically your own mapping.
	193	[0:16:49.9] Olison Sturm: So what should be in CycloneDX, what should be in SPDX?
	194	[0:16:54.8] Olison Sturm: So.
	195	[0:16:55.7] Olison Sturm: And that's why you can't use something like, I don't know, protobomb, for example, because you have like a loss of information.
	196	[0:17:02.5] Olison Sturm: Only the similar fields can be like transferred.
Tool- und Technologieeinsatz	197	[0:17:06.2] Olison Sturm: Right, right.
	198	[0:17:07.1] Olison Sturm: Do you know protobomb for example?
	199	[0:17:08.9] Josh Bressers (Anchore): Right! Yep, I do.
	200	[0:17:09.6] Josh Bressers (Anchore): And protobomb is the same basic idea of what was happening with Syft.
	201	[0:17:13.8] Josh Bressers (Anchore): No.
..Middleware (Protobom)	202	[0:17:14.1] Josh Bressers (Anchore): Is ProtoBOM still active?
	203	[0:17:15.9] Josh Bressers (Anchore): I haven't seen anything from the ProtoBOM.
Best Practices	204	[0:17:18.4] Olison Sturm: I used it in the starting of my prototype, but.
	205	[0:17:21.4] Olison Sturm: And I figured out it's way easier to focus one SBOM format.
	206	[0:17:26.1] Olison Sturm: So CycloneDX and like only map from OCM to CycloneDX and like specify my research like in this.
	207	[0:17:35.6] Josh Bressers (Anchore): Yeah, that is 100% the correct thing to do because it's a mess.
	208	[0:17:40.1] Josh Bressers (Anchore): Good, that's.
	209	[0:17:41.2] Josh Bressers (Anchore): And I know protobomb can do certain transformations.
Best Practices	210	[0:17:45.1] Josh Bressers (Anchore): There's what the tool SBOMit that the openSSF has been working on that could do some transformations and just from a generating perspective like CycloneDX has a million tools that can do all of these things.
..Empfehlungen	211	[0:17:58.1] Josh Bressers (Anchore): So like, for my knowledge, if I was

		looking for a tool, I would use SIF first.
Tool- und Technologieeinsatz Best Practices ..Empfehlungen	212	[0:18:03.9] Josh Bressers (Anchore): Then I would probably just go hunting for something from CycloneDx if I was looking to generate CycloneDx.
	213	[0:18:09.0] Olison Sturm: Yeah.
	214	[0:18:09.5] Josh Bressers (Anchore): Answer your question?
	215	[0:18:10.5] Olison Sturm: Yeah, that answers my question.
	216	[0:18:11.9] Olison Sturm: Thanks.
	217	[0:18:13.1] Olison Sturm: Okay.
	218	[0:18:15.1] Olison Sturm: Okay, second question.
	219	[0:18:17.0] Olison Sturm: Which CycloneDX structures are most critical for ensuring downstream tool capability?
	220	[0:18:22.3] Olison Sturm: Because you're now on the tool part, I think you can answer really good.
	221	[0:18:27.9] Josh Bressers (Anchore): I mean, it depends what you want to do.
	222	[0:18:30.9] Olison Sturm: Right.
Konkrete Anforderungen	223	[0:18:32.0] Josh Bressers (Anchore): So from the perspective of what I do at Anchore with the data, the most important thing we use is just the package identification.
CycloneDX-spezifische Struktur	224	[0:18:43.1] Josh Bressers (Anchore): Generally the PURL that the scanners stick inside of the SBOMs and it varies, right, Depending upon what you want to do because.
Tool- und Technologieeinsatz	225	[0:18:54.3] Josh Bressers (Anchore): Sorry, what?
	226	[0:18:54.9] Olison Sturm: Yeah, so you mainly using the pearls or you use SWID or any other CPE or something.
	227	[0:19:04.0] Josh Bressers (Anchore): So it depends is the answer.
	228	[0:19:06.2] Josh Bressers (Anchore): The PURL is the best, generally speaking.
	229	[0:19:09.2] Josh Bressers (Anchore): But the way Grype matches packages is right now based on ecosystems.
	230	[0:19:15.9] Josh Bressers (Anchore): So for example, I would say the vast majority of ecosystems we match against, we're going to look at the, going to look for pearls first.

	231	[0:19:21.9] Josh Bressers (Anchore): If there isn't a pearl, we're going to say is there a cpe?
	232	[0:19:25.9] Josh Bressers (Anchore): And if there isn't a cpe, we're just going to basically try to figure out what the heck is this thing.
	233	[0:19:30.7] Olison Sturm: Okay.
Tool- und Technologieeinsatz ..Herausforderungen	234	[0:19:31.9] Josh Bressers (Anchore): It's.
	235	[0:19:32.3] Josh Bressers (Anchore): It's kind of dicey and you can imagine the quality varies greatly depending upon what we end up matching with.
	236	[0:19:38.8] Josh Bressers (Anchore): But fundamentally the package identification is the thing we care about the most.
	237	[0:19:43.7] Josh Bressers (Anchore): And that's what our tooling is focused on today.
	238	[0:19:46.2] Josh Bressers (Anchore): That's what people seem to care about.
CycloneDX-spezifische Strukturen	239	[0:19:48.6] Josh Bressers (Anchore): Now once you get into questions about like provenance now we have, you know, checksums of things that are coming in there.
	240	[0:19:57.2] Josh Bressers (Anchore): There's various.
	241	[0:19:58.2] Josh Bressers (Anchore): Every ecosystem has a different sort of identifier.
	242	[0:20:01.1] Josh Bressers (Anchore): Some have checksums of a package, some might have checksums of everything in a package.
	243	[0:20:05.6] Josh Bressers (Anchore): You know, it's a freaking mess.
	244	[0:20:07.3] Olison Sturm: Yeah.
Konkrete Anforderungen	245	[0:20:07.8] Josh Bressers (Anchore): But I would say to.
	246	[0:20:09.5] Josh Bressers (Anchore): If I was going to answer your question, which I guess I can.
	247	[0:20:11.9] Josh Bressers (Anchore): It's.
	248	[0:20:12.2] Josh Bressers (Anchore): It's the package information is what we truly care about.
	249	[0:20:14.9] Olison Sturm: Okay, and what do you do with the package URL?
Tool- und Technologieeinsatz	250	[0:20:18.0] Olison Sturm: So you search in the database or where do you get the vulnerability information?

	251	[0:20:23.8] Josh Bressers (Anchore): Yeah, yeah, it's a database.
	252	[0:20:25.4] Josh Bressers (Anchore): We have a tool it's all open source.
Tool- und Technologieeinsatz ..Herausforderungen	253	[0:20:27.9] Josh Bressers (Anchore): I'll just reply to your email after we're done talking, Olson. We have a.
	254	[0:20:32.9] Josh Bressers (Anchore): We call it GrypeDB.
	255	[0:20:34.7] Josh Bressers (Anchore): So Grype every once a day it will check, is there a new database I can download?
	256	[0:20:40.4] Josh Bressers (Anchore): It will download the database and then that's what it uses.
	257	[0:20:43.4] Josh Bressers (Anchore): And then we have all our tooling for generating.
	258	[0:20:45.6] Josh Bressers (Anchore): The database is all open source.
	259	[0:20:47.9] Josh Bressers (Anchore): So you can see like where we're pulling information from.
	260	[0:20:50.7] Josh Bressers (Anchore): We get it from like GitHub and like Red Hat and suse and we use nvd but the NVD is a mess so we also enrich NVD data.
	261	[0:20:58.8] Josh Bressers (Anchore): Now it's like this.
	262	[0:21:00.7] Josh Bressers (Anchore): It's.
	263	[0:21:01.3] Josh Bressers (Anchore): If I, if I drew you a flow diagram you'd be like, that's the dumbest thing I've ever seen because none of it makes sense.
	264	[0:21:10.0] Josh Bressers (Anchore): But I'll send you some links and you can go.
Mapping-Kompatibilität	265	[0:21:12.2] Olson Sturm: Thanks a lot.
	266	[0:21:13.2] Olson Sturm: And what would you say, which OCM concepts?
	267	[0:21:20.8] Olson Sturm: So like we have the hierarchical structure in OCM and also the resources and sources and many different fields.
Tool- und Technologieeinsatz CycloneDX-spezifische Struktur	268	[0:21:30.0] Olson Sturm: What are the difficults or are there any.
	269	[0:21:33.9] Olson Sturm: Is it impossible to represent it in CycloneDX?
	270	[0:21:37.2] Olson Sturm: What would you say?
	271	[0:21:38.3] Josh Bressers (Anchore): I mean, I think a lot of this, I don't think it's impossible, but I think it's going to be difficult because you just have like especially a cyclonedx they have so many like types of SBOMs that they

		support and obviously if you're talking about source code versus build time versus what signed, what's not signed?
OCM-spezifische Strukturen	272	[0:21:57.1] Josh Bressers (Anchore): Once you get into things like how is it deployed?
Mapping-Kompatibilität	273	[0:21:59.5] Josh Bressers (Anchore): Because I know like OCM has you know, deployment instructions and instructions on how to like move things around.
Tool- und Technologieeinsatz	274	[0:22:04.6] Josh Bressers (Anchore): We can't represent a lot of that very well in SBOMs today, right?
CycloneDX-spezifische Struk	275	[0:22:07.4] Josh Bressers (Anchore): S bombs are very good at like I have a static thing on a disk and I'm going to look at it and poke it with a stick.
	276	[0:22:15.3] Josh Bressers (Anchore): But once you.
	277	[0:22:16.1] Josh Bressers (Anchore): Once things start to move, then I think SBOMs capabilities sort of fall apart.
Zukunftsperspektiven	278	[0:22:21.1] Josh Bressers (Anchore): I'm sure Steve has something they're working on because Cyclone DX runs at like a million miles an hour and I can't keep track of anything they're doing ever.
	279	[0:22:29.1] Olison Sturm: You know about CycloneDX 2.0?
	280	[0:22:32.7] Josh Bressers (Anchore): I have absolutely no idea what they're working on.
	281	[0:22:35.2] Olison Sturm: He told me before they are not focusing on BOM, like bill of material anymore in 2.0.
	282	[0:22:42.6] Olison Sturm: They will focus on a transparency format so it's more open, like everyone can use it and there's not even a bill of material anymore used.
	283	[0:22:56.6] Olison Sturm: So that's what he.
	284	[0:22:57.5] Olison Sturm: Yeah, I don't know how to, you.
	285	[0:22:58.8] Josh Bressers (Anchore): Know, what's going to be the problem CycloneDX has is there, like, no one has figured out like, boring SBOMs yet.
..Zukunft der SBOM-Standards	286	[0:23:05.5] Josh Bressers (Anchore): And they're 10 years in the future and the problem they're going to have is they're moving so fast.
	287	[0:23:10.7] Josh Bressers (Anchore): People are going to be like, I don't know what's going on.
	288	[0:23:12.6] Josh Bressers (Anchore): And this scares me.
	289	[0:23:13.5] Josh Bressers (Anchore): I'm just going to go use like, SPDX is slow and boring.

	290	[0:23:16.6] Josh Bressers (Anchore): I'll go use that.
..Zukunft der SBOM-Standards	291	[0:23:17.9] Josh Bressers (Anchore): It's just going to hurt them, I think, and they don't know this.
	292	[0:23:21.1] Josh Bressers (Anchore): I've tried telling Steve this more than once and he's like, no, it's fine.
	293	[0:23:24.1] Josh Bressers (Anchore): I'm like, I don't think it is Steve.
	294	[0:23:26.9] Josh Bressers (Anchore): Like.
	295	[0:23:28.7] Olison Sturm: So you really know each other?
	296	[0:23:30.5] Olison Sturm: You have often, like, talks.
	297	[0:23:32.3] Josh Bressers (Anchore): Oh, yeah.
	298	[0:23:33.1] Josh Bressers (Anchore): I mean, I've known Steve for a long time.
	299	[0:23:35.1] Josh Bressers (Anchore): He's.
	300	[0:23:35.4] Josh Bressers (Anchore): He's really good.
	301	[0:23:36.1] Josh Bressers (Anchore): He's so smart.
	302	[0:23:37.2] Josh Bressers (Anchore): Steve has the problem.
	303	[0:23:38.3] Josh Bressers (Anchore): He's so much smarter than everyone else.
	304	[0:23:39.9] Josh Bressers (Anchore): He doesn't know how dumb we are. Haha. Like.
	305	[0:23:43.9] Olison Sturm: Haha. So good that I got him as an expert and you as well, of course.
	306	[0:23:48.4] Josh Bressers (Anchore): But I'm a nobody.
	307	[0:23:49.8] Josh Bressers (Anchore): Like, Steve is where it's at.
	308	[0:23:52.7] Olison Sturm: Okay, let's.
OCM-spezifische Strukturen	309	[0:23:54.2] Olison Sturm: Let's continue.
CycloneDX-spezifische Struk	310	[0:23:55.5] Olison Sturm: Which options exist for modeling component hierarchies from ocm?
Modellierungsstrategien	311	[0:24:01.3] Olison Sturm: What do you think?
Mapping-Kompatibilität	312	[0:24:02.4] Olison Sturm: Like, what could I use to build it in Cyclone DX?
Best Practices		

	313	[0:24:06.8] Josh Bressers (Anchore): Well, I mean, CycloneDX has the.
	314	[0:24:09.0] Josh Bressers (Anchore): I know they have.
CycloneDX-spezifische Struktur	315	[0:24:09.9] Josh Bressers (Anchore): I forget the term off the top of my head, but I know they have the ability to represent like.
	316	[0:24:14.3] Josh Bressers (Anchore): Like dependency trees.
	317	[0:24:16.3] Olison Sturm: Yeah.
	318	[0:24:16.8] Olison Sturm: Like nested components, I think, or embedded components.
	319	[0:24:19.4] Olison Sturm: It's.
Best Practices	320	[0:24:19.8] Josh Bressers (Anchore): Yeah, yeah, right.
	321	[0:24:20.8] Josh Bressers (Anchore): I know they do that.
	322	[0:24:21.8] Josh Bressers (Anchore): And then there's can reference external documents which I don't remember what they call it.
Modellierungsstrategien	323	[0:24:28.5] Josh Bressers (Anchore): We just call it in my world.
	324	[0:24:30.6] Olison Sturm: Okay.
Tool- und Technologieeins ..Herausforderungen	325	[0:24:31.9] Josh Bressers (Anchore): But I think kind of those two things can probably do whatever you need today.
Zukunftsperspektiven	326	[0:24:37.5] Josh Bressers (Anchore): I mean, one of the challenges though, is when you have a lot of SBOMs, there aren't a lot of tools that properly stitch together all of the external references.
	327	[0:24:47.0] Josh Bressers (Anchore): I think today most tooling just treats it as like multiple separate documents.
	328	[0:24:52.0] Josh Bressers (Anchore): And we think maybe someday we'll do something with these.
	329	[0:24:54.7] Josh Bressers (Anchore): But I think from the, from what I understand of ocm, that is a more reasonable way to think about what's going on.
	330	[0:25:02.4] Josh Bressers (Anchore): Right.
	331	[0:25:02.6] Josh Bressers (Anchore): Where you.
	332	[0:25:03.0] Olison Sturm: Yeah.
	333	[0:25:03.3] Olison Sturm: Like a whole overview over everything.

	334	[0:25:05.8] Josh Bressers (Anchore): Yeah, yeah.
	335	[0:25:06.4] Josh Bressers (Anchore): Points down to other things.
	336	[0:25:08.7] Olison Sturm: Yeah.
	337	[0:25:10.0] Olison Sturm: Okay, nice.
	338	[0:25:11.9] Olison Sturm: From your perspective, what are essential quality criteria that we have by generating an SBOM?
Qualitätskriterien	339	[0:25:23.6] Josh Bressers (Anchore): So I have a unique perspective probably in that there's all these discussions about quality all the time.
	340	[0:25:30.8] Josh Bressers (Anchore): I don't care about quality.
	341	[0:25:32.3] Josh Bressers (Anchore): I don't think that's a meaningful term.
	342	[0:25:34.4] Josh Bressers (Anchore): I care about making a document that can be.
	343	[0:25:36.9] Josh Bressers (Anchore): That can solve a problem someone has.
	344	[0:25:39.2] Josh Bressers (Anchore): In our case, most of the use cases we see are focused very heavily on vulnerabilities and attestation, where when I say attestation, I don't mean digital signatures because no one can get that right today.
	345	[0:25:50.7] Josh Bressers (Anchore): I just mean saying I have a file of this checksum.
Best Practices ..Vollständigkeit	346	[0:25:53.9] Josh Bressers (Anchore): The SBOM says the file should have this checksum.
	347	[0:25:56.8] Josh Bressers (Anchore): They're not the same.
	348	[0:25:57.5] Josh Bressers (Anchore): Something's wrong.
	349	[0:25:58.5] Josh Bressers (Anchore): Right.
	350	[0:25:59.3] Josh Bressers (Anchore): And that is what we see people use it for today also.
Konkrete Anforderungen Tool- und Technologieeinsatz	351	[0:26:02.5] Josh Bressers (Anchore): Then there's just the whole vulnerability scanning aspect of I need to know what packages I have in a way that Grype can understand them.
	352	[0:26:11.7] Josh Bressers (Anchore): And like we're working on making Grype understand lots of different ways and transforming things and whatever.
	353	[0:26:17.1] Josh Bressers (Anchore): In the third thing we see is the log 4J case where someone says, have I ever had log 4J running in my environment?

			[0:26:25.1] Josh Bressers (Anchore): Like I've got this huge corpus of SBOMs.
Qualitätskriterien	354		[0:26:27.8] Josh Bressers (Anchore): Is log 4J somewhere in this data?
Best Practices	355		[0:26:30.8] Josh Bressers (Anchore): And again, that's very similar to the Gype use case of I have.
Konkrete Anforderungen	356		[0:26:35.7] Josh Bressers (Anchore): I have to be able to identify packages.
Tool- und Technologieeinsatz	357		[0:26:38.4] Olison Sturm: Okay, got it.
	358		[0:26:43.2] Olison Sturm: So wait, I will see.
	359		[0:26:46.4] Olison Sturm: Okay.
Konkrete Anforderungen	360		[0:26:47.0] Olison Sturm: The next question would be, would you say there's a potential risk or limitation for pursuing a direct OCM to CycloneDX mapping approach?
	361		[0:26:59.9] Josh Bressers (Anchore): I mean, I would be worried about losing some of the richness, I think.
	362		[0:27:04.2] Josh Bressers (Anchore): So here's what I would say.
Mapping-Kompatibilität	363		[0:27:06.6] Josh Bressers (Anchore): If I'm going to put on like a manager hat and go up like 10 levels from where you're at is today ocm, if there is some sort of standard underneath it, which I don't understand at all, like I've never heard of it before, it's clear it's not heavily used.
	364		[0:27:22.4] Josh Bressers (Anchore): You know, maybe SAP uses it all.
	365		[0:27:24.2] Josh Bressers (Anchore): Well, you probably do, but it's not widely adopted.
	366		[0:27:27.9] Josh Bressers (Anchore): Right.
	367		[0:27:28.7] Josh Bressers (Anchore): So now let's say we need to turn it into something that is a standard and is widely used.
..Konzeptuelle Inkompatibilität	368		[0:27:34.1] Josh Bressers (Anchore): That's fine.
	369		[0:27:35.0] Josh Bressers (Anchore): But like just from looking at the small amount of information I've gleaned from the website and checking, chatting with you, Allison, is there is more richness in the OCM data than I would say you can capture in SBOMs today.
	370		[0:27:51.5] Olison Sturm: Yeah.
	371		[0:27:53.0] Olison Sturm: You're going to lose some of that, I understand.
	372		[0:27:56.3] Olison Sturm: Yeah.
	373		

- 374 [0:27:56.7] Olison Sturm: The future of OCM is mainly that it is whole, wholly accepted in SAP.
- 375 [0:28:03.0] Olison Sturm: So every delivery process will Use OCM at SAP, but mainly as well, like making impact in the open source community and like, yeah, making it accessible for everyone using the open component model.
- 376 [0:28:18.3] Olison Sturm: Because I don't know if there are many different approaches of like doing that, what OCM does.
- 377 [0:28:24.9] Olison Sturm: I think maybe Zarf or something if you know it.
- 378 [0:28:29.7] Olison Sturm: But yeah, yeah.
- 379 [0:28:32.2] Josh Bressers (Anchore): Well, and this looks nicer because Zarfs is a mess.
- 380 [0:28:35.7] Josh Bressers (Anchore): Like it's terrible xml.
- 381 [0:28:37.6] Josh Bressers (Anchore): I think they have a JSON version now, but it's still really complicated.
- 382 [0:28:42.1] Josh Bressers (Anchore): But now the other side of this though is what you showed me and what we discussed.
- 383 [0:28:46.3] Josh Bressers (Anchore): It's very like humans kind of put the pieces together.
- 384 [0:28:49.7] Olison Sturm: Yeah.
- 385 [0:28:50.3] Josh Bressers (Anchore): Most people don't want a human touch this stuff.
- 386 [0:28:52.8] Josh Bressers (Anchore): They just want to point a tool at it and be like, go nuts, I'm done.
- 387 [0:28:56.4] Olison Sturm: Right.
- 388 [0:28:57.5] Josh Bressers (Anchore): And so I think one of the challenges I could see OCM having in the future is as SBOM automation continues to mature and receive development, it's going to get better.
- 389 [0:29:08.5] Josh Bressers (Anchore): Right.
- 390 [0:29:08.8] Josh Bressers (Anchore): And eventually it's going to be able to do the things like OCM can do today that a human has to do.
- 391 [0:29:14.1] Josh Bressers (Anchore): Except it'll all be magic.
- 392 [0:29:16.4] Olison Sturm: I understand that, yes.
- 393 [0:29:16.9] Olison Sturm: Okay.

OCM-spezifische Strukturen

Kritik & Gegenargumente

Zukunftsperspektiven

	394	[0:29:19.0] Olison Sturm: So that would be the core questions I have now, like group specific follow up questions.
	395	[0:29:25.7] Olison Sturm: Like I had one for RCM experts in cyclonetics, but now I have like some questions about the in general SBOM scanners.
	396	[0:29:32.8] Josh Bressers (Anchore): Sure.
	397	[0:29:33.2] Olison Sturm: And so the first one would be which cyclomdix fields does your tool rely on and detect?
	398	[0:29:42.6] Olison Sturm: Okay, I already asked.
	399	[0:29:44.6] Olison Sturm: Yeah, yeah.
	400	[0:29:45.4] Olison Sturm: Okay.
	401	[0:29:46.6] Olison Sturm: And let's move on with the next one.
	402	[0:29:49.0] Olison Sturm: Are there SBOM fields or formats you find prompt on problematic like underspecified or redundant or.
	403	[0:30:00.0] Josh Bressers (Anchore): I cannot answer that question for you.
	404	[0:30:02.1] Josh Bressers (Anchore): I. Yeah, if you'd like I can put you in touch with the people who can.
	405	[0:30:06.4] Olison Sturm: Okay.
	406	[0:30:06.9] Josh Bressers (Anchore): Because our developers who work on this stuff every day, 100% know the answer to that.
	407	[0:30:11.8] Josh Bressers (Anchore): But from my perspective, I have no idea.
	408	[0:30:14.6] Olison Sturm: Okay, it's fine, no worries.
Konkrete Anforderungen	409	[0:30:19.3] Olison Sturm: So what condition must the CycloneDX SBOM be under to meet like all the requirements for your tool, like Grype?
CycloneDX-spezifische Struk	410	[0:30:32.1] Josh Bressers (Anchore): I cannot give you the list off the top of my head.
Best Practices	411	[0:30:34.4] Josh Bressers (Anchore): I mean the biggest one is going to be capturing the package information.
Qualitätskriterien	412	[0:30:40.1] Josh Bressers (Anchore): That is, that is what we need and I can either give you a person you can ask that to or I can probably just figure it out and we can send you some information.
	413	[0:30:48.9] Olison Sturm: Thanks.
	414	[0:30:49.2] Olison Sturm: It's fine.

	415	[0:30:50.3] Olson Sturm: So then there's another question because I figured out and that what we had, that was the topic about in the community call. Right now If I give Grype an SBOM and this has like a nested component structure, hierarchical structure, you can make a CycloneDX SBOM like as an output, but you not persist the whole structure of the CycloneDX, you create a new one with the different metadata tool and so on.
Mapping-Kompatibilität	416	[0:31:31.7] Olson Sturm: And you like flatten all the components, right?
	417	[0:31:35.1] Olson Sturm: As an output.
	418	[0:31:36.5] Olson Sturm: Yeah.
	419	[0:31:40.0] Olson Sturm: And as well in Trivy we have the same issue, almost like they don't even understand like only the top layer, not even the layers that are nested.
	420	[0:31:55.5] Olson Sturm: So is it a use case to support it?
Modellierungsstrategien	421	[0:32:05.7] Josh Bressers (Anchore): I would say yes, we want to do it right.
Tool- und Technologieeinsatz	422	[0:32:09.3] Josh Bressers (Anchore): I mean, so the reason Syft doesn't do this correctly is because the documenting nested packages is not, we'll say as standard as I wish it was.
	423	[0:32:24.6] Josh Bressers (Anchore): So what we see is we can get.
Zukunftsperspektiven	424	[0:32:27.4] Josh Bressers (Anchore): There are packaging ecosystems that will tell you what the relationships are between, you know, dependencies and there are packaging ecosystems that don't, where they'll just give you like here's a flat list of everything that's been installed.
	425	[0:32:42.1] Josh Bressers (Anchore): And so Syft will try to capture that if it can and it will try to document it if it can, but it's not always there.
	426	[0:32:48.8] Josh Bressers (Anchore): And so one of the, this is just one of those examples of like we can't do this as well as we want to and so we end up throwing away a lot of information when we end up doing some conversions to other formats just because we can't guarantee it's there for sure.
..Herausforderungen	427	[0:33:03.5] Josh Bressers (Anchore): And it's, we know it's a problem, we want to fix it, but it just hasn't.
	428	[0:33:08.9] Josh Bressers (Anchore): I think the other challenge is we're not seeing kind of demand for this yet.
	429	[0:33:13.7] Josh Bressers (Anchore): We're generally speaking, when customers in the field are using this data, they're not as worried about having dependency trees.
	430	[0:33:23.2] Josh Bressers (Anchore): They're looking at lists of vulnerabilities.

			[0:33:26.5] Josh Bressers (Anchore): And I think this also comes into how we're using this tooling today is very compliance focused compliance like your auditors you've talked to, they don't care what your dependency tree is.
..Herausforderungen		431	[0:33:26.5] Josh Bressers (Anchore): And I think this also comes into how we're using this tooling today is very compliance focused compliance like your auditors you've talked to, they don't care what your dependency tree is.
		432	[0:33:39.3] Josh Bressers (Anchore): They're just going to say if you have a critical in your application, you're fixing it.
		433	[0:33:43.8] Josh Bressers (Anchore): They don't care.
Modellierungsstrategien		434	[0:33:45.6] Josh Bressers (Anchore): So I think until we see some of the compliance standards starting to catch up with reality and understanding that yes, there are like trees of dependencies, how things are used is different.
Tool- und Technologieeinsatz		435	[0:33:55.8] Josh Bressers (Anchore): I think VEX is going to help with this in the future just because that seems to be getting a lot of attention from a lot of vendors.
Zukunftsperspektiven		436	[0:34:01.4] Josh Bressers (Anchore): But for the moment, yes, we know it's a problem, we would like to fix it.
		437	[0:34:05.8] Josh Bressers (Anchore): But it is, I guess low on the priority list because we're, we get more bugs about like proper component identification and proper vulnerability identification of those components.
		438	[0:34:18.1] Olison Sturm: Okay.
		439	[0:34:18.8] Josh Bressers (Anchore): The, the, the universe just hasn't made it there yet.
		440	[0:34:23.0] Olison Sturm: Yeah, get it.
		441	[0:34:23.8] Olison Sturm: So there are different other priorities, of course.
		442	[0:34:26.6] Josh Bressers (Anchore): Yep, exactly.
		443	[0:34:28.2] Olison Sturm: Yeah, but why can you like export a cycloneDX SBOM, like as an output?
		444	[0:34:36.7] Olison Sturm: Of course there's a section vulnerabilities and you display all the vulnerabilities and you say for which component in the components this vulnerability is.
		445	[0:34:47.9] Olison Sturm: But how Usually use your users CycloneDX.
		446	[0:34:54.6] Olison Sturm: Would they output it as a CycloneDX or would they like in a different format?
		447	[0:35:01.4] Olison Sturm: Like both.
		448	[0:35:03.2] Josh Bressers (Anchore): I mean.

- 449 [0:35:03.6] Josh Bressers (Anchore): Well, I shouldn't say both.
- 450 [0:35:04.5] Josh Bressers (Anchore): Well, we support Cyclone DX and SPDX fundamentally.
- 451 [0:35:10.1] Josh Bressers (Anchore): And so what happens?
- 452 [0:35:12.2] Josh Bressers (Anchore): Excuse me, our customers, I mean, I don't know exactly what people are doing with like Syft, you know, just in the universe, but our customers, we, when we scan something, it's stored as the Syft-json format and then they can Download SPDX or CycloneDx whatever they want.
- 453 [0:35:30.7] Josh Bressers (Anchore): We don't care.
- 454 [0:35:31.6] Josh Bressers (Anchore): When people ask me what to do, I tell them just distribute both.
- 455 [0:35:35.8] Olison Sturm: No, I, I might.
- 456 [0:35:36.9] Olison Sturm: I meant, maybe I said it wrong.
- 457 [0:35:39.0] Olison Sturm: I meant Grype.
- 458 [0:35:41.6] Josh Bressers (Anchore): Oh, what about Grype?
- 459 [0:35:43.2] Olison Sturm: Yeah, like the same.
- 460 [0:35:44.8] Olison Sturm: Like you can output the Grype vulnerabilities as an as CycloneDX.
- 461 [0:35:50.6] Josh Bressers (Anchore): Yes.
- 462 [0:35:51.1] Olison Sturm: S bomb.
- 463 [0:35:51.9] Olison Sturm: That's what I meant.
- 464 [0:35:52.9] Josh Bressers (Anchore): Yeah, it does that.
- 465 [0:35:53.7] Josh Bressers (Anchore): Yeah, it does.
- 466 [0:35:54.3] Josh Bressers (Anchore): Cyclone.
- 467 [0:35:55.0] Olison Sturm: There's, there's a use case for it.
- 468 [0:35:57.0] Olison Sturm: Like, or how would you use the output of Grype?
- 469 [0:36:01.8] Josh Bressers (Anchore): Oh, you're asking if there's a use case for the Cyclone DX format?
- 470 [0:36:06.5] Josh Bressers (Anchore): I mean, for sure.
- 471 [0:36:08.2] Josh Bressers (Anchore): It, it just depends what you're doing with

Tool- und Technologieeinsatz

		it.
	472	[0:36:10.6] Josh Bressers (Anchore): Right.
Tool- und Technologieeinsatz	473	[0:36:10.9] Josh Bressers (Anchore): Like so from our perspective, in the enterprise tool we use Grype underneath Grype outputs in its own JSON, which again, it's same idea with the Syft-json where we just captured more data than we need.
Best Practices	474	[0:36:26.8] Josh Bressers (Anchore): And then there are policies that get applied and enforced as well as reports and that's what we're seeing our customers use.
	475	[0:36:36.0] Josh Bressers (Anchore): I know there are many people in the community that output a variety of formats from Grype, where some of them are using the CycloneDx format, some of them are using Grype-json, some of them are using just the text table.
	476	[0:36:49.2] Josh Bressers (Anchore): The thing spits out.
	477	[0:36:52.0] Josh Bressers (Anchore): I guess it just depends what you're going to do with it because obviously spinning out a JSON file is worthless if you don't have somewhere to put a JSON file versus the human readable table format.
	478	[0:37:02.7] Josh Bressers (Anchore): Everyone can understand.
	479	[0:37:03.7] Olison Sturm: That's for me the really hard part in my whole research because I don't really know who and how someone is using ocm.
	480	[0:37:13.8] Olison Sturm: And then as well if, if they like transforming OCM to CycloneDX and that's my part basically how they want to have the output and how should look the output as CycloneDX output and what will they do with the output?
	481	[0:37:27.9] Olison Sturm: Like putting them to some scanners, open source scanners or scanner from a company like different vendors.
Best Practices	482	[0:37:35.2] Olison Sturm: And I mean I can tell you.
	483	[0:37:38.1] Josh Bressers (Anchore): From my perspective of dealing with companies and community and everyone, absolutely no one has answer to that question.
Qualitätskriterien	484	[0:37:46.3] Josh Bressers (Anchore): Today where there's a ton of demand, like we have a number of large German automakers we talk to about SBOM.
	485	[0:37:55.2] Josh Bressers (Anchore): Like what do we do with these things?
	486	[0:37:56.4] Josh Bressers (Anchore): We're like, we'll put them in the system, like someday they'll be useful.

- Best Practices
- Qualitätskriterien
- 487 [0:38:02.6] Josh Bressers (Anchore): And I think today what we're seeing is the regulations are starting to say you need an SBOM.
- 488 [0:38:08.8] Josh Bressers (Anchore): They're not saying what you need to do with the SBOM.
- 489 [0:38:11.3] Josh Bressers (Anchore): So it's very much like we bought your tool, we're putting SBOMs in it.
- 490 [0:38:14.7] Josh Bressers (Anchore): Like the box is checked, go away, we're done.
- 491 [0:38:17.1] Olison Sturm: Okay, then it's fine.
- 492 [0:38:20.8] Olison Sturm: So if I generate my Cyclone DX like the specification of Cyclone DX allows it, then I have a good solution.
- 493 [0:38:30.3] Olison Sturm: Would I say probably.
- 494 [0:38:31.7] Josh Bressers (Anchore): I think that's step one.
- Qualitätskriterien
- 495 [0:38:33.2] Josh Bressers (Anchore): Right.
- 496 [0:38:33.5] Josh Bressers (Anchore): And then step two is when people want to use it.
- 497 [0:38:39.2] Josh Bressers (Anchore): But this is I guess a very open source model.
- 498 [0:38:42.7] Josh Bressers (Anchore): You need to be quick.
- 499 [0:38:44.0] Josh Bressers (Anchore): Right?
- 500 [0:38:44.4] Olison Sturm: Okay.
- 501 [0:38:45.1] Olison Sturm: Yeah.
- 502 [0:38:45.6] Olison Sturm: That's, that's why I have like some experts from scanners.
- 503 [0:38:49.2] Olison Sturm: So like you, because I thought maybe you know how like a CycloneDX SBOM should look because you are at the end consuming it for giving like vulnerability data and stuff.
- 504 [0:39:00.7] Olison Sturm: That's, that's was my thought.
- 505 [0:39:02.9] Josh Bressers (Anchore): Yep, for sure.
- 506 [0:39:04.3] Olison Sturm: Okay, let me see for another question.
- Best Practices
- 507 [0:39:17.7] Olison Sturm: Yeah, this question.

- 508 [0:39:19.0] Olison Sturm: Could you like see in the future that canners like Grype or other scanners will understand OCM or would you think it's really important to have some standardized SBOM format because that's who everyone is working with and.
- 509 [0:39:40.8] Olison Sturm: Or is there any possibility to like maybe understand OCM in the future?
- 510 [0:39:48.1] Josh Bressers (Anchore): I mean I would not expect it.
- 511 [0:39:49.6] Josh Bressers (Anchore): I'll never say never.
- 512 [0:39:50.8] Josh Bressers (Anchore): But what we would probably do is if something like OCM started Becoming popular, we would build the functionality into Syft.
- 513 [0:40:01.7] Josh Bressers (Anchore): That Syft would just turn OCM into Syft JSON.
- 514 [0:40:07.2] Josh Bressers (Anchore): In fact, when you use Grype, if you hand Grype a CycloneDX SBOM, Grype includes the Syft library in it.
- 515 [0:40:18.2] Josh Bressers (Anchore): So what grype will do is take your CycloneDX SBOM can it to the Syft library and the syft library will convert that into a Syft SBOM and then the Syft SBOM is what Grype scan.
- 516 [0:40:29.8] Josh Bressers (Anchore): Okay, which is.
- 517 [0:40:31.0] Josh Bressers (Anchore): But we can do that with SPDX or CycloneDx today.
- 518 [0:40:34.1] Josh Bressers (Anchore): Yeah, and so that's kind of our.
- 519 [0:40:36.5] Josh Bressers (Anchore): Our how we cheat and keep Grype a little simpler.
- 520 [0:40:40.3] Josh Bressers (Anchore): And so in the event like OCM became a demanded feature, Syft would just turn OCM into Syft SBOM and that would still go to Grype.
- 521 [0:40:49.5] Josh Bressers (Anchore): That's what happened.
- 522 [0:40:50.9] Olison Sturm: Okay, yeah, maybe it goes in.
- 523 [0:40:52.5] Olison Sturm: My next question goes in this direction because in CycloneDX there are properties and those properties can be specifically specified with reserved namespace.
- 524 [0:41:02.0] Olison Sturm: I saw Swift has their own namespace as well, like Trivy and other scanners have their own namespace.
- 525 [0:41:07.8] Olison Sturm: I requested for OCM as well, a namespace to like put all the information into CycloneDX and what I'm thinking about because

	Trivy not Grype in this case.
526	[0:41:18.3] Olison Sturm: But Trivy shows me an error message if I scan or if I put the SBOMs in it and want to have the vulnerabilities.
527	[0:41:27.0] Olison Sturm: They tell me, oh, you should use a dreary SBOMs to scan because probably because of the properties.
528	[0:41:35.1] Olison Sturm: I don't know how they really figuring out that it isn't trivy SBOM because in my case it's always a Swift SBOM.
529	[0:41:44.6] Olison Sturm: Is it in Grype as well?
530	[0:41:46.1] Olison Sturm: Is there any similarity that you like.
531	[0:41:49.9] Olison Sturm: Prefer.
532	[0:41:50.7] Olison Sturm: Prefer a Syft SBOM sent to Grype because there are some properties that you use in Grype we prefer, I would.
533	[0:41:59.2] Josh Bressers (Anchore): Say Syft generated output where if you have Syft output of cyclondx SBOM, grype will do a fine job with it.
534	[0:42:05.4] Josh Bressers (Anchore): But if trivy puts a CycloneDX SBOM, it's not as nice.
535	[0:42:10.5] Josh Bressers (Anchore): This is one of the things we're starting to work on and what happens more often than not with Grype is it just can't identify the packages and it will kind of silently fail it.
536	[0:42:23.0] Josh Bressers (Anchore): It will does great.
537	[0:42:25.2] Josh Bressers (Anchore): I can't remember.
538	[0:42:26.0] Josh Bressers (Anchore): I know Syft has a concept we call known unknowns where we can say like we found a bunch of stuff but we don't know what it is and I think, I don't remember if we're doing anything with Grype on that to just have Grype issue warnings saying like there's a bunch of stuff here.
539	[0:42:45.4] Josh Bressers (Anchore): I don't know what it is and I don't know what to do with it.
540	[0:42:47.6] Josh Bressers (Anchore): But I think today it's pretty quiet and you'll just get nothing.
541	[0:42:53.2] Olison Sturm: In my use case, I, I don't export a Syft SBOM.
542	[0:42:56.2] Olison Sturm: I, I do a CycloneDX SBOM with Syft.
543	[0:43:02.0] Olison Sturm: So would you say as well there, it doesn't matter for Grype if it is a CycloneDX.

	544	[0:43:08.6] Olison Sturm: The SBOM that's granted from Syft or from Trivy, it makes it.
	545	[0:43:14.7] Josh Bressers (Anchore): Any Grype does a CycloneDX from Syft just fine.
	546	[0:43:18.8] Josh Bressers (Anchore): It'll have.
	547	[0:43:19.4] Josh Bressers (Anchore): No, it'll.
	548	[0:43:20.0] Josh Bressers (Anchore): It'll do all the right things.
Qualitätskriterien	549	[0:43:21.1] Olison Sturm: Okay.
Tool- und Technologieeinsatz	550	[0:43:21.7] Josh Bressers (Anchore): It has all the right stuff.
	551	[0:43:23.0] Olison Sturm: Yeah.
	552	[0:43:23.6] Josh Bressers (Anchore): What, what it does with a Trivy spawn is definitely a little more unknown.
Best Practices	553	[0:43:28.8] Josh Bressers (Anchore): Sometimes it does okay.
	554	[0:43:30.0] Josh Bressers (Anchore): Sometimes it does a terrible job.
	555	[0:43:31.7] Olison Sturm: Okay, perfect.
	556	[0:43:32.6] Josh Bressers (Anchore): That's a little more than not.
	557	[0:43:34.5] Olison Sturm: Yeah.
	558	[0:43:36.5] Olison Sturm: So do you think there are better scanners in general than Trivy and Grype and only if you want to talk about it.
	559	[0:43:46.1] Olison Sturm: You don't have to.
	560	[0:43:47.1] Josh Bressers (Anchore): I mean, no, this is a good question.
	561	[0:43:48.8] Josh Bressers (Anchore): Honestly.
Tool- und Technologieeinsatz	562	[0:43:49.5] Josh Bressers (Anchore): Like, obviously if I'm putting on my Anchore hat, I'm going to be like, Grype is the best scanner.
Qualitätskriterien	563	[0:43:53.6] Olison Sturm: Yeah.
	564	[0:43:54.2] Olison Sturm: Because I want to ask you because maybe if I have your allowance, I would maybe public my publish my thesis.
	565	[0:44:01.8] Olison Sturm: So if you don't want to have it in there, your answer maybe because you like another tool or something.
Best Practices	566	[0:44:07.6] Josh Bressers (Anchore): I'm going to give you a, a weird answer.

	567	[0:44:10.2] Josh Bressers (Anchore): We'll say, okay, so I, from what I found, every scanner is good at certain things.
	568	[0:44:17.9] Josh Bressers (Anchore): They're not all good at everything.
	569	[0:44:19.6] Josh Bressers (Anchore): Like this is.
	570	[0:44:20.4] Josh Bressers (Anchore): This happens all the time at Anchore where we'll have someone be like, oh, we should compare Gype to Trivy and then we'll publish the results.
	571	[0:44:26.8] Josh Bressers (Anchore): And I'm like, that is not particularly helpful because this is, you know, it's the old lies, damned lies and statistics of like.
Qualitätskriterien Best Practices	572	[0:44:34.9] Josh Bressers (Anchore): I could make a test that would make Gype look really good compared to Trivy.
Tool- und Technologieeinsatz	573	[0:44:39.4] Josh Bressers (Anchore): I could also make test images that would make Trivy look really good compared to Gype because they are, they do things a little different, which is, I feel like one of the unfortunate realities of the software universe we exist in today.
..Herausforderungen	574	[0:44:54.7] Josh Bressers (Anchore): Ideally, in a perfect world, you should be able to run 100 scanners and get exactly the same results 100 times.
..Herausforderungen	575	[0:45:00.7] Josh Bressers (Anchore): Right.
..Herausforderungen	576	[0:45:01.5] Josh Bressers (Anchore): That's what we should be getting we don't.
..Herausforderungen	577	[0:45:03.6] Josh Bressers (Anchore): You run 100 scanners, you're going to get like 200 results because they're all freaking different.
Best Practices ..Empfehlungen	578	[0:45:08.8] Olison Sturm: Yeah.
Best Practices ..Empfehlungen	579	[0:45:09.3] Olison Sturm: Are we only talking about open source scanners or is it the same for everything?
Best Practices ..Empfehlungen	580	[0:45:16.0] Josh Bressers (Anchore): Everything.
Best Practices ..Empfehlungen	581	[0:45:17.4] Josh Bressers (Anchore): And so I mean this is one of the challenges now if we, if you just want to focus on open source scanners.
Best Practices ..Empfehlungen	582	[0:45:23.7] Josh Bressers (Anchore): The thing I found in my research is like Gype and Trivy are probably the two most maintained scanners that exist today.
Best Practices	583	[0:45:31.3] Josh Bressers (Anchore): And the challenge is not even, I'd say it's good or bad.
Best Practices	584	[0:45:34.3] Josh Bressers (Anchore): There are like, there are scanners, like

		Google has OSV scanner, you've got like the Go BOM checker, like Go BOM checker is really good at scanning go things.
	585	[0:45:42.0] Josh Bressers (Anchore): But if I said Go volume checker scan this Python application, it has no idea.
..Empfehlungen Best Practices	586	[0:45:45.9] Josh Bressers (Anchore): Does that make Go BOM checker better than Trivy Syft or Grype?
	587	[0:45:50.2] Olison Sturm: I mean, yeah, in this case, yes.
	588	[0:45:51.9] Josh Bressers (Anchore): But yeah, like Albert Einstein quote about you expect a fish to climb a tree sort of thing.
	589	[0:45:58.7] Olison Sturm: Okay, so, but in my case in OCM we I only have the binaries, maybe source code, I told you.
	590	[0:46:06.0] Olison Sturm: But I should mainly focus on something that like supports almost everything and that's then basically driv and Grype, isn't it?
	591	[0:46:14.2] Josh Bressers (Anchore): Well so I mean they try to do a lot of things.
Tool- und Technologieeinsatz	592	[0:46:17.0] Josh Bressers (Anchore): I mean so for binaries intel had their what Bin CVE bin tool I think it was called.
	593	[0:46:22.5] Josh Bressers (Anchore): Okay, that actually does a really good job of detecting binaries but like that tools just got shelved when they had their latest round of layoffs.
Best Practices	594	[0:46:31.5] Josh Bressers (Anchore): So now we're back to the situation of we have an open source tool but it's not maintained and while it might outperform Grype and Trivy, I actually don't know.
	595	[0:46:40.2] Josh Bressers (Anchore): I haven't looked at it in a long time.
	596	[0:46:41.8] Josh Bressers (Anchore): It's obviously not going to, there isn't continued maintenance.
	597	[0:46:45.1] Josh Bressers (Anchore): It's going to eventually.
	598	[0:46:46.6] Josh Bressers (Anchore): You know the, it's only down into the right for now.
	599	[0:46:49.6] Josh Bressers (Anchore): And this is the challenge with open source scanners is for whatever reason that could be a thesis of its own.
..Herausforderungen	600	[0:46:55.6] Josh Bressers (Anchore): But for whatever reason it is very difficult to keep something like an open source scanner like alive for a long period of time.
	601	[0:47:02.2] Josh Bressers (Anchore): They always seem to pop up for a little

<p>Tool- und Technologieeinsatz</p> <p>..Herausforderungen</p> <p>602</p> <p>603</p> <p>604</p> <p>605</p> <p>606</p> <p>607</p> <p>Zukunftsperspektiven</p> <p>..Zukunft der SBOM-Standards</p> <p>608</p> <p>609</p> <p>610</p> <p>611</p> <p>612</p> <p>613</p> <p>614</p> <p>615</p> <p>616</p> <p>Konkrete Anforderungen</p> <p>617</p> <p>618</p> <p>Best Practices</p> <p>619</p>	<p>while, they get some attention and then they just kind of die.</p> <p>[0:47:08.3] Olison Sturm: I see.</p> <p>[0:47:10.5] Olison Sturm: So that are running out of questions.</p> <p>[0:47:13.5] Olison Sturm: So I have only like a couple of closing questions.</p> <p>[0:47:17.2] Olison Sturm: So would you say there's in the future maybe a possibility that we have only one SBOM standard in general or do you think there's like always like many of like different formats and it will be more in the future, like for example, your Syft standard and so on.</p> <p>[0:47:39.3] Josh Bressers (Anchore): I would like to live in a world that had one SBOM standard.</p> <p>[0:47:44.5] Josh Bressers (Anchore): I don't think that will happen because I think if you look at every single industry ever, there are multiple standards because someone always thinks they can do a better job.</p> <p>[0:47:53.5] Josh Bressers (Anchore): Yeah, I'm a little surprised. SBOM only has two, to be perfectly honest with you.</p> <p>[0:47:59.4] Josh Bressers (Anchore): But I. I have a strong suspicion there will be more than two over time.</p> <p>[0:48:05.9] Josh Bressers (Anchore): Okay, this is that XKCD comic of.</p> <p>[0:48:08.3] Josh Bressers (Anchore): There are 14 standards.</p> <p>[0:48:09.9] Josh Bressers (Anchore): Like that's, oh, I'll make one that does it all.</p> <p>[0:48:12.5] Josh Bressers (Anchore): You know, now there are 15 standards.</p> <p>[0:48:14.7] Josh Bressers (Anchore): Like that is humans.</p> <p>[0:48:18.7] Olison Sturm: Okay, and another question.</p> <p>[0:48:20.9] Olison Sturm: Is there anything you can think about, like especially technical requirements that are important for me in my PoC of my CLI tool that I want to develop and go, do you have like certain requirements that you would like to give me on the way?</p> <p>[0:48:41.4] Josh Bressers (Anchore): So I would.</p> <p>[0:48:42.0] Josh Bressers (Anchore): The requirement I would ask you to maybe keep an eye on is take the output of your tool and put it into other things.</p> <p>[0:48:50.0] Josh Bressers (Anchore): You know, there's things like dependency track, you've got Trivy and Grype and I don't even know.</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	620	[0:48:56.4] Josh Bressers (Anchore): I'm sure CycloneDX has a million other tools that do a ton of other stuff.
	621	[0:48:59.8] Josh Bressers (Anchore): You know, if you got like SBOMit from the open ssf, see what happens.
Konkrete Anforderungen	622	[0:49:04.2] Josh Bressers (Anchore): I'm not saying you should necessarily support every tool, but I think the reality is like people are going to take your output and they have to do something with it.
Best Practices	623	[0:49:15.5] Josh Bressers (Anchore): Like let's say in the context of SAP, I assume it would be SAP as a customer saying, give me the SBOM for the thing I'm getting from you.
	624	[0:49:23.3] Josh Bressers (Anchore): So if you give them the SBOM and if the SBOM doesn't work with any tooling, you might as well just given them, you know, random data.
	625	[0:49:29.7] Josh Bressers (Anchore): It doesn't matter.
	626	[0:49:31.0] Olison Sturm: The goal is that the customer basically or like can generate their own SBOM based on that what we deploy to him.
	627	[0:49:40.6] Olison Sturm: So he like can like per runtime get their SBOM for the whole system that is described in all the open component descriptors.
	628	[0:49:51.4] Olison Sturm: So that's maybe the goal.
	629	[0:49:53.2] Olison Sturm: So, and as well, maybe we have, as I told you, we have like using Syft, for example, to build per artifact in one OCM component, all the SBOMs and then merging them together and maybe get their nested components.
	630	[0:50:08.8] Olison Sturm: But there's also a way that all like the artifact there in there that the maintainers of these artifacts like publish with the type SBOM.
	631	[0:50:16.7] Olison Sturm: I showed you their SBOM and then it's only using them and like aggregating them or a later use case.
	632	[0:50:24.1] Olison Sturm: So something like that.
	633	[0:50:25.4] Olison Sturm: That's what I have to figure out.
	634	[0:50:28.4] Josh Bressers (Anchore): That's pretty cool.
	635	[0:50:30.0] Olison Sturm: Yeah.
Konkrete Anforderungen	636	[0:50:33.2] Josh Bressers (Anchore): But again, make sure you can put them somewhere.
Qualitätskriterien	637	[0:50:36.4] Josh Bressers (Anchore): Right.

Kritik & Gegenargumente
Best Practices

- 638 [0:50:37.2] Olison Sturm: Get it.
- 639 [0:50:37.7] Josh Bressers (Anchore): Yeah, we actually have that.
- 640 [0:50:40.6] Josh Bressers (Anchore): We have that problem with customers where we had to build special functionality into our enterprise tool that customers were like, normally we would receive an SBOM and we'd like, just make sure it's like functionally complete and it has certain information.
- 641 [0:50:55.2] Josh Bressers (Anchore): We had customers sending us stuff that was just like, what the hell is this?
- 642 [0:50:58.8] Josh Bressers (Anchore): We can't figure this out at all.
- 643 [0:51:01.0] Josh Bressers (Anchore): No, just.
- 644 [0:51:01.5] Josh Bressers (Anchore): We want you to store this SBOM.
- 645 [0:51:03.0] Josh Bressers (Anchore): We're like, there's no information in this.
- 646 [0:51:05.4] Josh Bressers (Anchore): We don't care.
- 647 [0:51:05.9] Josh Bressers (Anchore): Just store it.
- 648 [0:51:06.7] Josh Bressers (Anchore): So we literally created a special API that was like, just take JSON in and store it in the database.
- 649 [0:51:12.5] Josh Bressers (Anchore): We don't care what it is.
- 650 [0:51:13.8] Josh Bressers (Anchore): Just store the JSON and it's like whatever, what they wanted.
- 651 [0:51:19.8] Olison Sturm: Clueless.
- 652 [0:51:20.2] Olison Sturm: Yeah.
- 653 [0:51:20.6] Olison Sturm: Okay.
- 654 [0:51:20.8] Olison Sturm: Get it.
- 655 [0:51:22.7] Olison Sturm: Yeah.
- 656 [0:51:23.7] Olison Sturm: That are all my questions.
- 657 [0:51:25.2] Olison Sturm: So is there, I don't know, anything else you want to add that we missed discussing?
- 658 [0:51:30.0] Josh Bressers (Anchore): No, this has been really fun.
- 659 [0:51:31.4] Josh Bressers (Anchore): I would say the one thing I have, if you like, I can put you in touch with like some of the Syft Grype developers and yeah, they'd be happy to answer.

- 660 [0:51:38.8] Josh Bressers (Anchore): They.
- 661 [0:51:39.1] Josh Bressers (Anchore): They could answer the technical questions that I could not.
- 662 [0:51:42.1] Olison Sturm: Okay, yeah, that would be great.
- 663 [0:51:44.1] Olison Sturm: Maybe one contact and then I can ask the three questions that I still have.
- 664 [0:51:48.2] Olison Sturm: Yeah, maybe in a text or something.
- 665 [0:51:50.3] Josh Bressers (Anchore): I'll put you in touch with.
- 666 [0:51:51.9] Olison Sturm: Yeah, I joined the select this the Slack channel of you.
- 667 [0:51:58.4] Olison Sturm: Maybe I can contact them there.
- 668 [0:52:01.9] Josh Bressers (Anchore): They.
- 669 [0:52:02.3] Olison Sturm: No, I didn't.
- 670 [0:52:03.3] Olison Sturm: Sorry.
- 671 [0:52:03.7] Olison Sturm: My fault.
- 672 [0:52:04.4] Olison Sturm: I'm not on your slack.
- 673 [0:52:06.1] Josh Bressers (Anchore): My fault we don't have a slack anymore.
- 674 [0:52:08.0] Olison Sturm: Yeah, yeah, that's.
- 675 [0:52:09.1] Olison Sturm: I was.
- 676 [0:52:09.7] Josh Bressers (Anchore): I mean, I'll just ask them.
- 677 [0:52:11.5] Olison Sturm: Yeah.
- 678 [0:52:12.0] Josh Bressers (Anchore): And I'll connect you with email, so that'll probably be easiest, but.
- 679 [0:52:16.0] Olison Sturm: Yeah, that's.
- 680 [0:52:16.6] Olison Sturm: That's great.
- 681 [0:52:18.5] Josh Bressers (Anchore): Well, this is fun.
- 682 [0:52:20.0] Olison Sturm: Yeah.
- 683 [0:52:20.9] Olison Sturm: Thanks for taking the time.

- 684 [0:52:22.8] Josh Bressers (Anchore): Will your research be public when you're done with it?
- 685 [0:52:24.4] Olison Sturm: Everything is open source and if you give me like the allowance to publicly use your transcript of this meeting.
- 686 [0:52:35.1] Olison Sturm: Of course.
- 687 [0:52:36.7] Josh Bressers (Anchore): Yeah.
- 688 [0:52:37.0] Josh Bressers (Anchore): Everything I've told you is 100% like do what you want.
- 689 [0:52:40.6] Josh Bressers (Anchore): You can, you can publish it verbatim.
- 690 [0:52:42.8] Olison Sturm: So then of course I would love to share my research results with you at the end.
- 691 [0:52:49.7] Olison Sturm: Okay.
- 692 [0:52:51.0] Olison Sturm: Yeah, I write it in German to be.
- 693 [0:52:53.0] Olison Sturm: To be honest, but I can't like translate it.
- 694 [0:52:57.5] Josh Bressers (Anchore): Yeah, that's fine.
- 695 [0:52:58.3] Josh Bressers (Anchore): I'm happy to Google translate it.
- 696 [0:52:59.9] Olison Sturm: Okay, that's cool.
- 697 [0:53:01.8] Josh Bressers (Anchore): All right.
- 698 [0:53:02.8] Josh Bressers (Anchore): All right.
- 699 [0:53:03.3] Olison Sturm: Yeah, that's all.
- 700 [0:53:04.6] Olison Sturm: And as well, what I wanted to ask you on behalf of the OCM team.
- 701 [0:53:08.4] Olison Sturm: They would like.
- 702 [0:53:09.1] Olison Sturm: Because I leaving the team in a month, to be honest.
- 703 [0:53:12.8] Olison Sturm: Yeah, they would like to collaborate with you.
- 704 [0:53:16.0] Olison Sturm: So we also opened an issue.
- 705 [0:53:18.6] Olison Sturm: Maybe we could like.
- 706 [0:53:20.1] Olison Sturm: It's low priority but maybe in the future we could contribute to your project to maybe like preserve the structure of the CycloneDX SBOM as an output of Grype, you know what I mean?

- 707 [0:53:33.9] Olison Sturm: So maybe things like that.
- 708 [0:53:35.7] Olison Sturm: So yeah, we would really appreciate it.
- 709 [0:53:38.8] Josh Bressers (Anchore): Yeah, they would love that.
- 710 [0:53:40.7] Josh Bressers (Anchore): So let me.
- 711 [0:53:41.9] Josh Bressers (Anchore): I'm gonna.
- 712 [0:53:42.5] Josh Bressers (Anchore): I'm gonna put a link in the chat here, but it's just on.
- 713 [0:53:46.7] Josh Bressers (Anchore): It's on the GitHub page for Syft.
- 714 [0:53:48.2] Josh Bressers (Anchore): It's just in the readme at the very bottom it says Syft team meetings.
- 715 [0:53:52.3] Josh Bressers (Anchore): And this is just the open source.
- 716 [0:53:54.1] Josh Bressers (Anchore): This is where we have a.
- 717 [0:53:57.0] Josh Bressers (Anchore): Excuse me.
- 718 [0:53:57.5] Josh Bressers (Anchore): Every other Thursday.
- 719 [0:53:58.6] Josh Bressers (Anchore): Is there one this week?
- 720 [0:54:00.8] Josh Bressers (Anchore): There's not one this week.
- 721 [0:54:02.1] Josh Bressers (Anchore): So it's next week.
- 722 [0:54:03.6] Olison Sturm: Okay.
- 723 [0:54:04.4] Josh Bressers (Anchore): They have just what they call the.
- 724 [0:54:06.5] Josh Bressers (Anchore): The Open Source community meeting.
- 725 [0:54:08.9] Josh Bressers (Anchore): And so your crew can totally just open issues.
- 726 [0:54:12.7] Josh Bressers (Anchore): They can come to the meeting and chat with the team.
- 727 [0:54:15.6] Josh Bressers (Anchore): They.
- 728 [0:54:16.0] Josh Bressers (Anchore): Yeah, but we love PRs.
- 729 [0:54:18.5] Josh Bressers (Anchore): I mean it's not going to be a high priority for us to do the work, but if SAP has people willing to, you know, chop wood and carry water.
- 730 [0:54:26.4] Josh Bressers (Anchore): Yeah, we'd love to accept it.

Mapping-Kompatibilität
Zukunftsperspektiven

	731	[0:54:28.0] Olison Sturm: Nice.
	732	[0:54:28.4] Olison Sturm: Yeah, maybe in my free time because sure, I leaving the team, but like a bit of open source work would be really nice.
	733	[0:54:36.4] Olison Sturm: I think maybe I have time in my free time.
Mapping-Kompatibilität	734	[0:54:40.1] Josh Bressers (Anchore): Awesome.
Zukunftsperspektiven	735	[0:54:40.9] Olison Sturm: To contribute like something like that because I have like an idea how we could implement it because I already am like got the source code in the community call like the problem like they told me or you, I don't remember.
	736	[0:54:57.2] Josh Bressers (Anchore): Yeah.
	737	[0:54:57.6] Olison Sturm: So maybe I can contribute to it.
	738	[0:54:59.6] Josh Bressers (Anchore): Cool.
	739	[0:55:00.1] Josh Bressers (Anchore): Yeah, that'd be awesome.
	740	[0:55:01.6] Olison Sturm: Yeah.
	741	[0:55:02.1] Olison Sturm: Okay.
	742	[0:55:03.1] Olison Sturm: Thanks a lot and thanks for your valuable input.
	743	[0:55:06.5] Josh Bressers (Anchore): No, this has been fun.
	744	[0:55:07.4] Josh Bressers (Anchore): And look for an email from me.
	745	[0:55:08.7] Josh Bressers (Anchore): Probably an hour or two with.
	746	[0:55:10.5] Olison Sturm: Yeah, no worries.
	747	[0:55:11.5] Josh Bressers (Anchore): The open source team.
	748	[0:55:12.5] Olison Sturm: Take your time.
	749	[0:55:13.1] Olison Sturm: Yeah.
	750	[0:55:13.5] Olison Sturm: So it can be tomorrow.
	751	[0:55:14.6] Olison Sturm: You don't have to worry.
	752	[0:55:15.6] Josh Bressers (Anchore): If I don't do it now, I'm going to forget.
	753	[0:55:17.5] Olison Sturm: Okay.
	754	[0:55:18.9] Josh Bressers (Anchore): All right, thanks.

755 [0:55:20.4] Olison Sturm: Have a nice day.

756 [0:55:21.3] Olison Sturm: Bye.

Anhang E: Kodierleitfaden (Codes + Memos)

MAXQDA 24

21.09.25

Liste der Codes	Memo	Häufigkeit
Codesystem		224
Expertise & Kontext	<p>Definition: Aussagen zur Rolle, Erfahrung und typischen Szenarien der Befragten im Bereich SBOMs/Supply Chain Security.</p> <p>Inklusion: Berufserfahrung, Verantwortlichkeiten, Nutzungskontexte.</p> <p>Exklusion: Technische Details zu OCM/CycloneDX.</p> <p>Signalwörter: „I've been working...“, „As chair...“, „In my role...“</p>	10
OCM-spezifische Strukturen	<p>Definition: Aussagen zu relevanten Konzepten und Feldern des Open Component Model (OCM), die für SBOM-Transformationen wichtig sind.</p> <p>Inklusion: Erwähnungen von Labels, Identitäten, Repository-Informationen, Component References, Metadaten oder Konventionen im OCM.</p> <p>Exklusion: Aussagen zu CycloneDX-Strukturen ohne Bezug zum OCM.</p> <p>Signalwörter: „labels“, „identity“, „repository context“, „component reference“, „OCM spec field“.</p>	14
CycloneDX-spezifische Strukturen	<p>Definition: Aussagen zu zentralen Feldern, Beziehungen und Erweiterungen im CycloneDX-Standard.</p> <p>Inklusion: Nennungen von Pflichtfeldern, Hierarchiemodellierung (dependencies, bom-links, embedded), Nutzung von properties oder Erweiterungen.</p> <p>Exklusion: Allgemeine OCM-Strukturen oder Tool-Diskussionen ohne Bezug zu CycloneDX.</p> <p>Signalwörter: „dependencies“, „bom-link“, „properties“, „mandatory fields“, „CycloneDX extension“.</p>	18
Mapping-Kompatibilität	<p>Definition: Aussagen zur Überführbarkeit von OCM-Strukturen in CycloneDX, inkl. Informationsverlust oder Workarounds.</p> <p>Inklusion: Passagen über kompatible Felder, nicht abbildbare Elemente, semantische Brüche, vorgeschlagene Lösungen.</p> <p>Exklusion: Allgemeine Beschreibungen von OCM oder CycloneDX ohne Bezug zur Übertragung.</p> <p>Signalwörter: „lossless“, „cannot represent“, „mapping problem“, „workaround“.</p>	19
Verlustfreie Konvertierung		2

MAXQDA 24

21.09.25

Hierarchische Strukturen		1
Modellierungsstrategien	Definition: Aussagen zu Optionen und Varianten, wie OCM-Komponentenhierarchien in CycloneDX dargestellt werden können. Inklusion: Beschreibungen und Vergleiche von Modellierungsansätzen (dependencies, bom-links, embedded) inkl. Vor-/Nachteile. Exklusion: Allgemeine Mapping-Probleme ohne Bezug zu Hierarchiemodellen. Signalwörter: „dependencies“, „bom-link“, „embedded“, „hierarchy modeling“.	13
Qualitätskriterien	Definition: Aussagen zu Kriterien, die eine aus OCM generierte CycloneDX-SBOM erfüllen muss, um praktisch nutzbar zu sein. Inklusion: Erwähnungen von Vollständigkeit, Eindeutigkeit, Konsistenz, Validität, Toolkompatibilität. Exklusion: Allgemeine Toolbeschreibungen ohne Bezug zu Qualitätskriterien. Signalwörter: „completeness“, „accuracy“, „valid“, „consistent“, „tool compatibility“.	14
Vollständigkeit		1
Tool- und Technologieeinsatz	Definition: Aussagen über eingesetzte oder bekannte Tools/Technologien zur Erstellung, Transformation oder Nutzung von SBOMs. Inklusion: Nennungen von Tools wie Syft, Gripe, Trivy, CycloneDX CLI, Protobom oder vergleichbare Software. Exklusion: Allgemeine Qualitätsanforderungen oder Konzepte ohne Toolbezug. Signalwörter: „Syft“, „Gripe“, „Trivy“, „CycloneDX CLI“, „Protobom“, „scanner“.	23
Middleware (Protobom)		3
Empfehlungen		2
Herausforderungen		8
SBOM-Generatoren		1
Best Practices	Definition: Aussagen zu etablierten Vorgehensweisen, Erfahrungswerten und Lessons Learned im Umgang mit OCM, CycloneDX oder SBOMs. Inklusion: Erwähnungen von Referenzprojekten, typischen Fehlern, Empfehlungen oder Community-Standards. Exklusion: Reine Toolnennungen oder theoretische Anforderungen ohne Erfahrungsbezug.	42

2

MAXQDA 24

21.09.25

	Signalwörter: „best practice“, „lessons learned“, „in our project“, „community uses“, „common mistake“.	
Kritik & Gegenargumente	Definition: Aussagen, die grundsätzliche Probleme, Risiken oder Einwände gegen ein OCM zu CycloneDX-Mapping oder gegen SBOMs allgemein formulieren. Inklusion: Hinweise auf hohen Aufwand, Standardisierungshindernisse, mangelnde Akzeptanz oder konzeptionelle Schwächen. Exklusion: Konkrete technische Mappingprobleme (gehören zu Code 4). Signalwörter: „too complex“, „not practical“, „lack of adoption“, „standardization issue“, „conceptual problem“.	8
Konzeptuelle Inkompatibilität		2
Zukunftsperspektiven	Definition: Aussagen zu erwarteten Entwicklungen, Trends oder künftigen Standards im Bereich SBOMs, OCM oder CycloneDX. Inklusion: Einschätzungen über die Durchsetzung bestimmter Formate, geplante Erweiterungen, erwartete Standardisierung. Exklusion: Aktuelle Probleme oder Anforderungen ohne Zukunftsbezug. Signalwörter: „in the future“, „expected“, „upcoming“, „planned“, „next version“.	14
Zukunft der SBOM-Standards		3
Konkrete Anforderungen	Definition: Aussagen, die konkrete Muss-Kriterien oder unverzichtbare Bedingungen für ein OCM zu CycloneDX-Mapping benennen. Inklusion: Passagen mit klaren Anforderungen an Felder, Informationen oder Toolfunktionen. Exklusion: Allgemeine Qualitätsmerkmale ohne Formulierung als Anforderung. Signalwörter: „must have“, „essential“, „required“, „unverzichtbar“, „necessary“.	26
Experteninterview Jakob Möller		0
Experteninterview Josh Bressers		0
Experteninterview Steve Springett		0

Anhang F: Summary-Tabelle

	Experteninterview Josh Bressers	Experteninterview Steve Springett	Experteninterview Jakob Möller
Dokumente und Variablen	Scanner > Experteninterview Josh Bressers	CycloneDX > Experteninterview Steve Springett	OCM > Experteninterview Jakob Möller
Expertise & Kontext	<p>Josh Bressers von Anchore ist seit 2004 im Bereich Software Supply Chain Security tätig. Er begann bei Red Hat im Security Response Team, wo er für das Tracking von Schwachstellen in Red Hat-Produkten zuständig war. Später wechselte er zu Elastic, um dort die Produktsicherheit zu verbessern und die Kontrolle über die Lieferkette zu erlangen. Heute ist er Vice President of Security bei Anchore, wo er sich um die Sicherheit der Anchore-Software kümmert, OpenSSF-Meetings leitet und die Informationssicherheit des Unternehmens verantwortet.</p> <p>Bressers sieht den Hauptanwendungsfall für die Transformation von OCM-Strukturen in standardisierte SBOM-Formate wie CycloneDX in stark regulierten Branchen. Regulierungsbehörden fordern zunehmend detaillierte Informationen darüber, wie Software erstellt wurde und woher sie stammt. Während Scanner diese Daten derzeit noch nicht optimal nutzen können, könnte die Integration von OCM-Informationen in SBOMs dazu beitragen, den</p>	<p>Steve Springett verfügt über mehr als 30 Jahre Erfahrung als Softwareentwickler, davon 20 Jahre im Bereich Softwaresicherheit und seit 2012 in der Software-Lieferkette. Seit 2008 ist er auch in der physischen Lieferkette tätig. Er ist der ursprüngliche Autor von CycloneDX und fungiert derzeit als Vorsitzender der Kernarbeitsgruppe sowie des ECMA TC54, das die Spezifikation standardisiert.</p> <p>Die Umwandlung von OCM-Strukturen in ein standardisiertes SBOM-Format wie CycloneDX ist hauptsächlich durch die Notwendigkeit der Interoperabilität motiviert. Angesichts Hunderter von Tools im CycloneDX-Tool-Ökosystem und Tausender weiterer, die die Spezifikation unterstützen, ermöglicht eine solche Konvertierung die Nutzung dieser umfangreichen Tool-Landschaft.</p> <p>CycloneDX hat sich von einer Full-Stack-Stücklisten-Spezifikation zu einer Sprache für Transparenz entwickelt. Zukünftige Versionen wie CycloneDX 1.7 und 2.0 konzentrieren sich stark auf Transparenz und weniger auf traditionelle</p>	<p>Jakob Möller begann vor sieben Jahren mit Supply Chain Security und der dynamischen Auslesung von SBOMs zur Identifizierung und Behebung von Sicherheitslücken. Seine erste konkrete Arbeit mit einem SBOM-Framework erfolgte vor fünf Jahren über das OCM, wobei er vor zwei Jahren aktiv an dessen Entwicklung mitwirkte. Aktuell ist er Lead Architect des OCM und trägt die Verantwortung für das Projekt unter der Open-Source-Schirmherrschaft von NeoNephos, wo er als Technical Advisory Council und Technical Steering Committee Member fungiert. Er ist Ansprechpartner für neue Projektinitiativen, organisiert Abstimmungen und ist als Maintainer an der Mitgestaltung der OCM-Kernspezifikation beteiligt.</p> <p>In Bezug auf die Thesis von Olison Sturm agiert Jakob Möller als beratende Instanz, da er Teil des OCM-Teams ist, das die Arbeit betreut. Er hat Einblicke in die Arbeit, nimmt jedoch keinen Einfluss auf die Kontrolle der Arbeitsergebnisse, sondern steht für Fragen zum OCM zur Verfügung, während die Arbeit</p>

Anhang F: Summary-Tabelle

149

	Nachweis für Regulierungsbehörden zu erbringen und die Komplexität der Erstellung mehrerer SBOMs für verschiedene Phasen des Software-Lebenszyklus zu reduzieren.	Stücklisten-Anwendungsfälle. Dies ermöglicht die Abbildung des gesamten Software-Lebenszyklus, von der Konzeption bis zur Zerstörung. Steve Springett sieht Anwendungsfälle für aus OCM-Komponenten generierte SBOMs in allen von OCM unterstützten Szenarien, einschließlich Richtliniendurchsetzung und Schwachstellen-Scans.	größtenteils eigenständig erfolgt. OCM zeigt Interesse daran, SBOM als Standardspeicherformat zu nutzen und OCM-Beschreiber lediglich als Laufzeitmodell zu verwenden, um bei Persistierung automatisch in Formate wie CycloneDXSBOM umzukonvertieren. Möller hinterfragt den Mehrwert für Integratoren, sollte es kein einheitliches Transparenzformat mehr geben. Er sieht CycloneDX in einer komplexen Position, da es als Konsortium von verschiedenen Firmen und Interessen getrieben wird, was zu einer ungleichmäßigen Investition in verschiedene Bereiche führen kann. Dennoch glaubt er, dass dies akzeptabel ist, da sich de jure Spezifikationen, die nicht angenommen werden, letztlich an de facto Implementierungen anpassen. Er sieht keine Schwierigkeiten in der Adoption oder darin, dass CycloneDX zu weit voraus sei.
OCM-spezifische Strukturen	OCM bietet Anweisungen für die Bereitstellung und Verschiebung von Komponenten, was die Frage aufwirft, welche Optionen für die Modellierung von Komponentenhierarchien innerhalb von OCM existieren. Aktuell erfordert die Zusammenstellung dieser Komponenten oft menschliches Eingreifen, was als Herausforderung für die zukünftige Akzeptanz von OCM gesehen wird. Die meisten Nutzer bevorzugen automatisierte Lösungen,	Im OCM können externe Identitäten frei definiert werden, indem man einen Namen als Schlüssel und einen String als Wert angibt. Typische Anwendungsfälle hierfür sind beispielsweise die Architektur (z.B. ARM64) oder das Betriebssystem (z.B. OS Linux). Zusätzlich zu diesen Identitäten gibt es Labels, die ebenfalls als Name-Wert-Paare spezifiziert werden können. Die Diskussion berührt auch die Notwendigkeit, Open-Source-Tools zu	OCM versteht sich als SBoD und beschreibt einen größeren Scope als ein klassisches SBOM (Software Bill of Materials), da es den gesamten Transfer von Softwareartefakten von einer Umgebung in die nächste abdeckt. Während ein SBOM einen Zustand einer Software zu einem Zeitpunkt X detailliert beschreibt, nimmt eine SBoD bereits existierende Softwareartefakte und schiebt sie von A nach B. OCM und SBOMs sind eng verwandt, aber nicht

Anhang F: Summary-Tabelle

150

<p>bei denen ein Tool die Arbeit eigenständig erledigt. Es wird erwartet, dass die Automatisierung von SBOMs weiter reifen wird, sodass sie in Zukunft Aufgaben übernehmen kann, die derzeit noch manuell in OCM ausgeführt werden müssen, wodurch der Prozess "magisch" und ohne menschliches Zutun ablaufen könnte.</p>	<p>verwenden, da das Open Component Model selbst quelloffen ist. Dies schränkt die Auswahl der Werkzeuge ein, da nur Open-Source-Lösungen in Betracht gezogen werden können.</p>	<p>äquivalent, da ein klassisches SBOM nicht ausreicht, eine Softwarelieferung vollständig zu beschreiben. Es besteht jedoch ein Interesse daran, die Kompatibilität mit SBOM-Ökosystemen zu gewährleisten, da die meisten Open-Source-Tools SBOMs für Scans und Analysen nutzen.</p>
	<p>Ein Delivery-Artefakt in einer Cloud- oder On-Premise-Umgebung kann verlustfrei in ein SBOM konvertiert werden, sobald es sich in der Zielumgebung befindet. Im OCM wird zwischen Artefakttypen, Ressourcen und Sourcen unterschieden, wobei Ressourcen die aktiven Lieferartefakte sind, die übertragen werden, und Sourcen der Quellcode, aus dem Ressourcen generiert werden. Ressourcen können in Komponenten wie bei CycloneDX umgewandelt werden, die dann ein SBOM enthalten. Der Access innerhalb einer Ressource beschreibt die Herkunft des Lieferartefakts, und die Kombination aus Metainformationen und Access kann gut in eine SBOM-Komponente umgewandelt werden.</p>	<p>OCM ermöglicht die Referenzierung von Beschreibern untereinander, sogenannte Komponentenreferenzen, um Softwareprodukte zu koppeln und Abhängigkeiten zu modellieren. Diese Komponentenmodellierung kann eins zu eins in CycloneDX übertragen werden, da auch</p>

		CycloneDX Referenzierungsfähigkeit besitzt. Sowohl OCM als auch CycloneDX unterstützen globale Identitäten, was eine semantisch äquivalente Struktur ermöglicht, solange diese Identitäten erhalten bleiben. Eine Komponente im OCM muss eine DNS-konforme Domain sein, um die globale Eindeutigkeit und Auflösbarkeit von Referenzen zu gewährleisten.	Felder wie Repository-Kontexte sind für den Delivery-Aspekt gedacht und sollten nicht in ein SBOM zurückgemappt werden, da sie sich nach jeder Lieferung ändern. Durch eine globale Komponentenidentität wird jede Ressourcenidentität ebenfalls global und eindeutig. Dies ermöglicht zusammen mit dem dynamischen Access den Aufbau einer statischen und immutablen SBOM-Referenz, die lokationsunabhängig funktioniert. Die OCM-Kernspezifikation ist entscheidend für die Laufzeit, und für die Erstellung neuer Komponenten gibt es eine Konstruktorspezifikation, die relevant ist, wenn SBOMs in Komponenten integriert werden.
CycloneDX-spezifische Strukturen	Josh Bressers betont, dass die Paketidentifikation die kritischste CycloneDX-Struktur für Downstream-Tools wie Gype ist. Er erklärt, dass PURLs im Allgemeinen am besten geeignet sind, aber Gype auch CPEs verwendet, wenn keine PURLs vorhanden sind. Die	Steve Springett erläutert, dass CycloneDX grundlegende Komponenten- und Serviceobjekte sowie Metadaten unterstützt, wobei er hervorhebt, dass die meisten Tools erweiterte Funktionen wie die Pedigree-Informationen einer Komponente nicht	Jakob Möller erläutert, dass die wichtigsten Artefakte Ressourcen sind, die in CycloneDX als Komponenten umgewandelt werden können und selbst wieder ein SBOM enthalten. Die Komponentenmodellierung und Referenzierung lassen sich eins zu eins in CycloneDX übertragen, da

Anhang F: Summary-Tabelle

152

Qualität der Identifikation variiert stark, wenn auf andere Methoden zurückgegriffen werden muss.	unterstützen, da diese nicht automatisiert werden können. Für nicht vorhandene Felder können CycloneDX-Eigenschaften als Schlüssel-Wert-Paare verwendet werden. CycloneDX bietet zudem die Möglichkeit, die Identität einer Komponente durch Evidenz zu untermauern, indem Methoden und Techniken zur Ableitung der Identität sowie das Vertrauen in diese Methoden dokumentiert werden.	es Referenzierungsfähigkeit für andere BOMs besitzt und sowohl OCM als auch CycloneDX eine globale Identität ermöglichen. Beim Ausliefern von Softwareartefakten kann der gesamte Baum traversiert werden, ähnlich wie bei einem SBOM, indem alle Referenzen aufgelöst und in einer abgeflachten Baumstruktur dargestellt werden. Alternativ kann man sich auf eine bestimmte Ebene beschränken und den Rest nur referenzieren.
Bressers weist darauf hin, dass SBOMs derzeit gut für statische Komponenten funktionieren, aber ihre Fähigkeiten nachlassen, sobald sich Dinge bewegen oder komplexere Aspekte wie Bereitstellungsanweisungen ins Spiel kommen. Er erwähnt, dass CycloneDX die Darstellung von Abhängigkeitsbäumen und verschachtelten Komponenten unterstützt, was für die Modellierung von Komponentenhierarchien nützlich ist. Für die vollständige Erfüllung der Anforderungen von Tools wie Gype ist die Erfassung umfassender Paketinformationen unerlässlich.	Die Spezifikation ermöglicht verschachtelte Komponenten und das Verknüpfen von BOMs über externe Referenzen, was die Modellierung komplexer Strukturen wie Hardware, Betriebssysteme und Anwendungen erlaubt. CycloneDX unterstützt Envelope-Signaturen, wobei eine Konvertierung zwischen Serialisierungsformaten die Signatur ungültig macht. Eine Herausforderung besteht im Verständnis des Unterschieds zwischen verschachtelten Komponenten (Assemblies) und Abhängigkeiten, was oft zu Problemen bei Tools wie Gype und Trivy führt.	CycloneDX SBOMs bieten äquivalente Strukturen, die über diverse Identifikationsstrukturen eine global eindeutige Identität zuweisen. Im Gegensatz zu SBoDs, die ein transientes Verhalten in einer Zielumgebung beschreiben, ist ein SBOM ein statisches Artefakt, das den Status Quo eines Softwareartefakts und dessen unveränderliche Abhängigkeiten darstellt. Möller betont, dass es bereits zwei etablierte Standards, CycloneDX und SPDX, gibt und viele Tools diese nutzen werden. Das Ziel ist nicht, diese zu ersetzen, sondern zu ergänzen und zu integrieren, wobei CycloneDX explizit Erweiterungsmöglichkeiten in seinen Spezifikationen bietet.
	Ein weiteres komplexes Thema ist die Formulierung, für die ein verbindlicher Leitfaden erstellt wird. Zudem gibt es terminologische Verwirrung in verschiedenen Branchen, insbesondere im Bereich KI, wo Begriffe wie "Basismodelle" oder "Grundlagenmodelle" für bereits existierende	

Anhang F: Summary-Tabelle

153

		Konzepte wie "Pedigree" verwendet werden. CycloneDX nutzt SPDX-Lizenz-IDs und -Ausdrücke, erlaubt aber auch die Angabe nicht aufgelöster Lizenznamen, um Beschaffungsprozesse zu integrieren. Die Version 2.0 von CycloneDX bietet neben dem dateibasierten Format auch ein API-Format, das als Grundlage für die Transparency Exchange API dient und die Normalisierung verschiedener BOM-Formate ermöglicht.	
Mapping-Kompatibilität	<p>Josh Bressers weist darauf hin, dass es derzeit keine Systeme gibt, die OCM-Dokumente verarbeiten können, im Gegensatz zu SBOMs, die ein etablierter Standard sind und über zahlreiche Parsing- und Verarbeitungstools verfügen. Er befürchtet, dass OCM-Dokumente ohne diese Infrastruktur auf Unverständnis stoßen würden. Olison Sturm fragt nach der Möglichkeit, OCM-Konzepte wie hierarchische Strukturen, Ressourcen und Quellen in CycloneDX abzubilden. Bressers hält dies für schwierig, da SBOMs, insbesondere CycloneDX, zwar viele Typen unterstützen, aber primär für statische Objekte konzipiert sind und ihre Fähigkeiten bei dynamischen Aspekten wie Bereitstellungsanweisungen, die OCM bietet, an ihre Grenzen stoßen.</p> <p>Bressers äußert Bedenken hinsichtlich eines direkten OCM-zu-CycloneDX-Mappings, da dabei ein Großteil der "Richness" der OCM-Daten verloren</p>	<p>Steve Springett ist der Ansicht, dass die meisten OCM-Konzepte ohne Datenverlust direkt in CycloneDX abgebildet werden können. Sollte ein Feld in CycloneDX nicht nativ existieren, können die CycloneDX-Eigenschaften als Schlüssel-Wert-Paare verwendet werden. Externe Identitäten und Labels in OCM, die frei spezifizierbar sind (z.B. Architektur, OS), würden wahrscheinlich als Eigenschaften in CycloneDX abgebildet werden.</p> <p>Die hierarchische Struktur in OCM lässt sich in CycloneDX durch verschachtelte Komponenten und externe Referenzen (BOM-Links) abbilden. Dies ermöglicht die Verknüpfung verschiedener BOMs, beispielsweise für Hardware, Betriebssystem, Konfiguration und Anwendungsbibliotheken. Beide Ansätze – externe Identitäten und verschachtelte Komponenten – können kombiniert werden, um</p>	<p>Jakob Möller betont das Interesse, nah am SBOM-Konzept zu bleiben, um Kompatibilität mit bestehenden Ökosystemen zu gewährleisten, auch wenn der Scope der SBoD größer ist. Eine verlustfreie Konvertierung zwischen Formaten ist möglich, solange man in einer Umgebung bleibt. Die wichtigsten Artefakte sind Ressourcen, die in CycloneDX-Komponenten umgemappt werden können, welche wiederum ein SBOM enthalten. Die Kombination aus Metainformationen der Ressource und ihrem Ursprung (Access) lässt sich gut in eine SBOM-Komponente überführen.</p> <p>Die Komponentenmodellierung und Referenzierung im OCM kann eins zu eins in CycloneDX übertragen werden, da beide über Referenzierungsfähigkeiten und globale Identitäten verfügen. Solange diese globalen Identitäten erhalten bleiben, ist eine semantisch äquivalente Struktur gewährleistet. Eine verlustfreie Auflösung</p>

Anhang F: Summary-Tabelle

154

Verlustfreie Konvertierung	Steve Springett äußert die Ansicht, dass es bei der Konvertierung von OCM zu CycloneDX zu keinem Datenverlust kommen sollte, da alle Felder entweder nativ abgebildet	Durch die Annahme einer globalen Komponentenidentität wird jede Ressourcenidentität ebenfalls global. Dies ermöglicht zusammen mit dem dynamischen Access den Aufbau einer statischen, immutablen SBOM-Referenz, die lokationsunabhängig funktioniert. Bei der Neuanlage von Komponenten über eine Konstruktorspezifikation ist es wichtig zu definieren, wie ein SBOM in diese Komponente integriert wird.

		werden können oder eine entsprechende Taxonomie für nicht-native Elemente entwickelt werden müsste. Olson Sturm fragt nach, ob dies bedeutet, dass immer ein Feld in CycloneDX zugeordnet werden kann.	
Hierarchische Strukturen		CycloneDX ermöglicht die Verschachtelung von Komponenten, sodass eine BOM andere Komponenten referenzieren kann. Durch externe Referenzen und die Verwendung von BOM-Links ist es möglich, die Stückliste einer Komponente über deren SerialNumber zu verknüpfen. Dies erlaubt die Spezifikation der BOM und des Lebenszyklus für verschiedene Operationen, wie Hardware, Betriebssystem, Konfiguration, Anwendung und Anwendungsbibliotheken, die alle miteinander verknüpft werden können.	
Modellierungsstrategien	<p>Josh Bressers erklärt, dass CycloneDX Abhängigkeitsbäume und verschachtelte Komponenten darstellen kann und auch externe Dokumente referenzieren kann. Er weist jedoch darauf hin, dass es an Tools mangelt, die externe Referenzen über mehrere SBOMs hinweg korrekt zusammenführen, da die meisten Tools diese als separate Dokumente behandeln.</p> <p>Josh Bressers merkt an, dass gängige Tools wie Gype und Trivy bei SBOMs mit verschachtelten Komponentenstrukturen die Hierarchie häufig nicht</p>	<p>Die kritischsten CycloneDX-Strukturen hängen stark vom Anwendungsfall ab, wobei das ursprüngliche Ziel von CycloneDX die einfache Automatisierbarkeit war, nicht die Schaffung eines perfekten Modells. Die grundlegenden Primitive umfassen das Metadaten-Segment, Komponenten und Dienste, die 95 % der Modellierungsanforderungen abdecken können. Fortgeschrittenere Anwendungsfälle, wie z.B. die Pedigree-Informationen einer Komponente, werden von den meisten Tools nicht unterstützt, da sie nicht automatisierbar sind.</p>	<p>Im OCM besteht die Möglichkeit, Beschreiber untereinander zu referenzieren, was zur Kopplung von Softwareprodukten und Modellierung von Abhängigkeiten genutzt wird. Diese sogenannten Component Referenzen, bei denen eine Komponente eine andere referenziert, die wiederum Ressourcen enthalten kann, lassen sich direkt in CycloneDX übertragen, da auch dort Referenzierungsfähigkeiten für andere BOMs existieren. Die globale Identität ist sowohl in OCM als auch in CycloneDX spezifiziert, wodurch bei deren Erhalt eine</p>

Anhang F: Summary-Tabelle

156

<p>beibehalten, sie flachen die Komponenten ab oder verstehen nur die oberste Ebene. Dieses Verhalten sei ein bekanntes Problem, das aus inkonsistenter Dokumentation verschachtelter Pakete in verschiedenen Ökosystemen resultiere. Syft versuche zwar, diese Informationen zu erfassen, sei bei Konvertierungen jedoch oft gezwungen, sie wieder zu verwerfen.</p>	<p>Es ist möglich, sowohl externe Identitäten als auch verschachtelte Komponenten innerhalb eines SBOM zu verwenden, insbesondere bei Unternehmenssoftware oder komplexen Systemen wie MRT-Geräten, die mehrere Software-Stacks umfassen. Bezuglich der Signierung von Artefakten unterstützt CycloneDX Envelope-Signaturen, jedoch ist eine direkte Konvertierung von Signaturen zwischen verschiedenen Serialisierungsformaten nicht praktikabel, da dies die Signatur ungültig machen würde.</p>	<p>semantisch äquivalente Struktur gewährleistet wird.</p> <p>Bei der Implementierung solcher Referenzen können Bäume asynchron oder synchron betrachtet werden, entweder durch vollständiges Traversieren oder durch die Behandlung von Referenzen als gegeben. Bei der Auslieferung von Softwareartefakten wird meist der komplette Baum traversiert, was auch bei SBOMs möglich ist, indem alle Referenzen aufgelöst und in einem abgeflachten Baum dargestellt werden. Alternativ kann man nur eine bestimmte Ebene betrachten und den Rest referenziert lassen, besonders wenn diese bereits im System vorhanden sind. Der Wert und die Umsetzung hängen stark vom jeweiligen Anwendungsfall ab; beispielsweise reicht bei der Identifizierung von Verwundbarkeiten oft eine Analyse des obersten Baumes aus, um vorhandene Scan-Ergebnisse für Referenzen zu nutzen.</p>
<p>Obwohl das Problem bekannt ist und behoben werden soll, hat es derzeit keine hohe Priorität, da die Nachfrage von Kunden und Auditoren eher auf die Identifizierung von Schwachstellen als auf detaillierte Abhängigkeitsbäume abzielt. Josh Bressers hofft, dass VEX in Zukunft dazu beitragen wird, dieses Problem zu lösen, da es mehr Aufmerksamkeit erhält.</p>	<p>Ein häufiges Missverständnis besteht in der Unterscheidung zwischen verschachtelten Komponenten und Abhängigkeiten in CycloneDX, was zu Problemen bei Scannern wie Gype und Trivy führen kann, die die verschachtelten Strukturen nicht korrekt beibehalten. Eine direkte Abbildung von OCM auf CycloneDX wird gegenüber der Verwendung einer Zwischenabstraktion wie Protobom bevorzugt, da Protobom nur gemeinsame Elemente unterstützt und oft hinter den aktuellen Spezifikationen von CycloneDX zurückbleibt.</p>	<p>Die Aggregationsproblematik von SBOMs bleibt bestehen, unabhängig davon, wo der Scan durchgeführt oder das SBOM aggregiert wird. Komponentenbeschreiber sind keine vollständig aufgelösten Bäume, weshalb für eine vollständige Baumtraversierung immer eine komplett Auflösung erforderlich ist, sei es zur Build- oder zur Delivery-Zeit. Neben der OCM-Kerns-Spezifikation gibt es die</p>

			Konstruktorspezifikation, eine Sonderform des Beschreibers, die relevant ist, wenn neue Komponenten angelegt werden, beispielsweise wie ein SBOM in eine Komponente integriert wird. Für die Aggregation bereits erstellter und transportierter Komponenten ist dies jedoch weniger relevant.
Qualitätskriterien	<p>Josh Bressers von Anchore betont, dass er sich nicht für die "Qualität" von SBOMs interessiert, da dies kein aussagekräftiger Begriff sei. Stattdessen konzentriert er sich darauf, Dokumente zu erstellen, die spezifische Probleme lösen können, insbesondere im Hinblick auf Schwachstellen und Attestierung. Er erklärt, dass die Hauptanwendungsfälle von SBOMs darin bestehen, Schwachstellen zu identifizieren (z.B. mit Grype) und das Vorhandensein bestimmter Pakete (wie Log4j) in einer Umgebung zu überprüfen.</p> <p>Für die Kompatibilität mit Tools wie Grype ist es entscheidend, dass das CycloneDX SBOM die Paketinformationen korrekt erfasst. Bressers merkt an, dass derzeit niemand eine definitive Antwort darauf hat, was genau mit einem SBOM zu tun ist, da Vorschriften lediglich die Erstellung fordern, aber nicht die Nutzung spezifizieren. Er sieht die Einhaltung der CycloneDX-Spezifikation als ersten Schritt, gefolgt von der tatsächlichen Nutzung durch Anwender.</p> <p>Hinsichtlich der</p>	<p>Steve Springett betont, dass essenzielle Qualitätskriterien für SBOMs über die bloßen Mindestelemente hinausgehen. Er plädiert für die Nutzung von Evidenz, um die Komponentenidentität zu untermauern, und verweist auf den Qualitäts-Spinnengraphen im CycloneDX Authoritative Guide to SBOM, der Methoden, Techniken und deren Vertrauenswürdigkeit beleuchtet. Weitere Qualitätsaspekte leiten sich vom SCVS ab, insbesondere dessen BOM-Maturitätsmodell, das die Schwierigkeit der Datenerfassung basierend auf Tool-Verfügbarkeit und menschlichem Eingreifen bewertet.</p> <p>Das SCVS ermöglicht es verschiedenen Stakeholdern, wie Jurist:innen für Open-Source-Lizenziierung oder M&A, Anwendungssicherheits- oder Infrastruktursicherheitsexpert:innen, eigene Profile zu erstellen. Diese Profile gewichten die Datentypen, die für spezifische Analysen relevant sind, da Qualität letztlich im Auge des Betrachters liegt. Eine Untersuchung von SCA-</p>	<p>Jakob Möller betont, dass eine verlustfreie Konvertierung auch mit nicht vollständig rekursiv aufgelösten Bäumen möglich ist, da sowohl CycloneDX als auch OCM Gegenreferenzierungen über Metadaten wie Labels oder Annotationen erlauben. Für ihn ist das wichtigste Qualitätskriterium bei der Generierung einer SBOM aus einer SBoD die Verwendbarkeit des resultierenden SBOM/OCM-Artefakts. Dies bedeutet, dass der konvertierte Komponentenbeschreiber von anderen Tools, die bereits mit SBOMs umgehen können, nutzbar sein muss, wobei etablierte Konventionen wie die Sicherstellung von Globalität zu beachten sind.</p> <p>Ein weiteres entscheidendes Kriterium ist die vollständige Rückreferenzierbarkeit des SBOMs, um Scan-Resultate sinnvoll den ursprünglichen OCM-Ressourcen zuordnen zu können und somit einen echten Mehrwert zu schaffen. Dies ermöglicht die semantische Verbindung zwischen einer Ressource und ihren Verwundbarkeiten. Möller</p>

Anhang F: Summary-Tabelle

158

	Kompatibilität von SBOMs verschiedener Generatoren mit Gype erklärt Bressers, dass Gype gut mit CycloneDX SBOMs umgeht, die von Syft generiert wurden, da diese die notwendigen Informationen enthalten. Bei SBOMs von Trivy ist die Leistung jedoch uneinheitlicher, da Gype möglicherweise Pakete nicht identifizieren kann und dann stillschweigend fehlschlägt. Bressers betont, dass jeder Scanner seine Stärken hat und Vergleiche oft irreführend sein können, da unterschiedliche Scanner unterschiedliche Ergebnisse liefern, anstatt der idealen Situation, in der alle Scanner identische Ergebnisse liefern würden.	und Container-Analyse-Tools zeigte, dass die meisten, einschließlich Open-Source-Tools, bei der genauen Identifizierung von Komponenten und deren Metadaten versagten. Die Tools machten Fehler auf unterschiedliche Weise, was die Herausforderung bei der Sicherstellung der SBOM-Qualität unterstreicht.	fasst seine Kriterien in Rückverfolgbarkeit nach der Konvertierung und semantische Vollständigkeit sowie Sinnhaftigkeit im SBOM zusammen, wobei alle Felder korrekt übertragen und von anderen Tools auslesbar sein müssen, ohne eine neue Zwischenspezifikation zu schaffen.
Vollständigkeit	Josh Bressers von Anchore stellt fest, dass die meisten Anwendungsfälle, die sie beobachten, stark auf Schwachstellen und Attestierungen fokussiert sind. Dabei versteht er unter Attestierung nicht digitale Signaturen, da diese heute niemand korrekt umsetzen kann, sondern lediglich die Bestätigung, dass eine Datei eine bestimmte	Nicht-funktionale Anforderungen umfassen die Lizenzierung von Hilfsframeworks unter Apache, um eine Weiterverbreitung zu gewährleisten. Zudem ist für Unternehmen die langfristige Verwaltbarkeit von Projekten wichtig, weshalb Sekundärkriterien wie die Maintainability und Popularität von Abhängigkeiten im Open-Source-Kontext eine Rolle spielen. Der Prototyp sollte die vollständige Rückführbarkeit der generierten SBOMs demonstrieren, idealerweise sowohl als Code-Abhängigkeit als auch als Tooling, beispielsweise in Form eines Command Line Interfaces, um die Konvertierung als kleine Bibliothek nutzen zu können.	

Anhang F: Summary-Tabelle

159

		Prüfsumme aufweist. Wenn die SBOM eine andere Prüfsumme angibt als die tatsächliche Datei, deutet dies auf ein Problem hin, und genau dafür wird es laut Bressers heute genutzt.	
Tool- und Technologieeinsatz	<p>Josh Bressers von Anchore nutzt hauptsächlich Syft zur Generierung und Transformation von SBOMs, das ein eigenes internes Format verwendet, um Datenverluste beim Konvertieren zwischen SPDX und CycloneDX zu vermeiden. Dieses Syft-SBOM-Format ist nicht für die externe Nutzung gedacht, sondern dient als Zwischenschicht, um die Ausgabe in verschiedene Standardformate zu ermöglichen. Olson Sturm merkt an, dass dies einer eigenen Abbildung von CycloneDX- und SPDX-Feldern entspricht, da Tools wie Protobom zu Informationsverlusten führen können. Bressers bestätigt, dass Protobom ein ähnliches Konzept verfolgt, aber er ist sich unsicher, ob es noch aktiv gepflegt wird. Er empfiehlt, sich auf ein SBOM-Format wie CycloneDX zu konzentrieren, da der Bereich der SBOM-Transformationen komplex ist.</p> <p>Für die Identifizierung von Paketen und Schwachstellen verwendet Anchore PURLs als bevorzugte Methode, greift aber bei deren Fehlen auf CPEs zurück oder versucht, die Pakete anderweitig zu identifizieren. Die Schwachstelleninformationen werden aus einer</p>	<p>Steve Springett nennt den CycloneDX CLI und dessen Web-Version, das CycloneDX Web Tool, welches auf WASM basiert und schnell im Browser läuft. Er erwähnt auch das SBOM-Dienstprogramm der CycloneDX-Community, das ursprünglich von IBM beigesteuert wurde. Außerhalb von CycloneDX kennt er nur Protobom, das seiner Meinung nach keine gute Arbeit leistet.</p> <p>Protobom verwendet SPDX-Lizenzen-IDs, die streng validiert werden, und SPDX-Ausdrücke, die nicht validiert sind, aber mit SPDX-Tools überprüft werden können. Es nutzt lediglich den Namen einer Lizenz. Einige Tools wie ScanCode und ORT leisten gute Arbeit bei der Lizenzierung, aber die meisten Tools, einschließlich der CycloneDX-Tools, verlassen sich auf die im Manifest deklarierte Lizenz. Steve Springett würde für die Erstellung eines CycloneDX SBOMs aus einem OCI-Image eher kommerzielle Tools empfehlen, da deren Datenteams eine bessere Arbeit bei der Lizenzierung, auch kommerzieller Lizenzen, leisten.</p> <p>Er berichtet von einem Vergleichstest von SCA- und Container-Analyse-Tools, bei dem die meisten</p>	<p>Die meisten existierenden Open-Source-Tools interagieren mit SBOMs, indem sie deren Momentaufnahme für Scans und Analysen nutzen. Bei der Auswahl von Tools zur Erstellung, Zusammenführung und Transformation von SBOMs empfiehlt es sich, auf das Open-Source-Ökosystem zurückzugreifen. Spezifikationen wie CycloneDX bieten etablierte Referenz-Tools und Community-Tools für Konvertierungen, wie beispielsweise Protobom. Da die Spezifikationen offen sind, ist es auch einfach, eigene Tools zu entwickeln, insbesondere für kundenspezifische Anforderungen, während für Standardaufgaben wie Security-Scanner etablierte Lösungen wie Trivy bevorzugt werden sollten.</p> <p>Für die Identifizierung von Schwachstellen ist es oft notwendig, Vulnerability-Datenbanken zu nutzen, die Referenzen auf bereits gescannte Komponenten enthalten, um den Scan-Aufwand zu optimieren. Ein signifikanter Mehrwert entsteht, wenn SBOMs nicht nur für Scans genutzt, sondern auch vollständig rückreferenzierbar sind, um Scan-Ergebnisse den ursprünglichen Ressourcen zuordnen zu können. Dies ermöglicht</p>

Anhang F: Summary-Tabelle

160

<p>täglich aktualisierten, quelloffenen Datenbank namens GrypeDB bezogen, die Daten von verschiedenen Quellen wie GitHub, Red Hat, SUSE und NVD aggregiert und anreichert. Bressers weist darauf hin, dass die NVD-Daten oft unzureichend sind und eine zusätzliche Anreicherung erfordern.</p>	<p>Tools, einschließlich aller Open-Source-Tools, bei der genauen Identifizierung von Komponenten und deren Metadaten versagten. Steve Springett kann kein Open-Source-Tool empfehlen, da alle auf unterschiedliche Weise Fehler machen.</p>	<p>eine semantische Verbindung zwischen den identifizierten Schwachstellen und den betroffenen Ressourcen. Wichtige Kriterien für den Einsatz von SBOMs sind daher die Rückverfolgbarkeit nach der Konvertierung und die semantische Vollständigkeit sowie Sinnhaftigkeit der Daten im SBOM, um die Auslesbarkeit durch andere Tools zu gewährleisten.</p>
<p>Bressers äußert Bedenken hinsichtlich der Fähigkeit von SBOMs, komplexe Szenarien wie Quellcode, Build-Zeit-Informationen oder dynamische Bereitstellungen abzubilden. Er betont, dass SBOMs derzeit am besten für die Analyse statischer Artefakte geeignet sind und ihre Fähigkeiten bei sich bewegenden Komponenten an ihre Grenzen stoßen. Ein weiteres Problem ist das Fehlen von Tools, die externe Referenzen in einer großen Anzahl von SBOMs korrekt zusammenführen können; die meisten Tools behandeln sie als separate Dokumente.</p>	<p>Bezüglich der Verwendung einer Zwischenabstraktion wie Protobom rät Steve Springett davon ab und empfiehlt eine direkte Abbildung. Protobom unterstützt nur die gemeinsamen Elemente der Spezifikationen und fehlt bei der Unterstützung neuerer Funktionen von CycloneDX hinterher. Dies habe zu der falschen Annahme geführt, dass CycloneDX bestimmte Dinge nicht unterstützt, obwohl dies seit Jahren der Fall sei. Eine direkte Abbildung sei daher die bessere Wahl.</p>	<p>OCM als Public-Domain-Projekt setzt voraus, dass alle verwendeten Tools und Frameworks ebenfalls unter einer kompatiblen Open-Source-Lizenz, wie der Apache-Lizenz, stehen. Dies stellt sicher, dass die entwickelten Hilfsframeworks und Tools innerhalb des OCM-Kontextes weiterverbreitet und genutzt werden können.</p>
<p>Obwohl Grype SBOMs mit verschachtelten Komponentenstrukturen verarbeiten kann, werden diese Strukturen beim Export in CycloneDX oft abgeflacht. Dies liegt daran, dass die Dokumentation verschachtelter Pakete nicht standardisiert ist und viele Ökosysteme keine detaillierten Abhängigkeitsbeziehungen liefern. Anchore ist sich dieses Problems bewusst und möchte es beheben, aber die Priorität ist derzeit gering, da die Nachfrage</p>		

	<p>von Kunden und Auditoren eher auf der Identifizierung von Schwachstellen als auf detaillierten Abhängigkeitsbäumen liegt. Bressers erwartet, dass VEX hier in Zukunft Abhilfe schaffen könnte.</p> <p>Hinsichtlich der Effektivität von Scannern wie Gype und Trivy erklärt Bressers, dass jeder Scanner seine Stärken und Schwächen hat und es schwierig ist, einen "besten" Scanner zu benennen. Er kritisiert, dass verschiedene Scanner oft unterschiedliche Ergebnisse liefern, selbst bei denselben Eingaben. Gype und Trivy gehören zu den am besten gepflegten Open-Source-Scannern, aber spezialisierte Tools wie der Go BOM Checker sind nur für bestimmte Ökosysteme effektiv. Bressers hebt hervor, dass die Wartung von Open-Source-Scannern eine große Herausforderung darstellt, da viele Projekte nach anfänglicher Aufmerksamkeit wieder eingestellt werden, wie das Beispiel des Bin CVE-Tools von Intel zeigt.</p>	
Middleware (Protobom)	<p>Josh Bressers ist mit Protobom vertraut und vergleicht es mit Syft, fragt sich aber, ob es noch aktiv ist, da er keine aktuellen Informationen dazu hat. Bressers hält fest, dass ein direktes Mapping von OCM auf CycloneDX dem Einsatz von Protobom vorzuziehen ist: Protobom kann zwar bestimmte Transformationen leisten, erreicht jedoch nicht die Qualität und Transparenz eines eigenständig definierten, formatfokussierten</p>	<p>Steve Springett weist darauf hin, dass es neben dem SBOM-Dienstprogramm innerhalb der CycloneDX-Community, das ursprünglich von IBM beigesteuert wurde, nur Protobuf gibt, welches jedoch keine gute Leistung erbringt. Er würde eine direkte Zuordnung von OCM zu CycloneDX bevorzugen, anstatt einen Zwischenschritt über Protobom zu gehen. Protobom unterstützt nur die gemeinsamen</p>

Anhang F: Summary-Tabelle

162

	Mappings und erhöht tendenziell die Komplexität.	Elemente der Spezifikationen und hinkt bei der Unterstützung neuerer Funktionen von CycloneDX hinterher, was zu Missverständnissen führen kann.
Empfehlungen	Für die Generierung von SBOMs empfiehlt Josh Bressers zunächst Syft und dann CycloneDX-Tools. Bei Open-Source-Scannern sind Grype und Trivy die am besten gepflegten. Spezialisierte Scanner wie der Google OSV-Scanner oder der Go BOM Checker sind für spezifische Sprachen sehr gut, aber nicht universell einsetzbar, was ihre Überlegenheit gegenüber Allzweck-Scannern wie Trivy oder Syft infrage stellt.	
Herausforderungen	Josh Bressers von Anchore beschreibt die Herausforderungen bei SBOM-Formaten, da eine Konvertierung zwischen SPDX und CycloneDX ohne Datenverlust nicht möglich ist, was zur Entwicklung eines dritten Formats durch Anchore führte. Er weist darauf hin, dass es an Tools mangelt, die externe Referenzen in SBOMs korrekt zusammenführen, und dass die aktuelle Nutzung von SBOMs stark auf Compliance und die Behebung kritischer Schwachstellen ausgerichtet ist, anstatt auf detaillierte Abhängigkeitsbäume. Ein weiteres Problem ist die Inkonsistenz von Scanner-Ergebnissen; idealerweise sollten 100 Scanner 100 gleiche Ergebnisse liefern, was aber nicht der Fall ist. Zudem ist es schwierig, Open-Source-Scanner langfristig zu pflegen, da	Steve Springett weist darauf hin, dass es in der CycloneDX-Community ein von IBM beigesteuertes SBOM-Utility gibt. Außerhalb von CycloneDX kennt er vor allem Protobuf-basierte Ansätze, die seiner Einschätzung nach jedoch qualitativ nicht besonders gut sind.

Anhang F: Summary-Tabelle

163

	sie oft nach kurzer Zeit wieder verschwinden. Die Qualität der Paketidentifikation variiert stark, und die zugrunde liegende GypeDB, die Daten aus verschiedenen Quellen wie GitHub, Red Hat und NVD bezieht, ist komplex und unübersichtlich.		
SBOM-Generatoren	Josh Bressers (Anchore) nutzt überwiegend Syft und betont, dass das Tool zwischen SBOM-Formaten konvertieren kann		
Best Practices	<p>Josh Bressers von Anchore betont, dass es derzeit keine Systeme gibt, die OCM-Dokumente verarbeiten können, während SBOMs als Industriestandard etabliert sind und über zahlreiche Parsing- und Verarbeitungstools verfügen. Er empfiehlt, sich auf ein SBOM-Format wie CycloneDX zu konzentrieren und OCM-Daten dorthin abzubilden, da dies den Prozess erheblich vereinfacht. Für die Generierung von CycloneDX-SBOMs schlägt er Tools wie Syft oder andere CycloneDX-spezifische Tools vor.</p> <p>CycloneDX unterstützt die Darstellung von Abhängigkeitsbäumen und kann externe Dokumente referenzieren, was für die Abbildung komplexer Softwarestrukturen nützlich ist. Eine Herausforderung besteht jedoch darin, dass die meisten Tools externe Referenzen in SBOMs nicht korrekt zusammenführen, sondern als separate Dokumente behandeln. Bressers legt Wert darauf, dass ein Dokument ein spezifisches</p>	<p>Steve Springett betont, dass CycloneDX nicht darauf abzielt, ein perfektes Modell zu schaffen, sondern etwas, das leicht automatisierbar ist, mit wenigen Primitiven, die für eine Vielzahl von Anwendungsfällen genutzt werden können, wie z.B. Hardware, Dienstleistungen und SBOMs. Er empfiehlt mindestens den Metadatenbereich, Komponenten und Dienste der CycloneDX-Spezifikation zu nutzen, da fortgeschrittene Anwendungsfälle wie die Herkunft einer Komponente von den meisten Tools nicht unterstützt werden, da sie nicht automatisiert werden können. Bei fehlenden Feldern können CycloneDX-Eigenschaften als Schlüssel-Wert-Paare verwendet werden, und verschachtelte Komponenten sowie externe Referenzen über BOM-Links sind möglich.</p> <p>Die Qualität eines SBOMs liegt im Auge des Betrachters, wobei Springett die Verwendung von Evidenz zur Untermauerung der</p>	<p>Jakob Möller betont die Wichtigkeit global eindeutiger und rückreferenzierbarer Komponentenidentitäten, die DNS-konform sein sollten, um Referenzen auf andere Komponenten sicher auflösen zu können. Er schlägt vor, etablierte Konventionen zu nutzen, um Globalität sicherzustellen, und verweist auf äquivalente Strukturen in CycloneDX SBOMs, die eine solche Identitätszuweisung ermöglichen. Diese Vorgehensweise könnte den Aufbau eines Scannernetzwerks unterstützen.</p> <p>Für die Generierung von SBOMs gibt es zwei Ansätze: zur Build Time oder zur Delivery Time. Die Build Time-Generierung, beispielsweise aus einem CI-System, erstellt einen statischen Snapshot, der jedoch bei nachträglichen Scan-Ergebnissen dynamisch verknüpft werden müsste. Alternativ kann ein Konsument zur Delivery Time eine Live-Analyse durchführen. Obwohl OCM bereits das Anhängen von SBOMs</p>

Anhang F: Summary-Tabelle

164

<p>Problem lösen kann, insbesondere im Hinblick auf Schwachstellen und Attestierungen, wobei er Attestierung als die Überprüfung von Checksummen versteht.</p>	<p>Komponentenidentität befürwortet und auf den Qualitätsspinngraphen im CycloneDX Authoritative Guide to SBOM verweist. Dieser Guide und das SCVS-Reifegradmodell für BOMs, das den Schwierigkeitsgrad der Datenerfassung basierend auf Tool-Verfügbarkeit und menschlichem Eingreifen bewertet, bilden die Grundlage für die Qualitätsbewertung.</p>	<p>über einen spezifizierten Artefakttyp erlaubt, geht dabei die dynamische Scan-Möglichkeit verloren, und das SBOM müsste bei jeder Anpassung des Komponentenbaums manuell aktualisiert werden.</p>
<p>Die Hauptanwendungsfälle für SBOMs sind die Identifizierung von Schwachstellen, wie sie von Tools wie Gype durchgeführt wird, und die Suche nach spezifischen Komponenten wie Log4j in großen Datenbeständen. Für Gype ist die Erfassung von Paketinformationen entscheidend. Bressers weist darauf hin, dass es keine einheitliche Antwort darauf gibt, was mit den SBOM-Outputs geschehen soll, da viele Unternehmen SBOMs primär zur Erfüllung regulatorischer Anforderungen sammeln, ohne eine klare Strategie für deren Nutzung zu haben.</p>	<p>Verschiedene Stakeholder, wie Rechtsteams oder Sicherheitsexperten, können basierend auf ihren spezifischen Bedürfnissen Profile erstellen, um die Relevanz der Daten für ihre Analyse zu gewichten.</p>	<p>Möller ist der Ansicht, dass der Markt für SBOMs bereits etabliert ist und es unwahrscheinlich ist, dass neue Frameworks ohne große Akzeptanz entstehen werden. Stattdessen erwartet er, dass bestehende Tools die etablierten Standards wie CycloneDX und SPDX verstärkt nutzen werden.</p>
<p>Obwohl Gype oder andere Scanner OCM in Zukunft nicht direkt verstehen werden, könnte Syft OCM-Daten in sein eigenes JSON-Format umwandeln, das dann von Gype verarbeitet werden kann. Dies ist bereits bei CycloneDX- und SPDX-SBOMs der Fall. Bressers hebt hervor, dass die Qualität von SBOM-Outputs je nach Generierungstool variieren kann, was zu Problemen bei der Identifizierung von Paketen durch Scanner wie Gype führen kann.</p>	<p>Springett sieht eine direkte Abbildung von OCM auf CycloneDX als praktikablen Weg, wobei die korrekte Taxonomie für nicht nativ in CycloneDX darstellbare Elemente und die Konsistenz zwischen den Tools entscheidend sind. Er warnt vor der Verwendung von Zwischenabstraktionen wie Protobom, da diese oft hinterherhinken und nicht alle Funktionen von CycloneDX unterstützen.</p>	<p>Häufige Missverständnisse bei der Nutzung von CycloneDX betreffen die Unterscheidung zwischen verschachtelten Komponenten (Assemblies) und Abhängigkeiten sowie die Formulierung, die oft zu präskriptiv implementiert wird. Auch die KI-Branche verwendet oft eigene Begriffe für bereits existierende Konzepte wie "Pedigree".</p>
<p>Er betont, dass kein Scanner für alles gut ist und Vergleiche zwischen Tools wie Gype und Trivy oft irreführend sind, da sie unterschiedliche Stärken</p>	<p>Er sieht CycloneDX in einer komplexen Position als Konsortium, das versucht, zukünftige Entwicklungen vorauszusehen und Unternehmen bei der Softwaretransparenz zu unterstützen. Möller glaubt, dass eine verlustfreie Konvertierung zwischen SBOM und OCM möglich ist, selbst bei nicht vollständig rekursiv aufgelösten Bäumen, da beide die Möglichkeit bieten, Gegenreferenzierungen über Metadaten aufzubauen. Er betont, dass Referenzen auf ursprüngliche Ressourcen im OCM wichtig sind, um</p>	

Anhang F: Summary-Tabelle

165

	<p>haben. Während Gypre und Trivy zu den am besten gepflegten Open-Source-Scannern gehören, gibt es auch spezialisierte Tools wie den Go BOM Checker, die in ihrem spezifischen Anwendungsbereich überlegen sind. Bressers rät dazu, die Ausgabe von SBOM-Tools mit anderen Systemen wie Dependency-Track, Trivy oder Gypre zu testen, um sicherzustellen, dass die Daten nutzbar sind.</p>	<p>Die SPDX-Spezifikation für Lizenzierung scheiterte daran, dass "license ref" in der Praxis nicht genutzt wird, was CycloneDX dazu veranlasste, unaufgelöste Lizenznamen und kommerzielle Lizenzierungsunterstützung zu integrieren, um Beschaffungsteams einzubeziehen. Springett empfiehlt kommerzielle Tools für die Lizenzierungsanalyse, da deren Datenwissenschaftsteams eine höhere Genauigkeit bei der Identifizierung von Komponenten, Metadaten und Schwachstellen aufweisen, während die meisten Open-Source-Tools in Tests versagten. Zukünftig könnte CycloneDX 2.0 mit seinem API-Format, das als normalisiertes Modell für den Transparency Exchange API dient, die Relevanz von Dateiformaten reduzieren und möglicherweise zu einer Vereinheitlichung oder sogar zu mehr Innovation bei BOM-Formaten führen.</p>	<p>eine sichere Rückkonvertierung zu gewährleisten.</p>
Kritik & Gegenargumente	<p>Josh Bressers von Anchore kritisiert, dass Gypre Schwierigkeiten hat, SBOMs zu scannen, die nicht von Syft erstellt wurden, obwohl an einer Lösung gearbeitet wird. Er äußert sich zudem verwundert darüber, dass die SBOM-Welt Informationen nicht so nachverfolgt, wie es OCM tut. Bressers empfindet die aktuelle Praxis, für jede Phase des Software-Lebenszyklus eine eigene SBOM zu erstellen, als übertrieben und "verrückt", da dies die einzige derzeitige Lösung sei.</p>	<p>Steve Springett weist darauf hin, dass es für Dienstleistungen ein entsprechendes Objekt gibt, das von zahlreichen kommerziellen und Open-Source-Tools unterstützt wird. Er betont, dass zumindest der Metadatenbereich der CycloneDX-Spezifikation für Komponenten und Dienste abgedeckt sein sollte. Allerdings merkt er an, dass fortgeschrittene Anwendungsfälle, wie beispielsweise der Pedigree-Teil einer Komponente, von den meisten Tools nicht unterstützt werden, da</p>	<p>Ein zentrales Gegenargument gegen die Übertragung von SBoDs in SBOMs ist der fundamentale Unterschied in ihrer Natur: SBoDs beschreiben ein transientes Verhalten in einer Zielumgebung, während SBOMs statische Artefakte sind, die sich nicht ändern. Eine SBOM repräsentiert den Status Quo eines Software-Artefakts zu einem bestimmten Zeitpunkt, dessen Abhängigkeiten konstant bleiben, sobald eine Softwareversion veröffentlicht wurde. Im Gegensatz dazu verändert</p>

Anhang F: Summary-Tabelle

166

	<p>Er hebt hervor, dass die meisten Menschen keine manuelle Bearbeitung von SBOMs wünschen, sondern eine automatisierte Lösung bevorzugen. Bressers prognostiziert, dass die Automatisierung von SBOMs in Zukunft so weit reifen wird, dass sie Aufgaben übernehmen kann, die heute noch menschliches Eingreifen erfordern, und somit eine Herausforderung für OCM darstellen könnte.</p> <p>Bressers berichtet von Problemen mit Kunden, die unvollständige oder unverständliche SBOMs einreichen. Dies führte dazu, dass Anchore eine spezielle API entwickeln musste, die JSON-Daten ohne Validierung speichert, um den Kundenanforderungen gerecht zu werden, selbst wenn die SBOMs keine nützlichen Informationen enthielten.</p>	<p>dies nicht automatisiert werden kann. Zudem erklärt Springett, dass eine Konvertierung der Signatur von einem Serialisierungsformat in ein anderes wahrscheinlich nicht möglich ist, da die Signatur für das jeweilige Datenpaket ungültig würde.</p>	<p>sich ein SBoD, sobald die Zielumgebung angepasst wird.</p> <p>Eine Konvertierung von SBoDs in SBOMs würde die statische Eigenschaft der SBOM aufheben, da angepasste Zielumgebungen zu dynamischen Änderungen führen würden. Dies würde konzeptionell zu Unklarheiten führen, da die Identität der SBOM in Frage gestellt wäre und sie nach den Definitionen eigentlich eine andere Identität annehmen müsste. Zudem müssten nachträglich anfallende Scan-Resultate dynamisch verknüpft werden, da sich zwar die SBOM selbst kaum ändern würde, aber die daraus resultierenden Scan-Ergebnisse.</p>	
Konzeptuelle Inkompatibilität	Zukunftsperspektiven	<p>Josh Bressers von Anchore merkt an, dass Cyclone DX sich sehr schnell entwickelt und es schwierig ist, den Überblick über alle Aktivitäten zu behalten. Er sieht eine Herausforderung darin, dass es bei einer großen Anzahl von SBOMs an Tools mangelt, die externe Referenzen korrekt zusammenführen, da die meisten Tools diese als separate Dokumente behandeln. Bressers prognostiziert, dass die SBOM-Automatisierung in Zukunft so weit reifen wird, dass sie Aufgaben übernehmen kann, die</p>	<p>CycloneDX hat sich von einer Spezifikation für eine vollständige Stückliste zu einer Sprache für Transparenz entwickelt. Zukünftige Versionen, insbesondere CycloneDX 1.7 und 2.0, konzentrieren sich nicht mehr auf Stücklisten, sondern auf die durchgängige Transparenz des gesamten Software-Lebenszyklus. Die Spezifikation zielt darauf ab, die Komplexität der Formulierung zu vereinfachen, indem ein maßgeblicher Leitfaden erstellt wird.</p> <p>Ein wichtiger Aspekt der</p>	<p>Jakob Möller geht davon aus, dass der Markt für SBOM-Formate bereits etabliert ist und keine neuen, weit verbreiteten Frameworks mehr entstehen werden. Er sieht die Herausforderung eher darin, dass bereits zwei etablierte Standards, CycloneDX und SPDX, existieren und viele Tools diese nutzen oder integrieren wollen, anstatt sie zu ersetzen. Er hält es für unrealistisch, dass Scanner zukünftig direkte OCM-Beschreibungen unterstützen könnten, da dies eine wohl definierte Implementierungsgrundlage erfordert und Scanner-</p>

Anhang F: Summary-Tabelle

167

	<p>heute noch von Menschen erledigt werden, was die Notwendigkeit manueller Eingriffe reduziert.</p> <p>Bezüglich der Unterstützung von OCM in Tools wie Syft erklärt Bressers, dass dies zwar wünschenswert wäre, die Dokumentation verschachtelter Pakete jedoch nicht standardisiert genug ist. Syft versucht, Beziehungsdaten zu erfassen, wo immer möglich, aber viele Ökosysteme bieten nur eine flache Liste von Abhängigkeiten. Er weist darauf hin, dass die aktuelle Nachfrage nach detaillierten Abhängigkeitsbäumen gering ist, da Kunden und Auditoren sich eher auf die Behebung kritischer Schwachstellen konzentrieren als auf die Struktur der Abhängigkeiten. Bressers hofft, dass VEX hier in Zukunft Abhilfe schaffen wird.</p> <p>Auf die Frage nach der Integration von OCM in Scanner wie Grype antwortet Bressers, dass er dies nicht direkt erwarten würde. Stattdessen würde Syft, sollte OCM populär werden, die Funktionalität entwickeln, um OCM in das Syft JSON-Format zu konvertieren, ähnlich wie es bereits mit CycloneDX und SPDX geschieht. Er bezweifelt, dass es jemals einen einzigen SBOM-Standard geben wird, da in jeder Branche mehrere Standards existieren, weil Akteure stets versuchen, bestehende Lösungen zu verbessern. Bressers ist sogar überrascht, dass es derzeit nur zwei Haupt-</p>	<p>zukünftigen Entwicklung ist die Transparency Exchange API, die den Austausch von Lieferkettenartefakten wie SBOMs ermöglicht und es erlaubt, autonom Fragen an Anbieter zu stellen, unabhängig vom Format der Stückliste. CycloneDX 2.0 wird nicht nur ein dateibasiertes Format bieten, sondern auch ein API-Format, das die Grundlage für die Transparency Exchange API bildet, insbesondere für Insights. Dies bedeutet, dass beliebige BOM-Formate in die API eingegeben werden können, aber bei der Verwendung von Insights das normalisierte Modell die CycloneDX 2.0 API ist.</p> <p>Die Version 1.7, die voraussichtlich im Dezember ratifiziert wird, wird die letzte der 1.x-Reihe sein, da alle weiteren Entwicklungen auf 2.0 basieren.</p> <p>Bezüglich der Zukunft von SBOM-Standards gibt es drei mögliche Szenarien: Entweder wird CycloneDX aufgrund seines normalisierten Modells bevorzugt, oder es wird weiterhin mehrere BOM-Formate geben, wobei die Normalisierung auf API-Ebene die Formate irrelevant macht, oder die Innovation in BOM-Formaten wird durch die neuen Möglichkeiten gefördert.</p>	<p>Anbieter nicht ständig neue Frameworks integrieren können. Stattdessen müsste es einen branchenweiten Standard geben, auf den man sich de facto einigt, wobei Ergänzungen mit etablierten Frameworks kompatibel sein sollten.</p> <p>Möller glaubt nicht, dass ein neues Transparenzformat ohne unterstützende Tools und Frameworks eine große Akzeptanz finden wird, wie es bei CycloneDX der Fall war. Er merkt an, dass OCM selbst überlegt, SBOM als Standardspeicherformat zu nutzen und OCM-Beschreiber nur als Laufzeitmodell zu verwenden, um bei Persistenz in ein CycloneDX-SBOM umzukonvertieren. Ein Wegfall von SBOM als vorgegebenes Format würde den Mehrwert für Integratoren mindern, wenn es kein Einheitsformat mehr gäbe. Bezüglich CycloneDX sieht er es in einer schwierigen Position als Konsortium, das von verschiedenen Firmen und Interessenvertretern getrieben wird. Er vergleicht dies mit der Politik, wo man versucht, zukünftige Entwicklungen abzuschätzen, was dazu führen kann, dass in manchen Bereichen zu viel und in anderen zu wenig investiert wird. Möller sieht jedoch keine Schwierigkeiten in der Adoption oder darin, dass CycloneDX zu weit voraus ist, da sich de jure Spezifikationen, die nicht angenommen werden, letztendlich an de facto</p>
--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Anhang F: Summary-Tabelle

168

	<p>SBOM-Standards gibt und erwartet, dass es in Zukunft mehr werden.</p> <p>Abschließend betont Bressers, dass Anchore PRs (Pull Requests) für die Implementierung von OCM-Unterstützung begrüßen würde, auch wenn es für sie selbst keine hohe Priorität hat. Olson Sturm zeigt sich bereit, in seiner Freizeit dazu beizutragen, da er bereits eine Idee zur Implementierung hat und sich mit dem Quellcode vertraut gemacht hat.</p>	Implementierungen anpassen.
Zukunft der SBOM-Standards	<p>Josh Bressers von Anchore befürchtet, dass CycloneDX aufgrund seiner schnellen Entwicklung und Komplexität Nutzer verlieren könnte, die sich stattdessen für das langsamere und "langweiligere" SPDX entscheiden werden. Er hat diese Bedenken bereits mehrfach geäußert, stößt aber auf Widerstand. Bressers wünscht sich zwar einen einzigen SBOM-Standard, ist aber realistisch, dass dies aufgrund der menschlichen Tendenz, immer bessere Standards schaffen zu wollen, unwahrscheinlich ist. Er ist sogar überrascht, dass es derzeit nur zwei SBOM-Standards gibt, und erwartet, dass es in Zukunft noch mehr geben wird, was er mit dem bekannten XKCD-Comic über die Schaffung neuer Standards vergleicht.</p>	<p>Steve Springett sagt, es gebe bereits einige Tools für „Formulation“, das Konzept sei zwar simpel, werde aber sehr strikt/preskriptiv umgesetzt. Daher erarbeitet das CycloneDX-Team einen autoritativen Leitfaden, wie „Formulation“ korrekt umzusetzen ist.</p>
Konkrete Anforderungen	<p>Josh Bressers von Anchore betont, dass für seine Arbeit, insbesondere mit dem Tool Gype, die Paketidentifikation, oft über PURL in SBOMs, von</p>	OCM ist notwendig, da ein klassisches SBOM nicht ausreicht, um eine Softwarelieferung vollständig zu beschreiben, wobei jedoch

<p>größter Bedeutung ist. Er hebt hervor, dass die Paketinformationen das Wesentliche sind, um Schwachstellen zu scannen und spezifische Anfragen, wie das Vorhandensein von Log4J in einer großen Menge von SBOMs, beantworten zu können. Eine direkte OCM-zu-CycloneDX-Abbildung birgt das Risiko, die Detailtiefe zu verlieren. Für die Anforderungen an CycloneDX-SBOMs für Tools wie Grype ist die Erfassung der Paketinformationen entscheidend. Bressers rät dazu, die Ausgabe eines neuen CLI-Tools mit anderen bestehenden Tools wie Dependency Track, Trivy oder SBOMit zu testen, um sicherzustellen, dass die generierten SBOMs in der Praxis nutzbar sind und nicht als "zufällige Daten" enden.</p>	<p>die Kompatibilität mit bestehenden SBOM-Ökosystemen angestrebt wird. Ein wesentlicher Mehrwert liegt in der Konvertierung eines Delivery-Artefakts in einen Snapshot oder ein Statement of Time, das dann wieder über ein SBOM repräsentiert werden kann. Dabei ist es entscheidend, etablierte Konventionen zu nutzen, um die globale Eindeutigkeit von Komponentenidentitäten sicherzustellen und semantisch äquivalente Strukturen zu erhalten.</p>
<p>Die Konvertierung eines OCM-Komponentenbeschreibers in ein SBOM muss so erfolgen, dass es von anderen Tools verarbeitet werden kann, ohne eine neue, inkompatible Zwischenbeschreibung zu schaffen. Zwei Hauptkriterien sind hierbei die Rückverfolgbarkeit nach der Konvertierung, um eine Rückreferenzierung oder Rückkonvertierung zu ermöglichen, sowie die semantische Vollständigkeit und Sinnhaftigkeit des SBOMs. Dies bedeutet, dass alle Felder korrekt referenziert und übertragen werden müssen und von anderen Tools, wie beispielsweise CycloneDX SBOM, auslesbar sein sollten.</p>	<p>Die Generierung eines SBOM kann entweder zur Build- oder zur Delivery-Zeit erfolgen. Alternativ kann der Beschreiber auch ausgeliefert werden, und ein Konsument führt dann eine Live-Analyse und Konvertierung durch. Das semantisch korrekteste</p>

Mapping basiert auf einer global eindeutigen und rückreferenzierbaren Komponentenidentität, die sich nie ändert. Durch die Annahme einer globalen Komponentenidentität wird jede Ressourcenidentität ebenfalls global, was den Aufbau einer statischen und immutablen SBOM-Referenz ermöglicht, die lokationsunabhängig funktioniert. Die Rückführbarkeit der generierten SBOMs ist von größter Bedeutung und sollte sowohl als Code-Abhängigkeit als auch als Tooling, beispielsweise über ein Command Line Interface, angeboten werden.

Anhang G: Table of Minimum Elements Data Fields

Data Field	Description
SBOM Author	The name of the entity that creates the SBOM data for this component.
Software Producer	The name of an entity that creates, defines, and identifies components.
Component Name	The name assigned by the <i>Software Producer</i> to a software component.
Component Version	Identifier used by the <i>Software Producer</i> to specify a change in software from a previously identified version.
Software Identifiers	Identifier(s) used to identify a component or serve as a look-up key for relevant databases.
Component Hash	The cryptographic value generated from taking the hash of the software component.
License	The license(s) under which the software component is made available.
Dependency Relationship	The relationship between two software components, specifically noting that Software X includes Component Y or that Component A is largely derived from Component B.
Tool Name	The name of the tool used by the <i>SBOM Author</i> to generate the SBOM.
Timestamp	Record of the date and time of the most recent update to the SBOM data
Generation Context	The relative software lifecycle phase and data available at the time the <i>Software Producer</i> generated the SBOM (before build, during build, after build).

Abb. 7.1: Table of Minimum Elements Data Fields
 (vgl. CISA, 2025, S. 14)

Anhang H: Exemplarischer OCM Component Descriptor

```
1 component:
2   name: example.org/myapp
3   version: 1.2.3
4   provider: example-inc
5   labels:
6     - name: org.example/deploy/namespace
7       value: production
8   repositoryContexts:
9     - type: OCIRegistry
10    baseUrl: ghcr.io
11    subPath: example/ocm
12    componentNameMapping: urlPath
13   resources:
14     - name: app-image
15       type: ociImage
16       version: "1.2.3"
17       relation: external          # by-reference
18     extraIdentity:
19       os: linux                  # extra Identity für Artefaktvarianten
20       architecture: amd64
21   access:
22     type: ociArtifact
23     imageReference: ghcr.io/example/myapp:1.2.3
24   digest:
25     hashAlgorithm: SHA-256
26     normalisationAlgorithm: ociArtifactDigest/v1
27     value: "a1b2c3...9def"      # Platzhalter
28   - name: app-chart
29     type: helmChart
30     version: "1.2.3"
31     relation: local           # by-value
32   access:
33     type: localBlob
34     mediaType: application/vnd.oci.image.manifest.v1+tar+gzip
35     localReference: "sha256:4e5f6a...bb88"      # Platzhalter
36     referenceName: "example.org/myapp/charts/myapp:1.2.3"
37   labels:
38     - name: org.example/helm/values-schema
39       value:
40         url: "https://example.org/myapp/values.schema.json"
41         signing: false
42   sources:
```

```
43   - name: myapp-src
44     type: git
45     version: "v1.2.3"
46     access:
47       type: GitHub
48       repoUrl: github.com/example/myapp
49       ref: refs/tags/v1.2.3
50       commit: "0123456abcdef..."           # Platzhalter
51   componentReferences:
52   - name: crypto-lib
53     componentName: example.org/libs/crypto
54     version: "0.9.0"
55     extraIdentity:
56       edition: "oss"                   # extraIdentity auch auf Referenzen
57     möglich
58     labels:
59       - name: org.example/required
60         value: true
61     digest:
62       hashAlgorithm: SHA-256
63       normalisationAlgorithm: jsonNormalisation/v1
64       value: "77aa...cc33"           # Platzhalter
65   signatures:
66   - name: prod
67     digest:
68       hashAlgorithm: SHA-256
69       normalisationAlgorithm: jsonNormalisation/v1
70       value: "1122...aabb"           # Platzhalter
71     signature:
72       algorithm: RSASSA-PKCS1-V1_5
73       mediaType: application/vnd.ocm.signature.rsa
74       value: "Base64Signatur..."  # Platzhalter
```

Listing 8.1: Exemplarischer OCM Component Descriptor

Anhang I: Quellcode cyclonedx_merge.go

```
1 package converter
2
3 import (
4     "errors"
5     "fmt"
6     "reflect"
7
8     "github.com/CycloneDX/cyclonedx-go"
9     "github.com/google/uuid"
10 )
11
12 // CycloneDxMergeMode defines the merge strategy.
13 type CycloneDxMergeMode string
14
15 const (
16     MergeModeHierarchical CycloneDxMergeMode = "hierarchical"
17     MergeModeFlat          CycloneDxMergeMode = "flat"
18 )
19
20 // CycloneDxMergeOptions holds the parameters for the ComponentSbomMerge
21 // function.
22 type CycloneDxMergeOptions struct {
23     BOMs      []cyclonedx.BOM
24     Group     string
25     Name      string
26     Version   string
27     MergeMode CycloneDxMergeMode
28 }
29
30 // ErrMissingMetadataComponent indicates that a BOM is missing its
31 // required metadata component
32 var ErrMissingMetadataComponent = errors.New("required metadata (top
33 // CycloneDXMerge orchestrates the merging of BOMs based on the provided
34 // options.
35 func CycloneDXMerge(options CycloneDxMergeOptions) (*cyclonedx.BOM, error
36 ) {
37     // Hierarchical merge requires a name and version for the top-level
38     // component.
39     if options.MergeMode == MergeModeHierarchical && (options.Name == "" ||
40         options.Version == "") {
41         return nil, errors.New("name and version must be specified for a
42         hierarchical merge")
```

```
37     }
38
39     var outputBom *cyclonedx.BOM
40     var err error
41
42     if options.MergeMode == MergeModeHierarchical {
43         bomSubject := &cyclonedx.Component{
44             Type:    cyclonedx.ComponentTypeApplication,
45             Group:   options.Group,
46             Name:    options.Name,
47             Version: options.Version,
48         }
49         outputBom, err = HierarchicalMerge(options.BOMs, bomSubject)
50     } else if options.MergeMode == MergeModeFlat {
51         outputBom, err = FlatMerge(options.BOMs, nil)
52     } else {
53         return nil, fmt.Errorf("unsupported merge mode: %s", options.
54         MergeMode)
55     }
56
57     if err != nil {
58         return nil, fmt.Errorf("merge failed: %w", err)
59     }
60
61     // Finalize BOM metadata.
62     if outputBom != nil {
63         outputBom.Version = 1
64         if outputBom.Metadata == nil {
65             outputBom.Metadata = &cyclonedx.Metadata{}
66         }
67
68         // Set a new UUID serial number for the output BOM
69         outputBom.SerialNumber = fmt.Sprintf("urn:uuid:%s", uuid.New().String
70         ())
71
72         return outputBom, nil
73     }
74
75     // HierarchicalMerge performs a hierarchical merge for multiple BOMs.
76     // To retain system component hierarchy, top level BOM metadata
77     // component must be included in each BOM.
78     func HierarchicalMerge(boms []cyclonedx.BOM, bomSubject *cyclonedx.
79         Component) (*cyclonedx.BOM, error) {
80         result := &cyclonedx.BOM{}
```

```
80 // Initialize metadata if bomSubject is provided
81 if bomSubject != nil {
82     if bomSubject.BOMRef == "" {
83         bomSubject.BOMRef = componentBOMRefNamespace(bomSubject)
84     }
85     result.Metadata = &cyclonedx.Metadata{
86         Component: bomSubject,
87         Tools:     &cyclonedx.ToolsChoice{},
88     }
89 }
90
91 // Initialize all lists as non-nil
92 result.Components = &[]cyclonedx.Component{}
93 result.Services = &[]cyclonedx.Service{}
94 result.ExternalReferences = &[]cyclonedx.ExternalReference{}
95 result.Dependencies = &[]cyclonedx.Dependency{}
96 result.Compositions = &[]cyclonedx.Composition{}
97 result.Vulnerabilities = &[]cyclonedx.Vulnerability{}
98
99 result.Declarations = &cyclonedx.Declarations{
100    Assessors:    &[]cyclonedx.Assessor{},
101    Attestations: &[]cyclonedx.Attestation{},
102    Claims:       &[]cyclonedx.Claim{},
103    Evidence:     &[]cyclonedx.DeclarationEvidence{},
104    Targets:      &cyclonedx.Targets{
105        Components:   &[]cyclonedx.Component{},
106        Organizations: &[]cyclonedx.OrganizationalEntity{},
107        Services:     &[]cyclonedx.Service{},
108    },
109 }
110
111 result.Definitions = &cyclonedx.Definitions{
112    Standards: &[]cyclonedx.StandardDefinition{},
113 }
114
115 var bomSubjectDependencies []cyclonedx.Dependency
116
117 for _, bom := range boms {
118     // Validate BOM has required metadata component
119     if bom.Metadata == nil || bom.Metadata.Component == nil {
120         serialNumber := "unknown"
121         if bom.SerialNumber != "" {
122             serialNumber = bom.SerialNumber
123         }
124         return nil, fmt.Errorf("%w (BOM: %s)", ErrMissingMetadataComponent,
125                                serialNumber)
126 }
```

```
125     }
126
127     // ComponentSbomMerge metadata tools - only if tools exist and result
128     // metadata is not nil
129     if result.Metadata != nil && bom.Metadata.Tools != nil {
130         // Handle Tools Components
131         if bom.Metadata.Tools.Components != nil && len(*bom.Metadata.Tools.
132             Components) > 0 {
133             if result.Metadata.Tools.Components == nil {
134                 result.Metadata.Tools.Components = &[]cyclonedx.Component{}
135             }
136             for _, component := range *bom.Metadata.Tools.Components {
137                 // Apply namespace to component
138                 namespaceComponentBOMRefs(componentBOMRefNamespace(bom.Metadata.
139                     Component), &component)
140                 if !containsComponent(*result.Metadata.Tools.Components,
141                     component) {
142                     *result.Metadata.Tools.Components = append(*result.Metadata.
143                         Tools.Components, component)
144                 }
145             }
146         }
147
148         // Handle Tools Services
149         if bom.Metadata.Tools.Services != nil && len(*bom.Metadata.Tools.
150             Services) > 0 {
151             if result.Metadata.Tools.Services == nil {
152                 result.Metadata.Tools.Services = &[]cyclonedx.Service{}
153             }
154             for _, service := range *bom.Metadata.Tools.Services {
155                 service.BOMRef = namespacedBOMRef(bom.Metadata.Component,
156                     service.BOMRef)
157                 if !containsService(*result.Metadata.Tools.Services, service) {
158                     *result.Metadata.Tools.Services = append(*result.Metadata.
159                         Tools.Services, service)
160                 }
161             }
162         }
163
164         // Process main component
165         thisComponent := bom.Metadata.Component
166         if thisComponent.Components == nil {
167             thisComponent.Components = &[]cyclonedx.Component{}
168         }
169         if bom.Components != nil {
```

```
163     *thisComponent.Components = append(*thisComponent.Components, *bom.
164     Components...)
165 }
166
167 // Add namespace to existing BOM refs (this modifies the original
168 // component!)
169 namespaceComponentBOMRefs(componentBOMRefNamespace(thisComponent),
170 thisComponent)
171
172 // Ensure BOM ref is set and add top level dependency reference
173 if thisComponent.BOMRef == "" {
174     thisComponent.BOMRef = componentBOMRefNamespace(thisComponent)
175 }
176 bomSubjectDependencies = append(bomSubjectDependencies, cyclonedx.
177 Dependency{Ref: thisComponent.BOMRef})
178
179 *result.Components = append(*result.Components, *thisComponent)
180
181 // ComponentSbomMerge services
182 if bom.Services != nil {
183     for _, service := range *bom.Services {
184         service.BOMRef = namespacedBOMRef(bom.Metadata.Component, service.
185         BOMRef)
186         *result.Services = append(*result.Services, service)
187     }
188 }
189
190 // ComponentSbomMerge external references
191 if bom.ExternalReferences != nil {
192     *result.ExternalReferences = append(*result.ExternalReferences, *
193     bom.ExternalReferences...)
194 }
195
196 // ComponentSbomMerge dependencies
197 if bom.Dependencies != nil {
198     namespaceDependencyBOMRefs(componentBOMRefNamespace(thisComponent),
199     *bom.Dependencies)
200     *result.Dependencies = append(*result.Dependencies, *bom.
201     Dependencies...)
202 }
203
204 // ComponentSbomMerge compositions
205 if bom.Compositions != nil {
206     namespaceCompositions(componentBOMRefNamespace(bom.Metadata.
207     Component), *bom.Compositions)
```

```
199     *result.Compositions = append(*result.Compositions, *bom.
200     Compositions...)
201 }
202
203 // ComponentSbomMerge vulnerabilities - NOTE: uses result.Metadata.
204 // Component namespace, not thisComponent!
205 if bom.Vulnerabilities != nil {
206     var namespaceForVulns string
207     if result.Metadata != nil && result.Metadata.Component != nil {
208         namespaceForVulns = componentBOMRefNamespace(result.Metadata.
209         Component)
210     } else {
211         namespaceForVulns = componentBOMRefNamespace(bom.Metadata.
212         Component)
213     }
214     namespaceVulnerabilitiesRefs(namespaceForVulns, *bom.
215     Vulnerabilities)
216     *result.Vulnerabilities = append(*result.Vulnerabilities, *bom.
217     Vulnerabilities...)
218 }
219
220
221 // Define local helper functions
222 namespaceBOMRefs := func(refs interface{}) {
223     namespaceBOMRefsWithComponent(thisComponent, refs)
224 }
225 namespaceReference := func(refs interface{}, propertyName string) {
226     namespaceProperty(componentBOMRefNamespace(thisComponent), refs,
227     propertyName)
228 }
229
230
231 // ComponentSbomMerge definitions
232 if bom.Definitions != nil && bom.Definitions.Standards != nil {
233     // Namespace all references
234     namespaceBOMRefs(*bom.Definitions.Standards)
235     for i := range *bom.Definitions.Standards {
236         standard := &(*bom.Definitions.Standards)[i]
237         if standard.Requirements != nil {
238             namespaceBOMRefs(*standard.Requirements)
239         }
240         if standard.Levels != nil {
241             namespaceBOMRefs(*standard.Levels)
242             namespaceReference(*standard.Levels, "Requirements")
243         }
244     }
245     *result.Definitions.Standards = append(*result.Definitions.
246     Standards, *bom.Definitions.Standards...)
247 }
```

```
237     }
238
239     // ComponentSbomMerge declarations
240     if bom.Declarations != nil {
241         // Assessors
242         if bom.Declarations.Assessors != nil {
243             namespaceBOMRefs(*bom.Declarations.Assessors)
244             *result.Declarations.Assessors = append(*result.Declarations.
245             Assessors, *bom.Declarations.Assessors...)
246         }
247
248         // Attestations
249         if bom.Declarations.Attestations != nil {
250             namespaceReference(*bom.Declarations.Attestations, "Assessor")
251             for _, attestation := range *bom.Declarations.Attestations {
252                 if attestation.Map != nil {
253                     namespaceReference(*attestation.Map, "Claims")
254                     namespaceReference(*attestation.Map, "CounterClaims")
255                     namespaceReference(*attestation.Map, "Requirement")
256
257                     // Handle conformance mitigation strategies
258                     for _, m := range *attestation.Map {
259                         if m.Conformance != nil {
260                             namespaceReference([]interface{}{m.Conformance}, "
261                             MitigationStrategies")
262                         }
263                     }
264                     *result.Declarations.Attestations = append(*result.Declarations.
265                     Attestations, *bom.Declarations.Attestations...)
266                 }
267
268             // Claims
269             if bom.Declarations.Claims != nil {
270                 namespaceBOMRefs(*bom.Declarations.Claims)
271                 namespaceReference(*bom.Declarations.Claims, "Evidence")
272                 namespaceReference(*bom.Declarations.Claims, "CounterEvidence")
273                 namespaceReference(*bom.Declarations.Claims, "Target")
274                 *result.Declarations.Claims = append(*result.Declarations.Claims,
275                 *bom.Declarations.Claims...)
276
277             // Evidence
278             if bom.Declarations.Evidence != nil {
279                 namespaceBOMRefs(*bom.Declarations.Evidence)
```

```
279         *result.Declarations.Evidence = append(*result.Declarations.
Evidence, *bom.Declarations.Evidence...)
280     }
281
282     // Targets
283     if result.Declarations.Targets != nil {
284         namespaceBOMRefs(*result.Declarations.Targets.Organizations)
285         namespaceBOMRefs(*result.Declarations.Targets.Components)
286         namespaceBOMRefs(*result.Declarations.Targets.Services)
287     }
288
289     if bom.Declarations.Targets != nil {
290         if bom.Declarations.Targets.Organizations != nil {
291             *result.Declarations.Targets.Organizations = append(*result.
Declarations.Targets.Organizations, *bom.Declarations.Targets.
Organizations...)
292         }
293         if bom.Declarations.Targets.Components != nil {
294             *result.Declarations.Targets.Components = append(*result.
Declarations.Targets.Components, *bom.Declarations.Targets.Components
...)
295         }
296         if bom.Declarations.Targets.Services != nil {
297             *result.Declarations.Targets.Services = append(*result.
Declarations.Targets.Services, *bom.Declarations.Targets.Services...)
298         }
299     }
300 }
301 }
302
303 // Add final dependency structure if bomSubject exists
304 if bomSubject != nil {
305     refs := make([]string, len(bomSubjectDependencies))
306     for i, dep := range bomSubjectDependencies {
307         refs[i] = dep.Ref
308     }
309     *result.Dependencies = append(*result.Dependencies, cyclonedx.
Dependency{
310         Ref:           result.Metadata.Component.BOMRef,
311         Dependencies: &refs,
312     })
313 }
314
315 // Cleanup empty top level elements
316 if result.Metadata != nil && result.Metadata.Tools != nil && result.
Metadata.Tools.Components != nil && len(*result.Metadata.Tools.
```

```
Components) == 0 {
    result.Metadata.Tools.Components = nil
}
if len(*result.Components) == 0 {
    result.Components = nil
}
if len(*result.Services) == 0 {
    result.Services = nil
}
if len(*result.ExternalReferences) == 0 {
    result.ExternalReferences = nil
}
if len(*result.Dependencies) == 0 {
    result.Dependencies = nil
}
if len(*result.Compositions) == 0 {
    result.Compositions = nil
}
if len(*result.Vulnerabilities) == 0 {
    result.Vulnerabilities = nil
}

return result, nil
}

// FlatMerge performs a flat merge of multiple BOMs, without a
// hierarchical structure.
func FlatMerge(boms []cyclonedx.BOM, bomSubject *cyclonedx.Component) (*
    cyclonedx.BOM, error) {
    result := &cyclonedx.BOM{}


    // Initialize lists
    result.Components = &[]cyclonedx.Component{}
    result.Services = &[]cyclonedx.Service{}
    result.ExternalReferences = &[]cyclonedx.ExternalReference{}
    result.Dependencies = &[]cyclonedx.Dependency{}
    result.Compositions = &[]cyclonedx.Composition{}
    result.Vulnerabilities = &[]cyclonedx.Vulnerability{}


    for _, bom := range boms {
        // Merge components
        if bom.Components != nil {
            for _, comp := range *bom.Components {
                if !containsComponent(*result.Components, comp) {
                    *result.Components = append(*result.Components, comp)
                }
            }
        }
    }
}
```

```
360     }
361 }
362
363 // Merge services
364 if bom.Services != nil {
365     for _, service := range *bom.Services {
366         if !containsService(*result.Services, service) {
367             *result.Services = append(*result.Services, service)
368         }
369     }
370 }
371
372 // Merge external references
373 if bom.ExternalReferences != nil {
374     *result.ExternalReferences = append(*result.ExternalReferences, *
375 bom.ExternalReferences...)
376 }
377
378 // Merge dependencies
379 if bom.Dependencies != nil {
380     *result.Dependencies = append(*result.Dependencies, *bom.
381 Dependencies...)
382 }
383
384 // Merge compositions
385 if bom.Compositions != nil {
386     *result.Compositions = append(*result.Compositions, *bom.
387 Compositions...)
388 }
389
390 // Merge vulnerabilities
391 if bom.Vulnerabilities != nil {
392     *result.Vulnerabilities = append(*result.Vulnerabilities, *bom.
393 Vulnerabilities...)
394 }
395
396 // Cleanup empty top level elements
397 if len(*result.Components) == 0 {
398     result.Components = nil
399 }
400 if len(*result.Services) == 0 {
401     result.Services = nil
402 }
403 if len(*result.ExternalReferences) == 0 {
404     result.ExternalReferences = nil
405 }
```

```
402     }
403     if len(*result.Dependencies) == 0 {
404         result.Dependencies = nil
405     }
406     if len(*result.Compositions) == 0 {
407         result.Compositions = nil
408     }
409     if len(*result.Vulnerabilities) == 0 {
410         result.Vulnerabilities = nil
411     }
412
413     return result, nil
414 }
415
416 // Helper functions
417
418 // componentBOMRefNamespace generates a namespace for a component's BOM
// reference
419 func componentBOMRefNamespace(component *cyclonedx.Component) string {
420     if component == nil {
421         return ""
422     }
423
424     if component.Group != "" {
425         return fmt.Sprintf("%s.%s@%s", component.Group, component.Name,
426                         component.Version)
427     }
428     return fmt.Sprintf("%s@%s", component.Name, component.Version)
429 }
430
431 // namespacedBOMRef creates a namespaced BOM reference
432 func namespacedBOMRef(component *cyclonedx.Component, bomRef string)
        string {
433     if bomRef == "" {
434         return ""
435     }
436     return namespacedBOMRefWithNamespace(componentBOMRefNamespace(component),
437                                         bomRef)
438 }
439
440 // namespacedBOMRefWithNamespace creates a namespaced BOM reference with
// namespace string
441 func namespacedBOMRefWithNamespace(bomRefNamespace, bomRef string) string
        {
442     if bomRef == "" {
443         return ""
444     }
```

```
442     }
443     if bomRefNamespace == "" {
444         return bomRef
445     }
446     return fmt.Sprintf("%s:%s", bomRefNamespace, bomRef)
447 }
448
449 // namespaceBOMRefsWithComponent adds namespace to objects with BOM
// references
450 func namespaceBOMRefsWithComponent(bomSubject *cyclonedx.Component,
451                                     references interface{}) {
452     if references == nil {
453         return
454     }
455     namespace := componentBOMRefNamespace(bomSubject)
456     namespaceBOMRefs(namespace, references)
457 }
458
459 // namespaceBOMRefs adds namespace to objects that implement BOM
// reference interface
460 func namespaceBOMRefs(namespace string, items interface{}) {
461     if namespace == "" || items == nil {
462         return
463     }
464     v := reflect.ValueOf(items)
465     if v.Kind() != reflect.Slice {
466         return
467     }
468
469     for i := 0; i < v.Len(); i++ {
470         item := v.Index(i)
471         if item.Kind() == reflect.Ptr {
472             item = item.Elem()
473         }
474
475         if item.Kind() == reflect.Struct {
476             bomRefField := item.FieldByName("BOMRef")
477             if bomRefField.IsValid() && bomRefField.CanSet() && bomRefField.
478                 Kind() == reflect.String {
479                 currentRef := bomRefField.String()
480                 if currentRef != "" {
481                     bomRefField.SetString(namespacedBOMRefWithNamespace(namespace,
482                                         currentRef))
483                 }
484             }
485         }
486     }
487 }
```

```
483     }
484   }
485 }
486
487 // namespaceProperty applies a namespace transformation to a specified
488 // property
489 func namespaceProperty(namespace string, items interface{}, propertyName
490   string) {
491   if namespace == "" || items == nil || propertyName == "" {
492     return
493   }
494
495   v := reflect.ValueOf(items)
496   if v.Kind() != reflect.Slice {
497     return
498   }
499
500   for i := 0; i < v.Len(); i++ {
501     item := v.Index(i)
502
503     if item.Kind() == reflect.Ptr {
504       item = item.Elem()
505     }
506
507     if item.Kind() != reflect.Struct {
508       continue
509     }
510
511     propField := item.FieldByName(propertyName)
512     if !propField.IsValid() || !propField.CanSet() {
513       continue
514     }
515
516     switch propField.Kind() {
517     case reflect.String:
518       currentValue := propField.String()
519       if currentValue != "" {
520         propField.SetString(namespacedBOMRefWithNamespace(namespace,
521                           currentValue))
522       }
523     case reflect.Slice:
524       if propField.Type().Elem().Kind() == reflect.String {
525         if propField.IsNil() {
526           continue
527         }
528         currentSlice := propField.Interface().([]string)
529         updatedSlice := make([]string, len(currentSlice))
```

```
526     for j, s := range currentSlice {
527         updatedSlice[j] = namespacedBOMRefWithNamespace(namespace, s)
528     }
529     propField.Set(reflect.ValueOf(updatedSlice))
530 }
531 case reflect.Ptr:
532     if propField.Type().Elem().Kind() == reflect.Slice {
533         if propField.IsNil() {
534             continue
535         }
536         sliceVal := propField.Elem()
537         if sliceVal.Type().Elem().Kind() == reflect.String {
538             currentSlice := sliceVal.Interface().([]string)
539             updatedSlice := make([]string, len(currentSlice))
540             for j, s := range currentSlice {
541                 updatedSlice[j] = namespacedBOMRefWithNamespace(namespace, s)
542             }
543             sliceVal.Set(reflect.ValueOf(updatedSlice))
544         }
545     }
546 }
547 }
548 }
549
550 // namespaceComponentBOMRefs adds namespace to component BOM references
551 // using stack
552 func namespaceComponentBOMRefs(bomRefNamespace string, topComponent *cyclonedx.Component) {
553     if topComponent == nil || bomRefNamespace == "" {
554         return
555     }
556     components := []*cyclonedx.Component{topComponent}
557
558     for len(components) > 0 {
559         currentComponent := components[len(components)-1]
560         components = components[:len(components)-1]
561
562         if currentComponent.Components != nil {
563             for i := range *currentComponent.Components {
564                 components = append(components, &(*currentComponent.Components)[i])
565             }
566         }
567     }
568 }
```

```
568     currentComponent.BOMRef = namespacedBOMRefWithNamespace(
569         bomRefNamespace, currentComponent.BOMRef)
570     }
571 }
572 // namespaceDependencyBOMRefs adds namespace to dependency BOM references
573 // using stack
574 func namespaceDependencyBOMRefs(bomRefNamespace string, dependencies []cyclonedx.Dependency) {
575     if bomRefNamespace == "" {
576         return
577     }
578
579     pendingDependencies := make([]*cyclonedx.Dependency, len(dependencies))
580     for i := range dependencies {
581         pendingDependencies[i] = &dependencies[i]
582     }
583
584     for len(pendingDependencies) > 0 {
585         dependency := pendingDependencies[len(pendingDependencies)-1]
586         pendingDependencies = pendingDependencies[:len(pendingDependencies)-
587             1]
588
589         if dependency.Dependencies != nil {
590             for j := range *dependency.Dependencies {
591                 (*dependency.Dependencies)[j] = namespacedBOMRefWithNamespace(
592                     bomRefNamespace, (*dependency.Dependencies)[j])
593             }
594         }
595     }
596 }
597 // namespaceCompositions adds namespace to composition BOM references
598 func namespaceCompositions(bomRefNamespace string, compositions []cyclonedx.Composition) {
599     if bomRefNamespace == "" {
600         return
601     }
602
603     for i := range compositions {
604         if compositions[i].Assemblies != nil {
605             for j := range *compositions[i].Assemblies {
606                 currentRef := string((*compositions[i].Assemblies)[j])
```

```
607         (*compositions[i].Assemblies)[j] = cyclonedx.BOMReference(
608             namespacedBOMRefWithNamespace(bomRefNamespace, currentRef))
609     }
610   }
611
612   if compositions[i].Dependencies != nil {
613     for j := range *compositions[i].Dependencies {
614       currentRef := string((*compositions[i].Dependencies)[j])
615       (*compositions[i].Dependencies)[j] = cyclonedx.BOMReference(
616           namespacedBOMRefWithNamespace(bomRefNamespace, currentRef))
617     }
618   }
619
620 // namespaceVulnerabilitiesRefs adds namespace to vulnerability BOM
621 // references using stack
622 func namespaceVulnerabilitiesRefs(bomRefNamespace string, vulnerabilities
623                                     []cyclonedx.Vulnerability) {
624   if bomRefNamespace == "" {
625     return
626   }
627
628   pendingVulnerabilities := make([]*cyclonedx.Vulnerability, len(
629     vulnerabilities))
630   for i := range vulnerabilities {
631     pendingVulnerabilities[i] = &vulnerabilities[i]
632   }
633
634   for len(pendingVulnerabilities) > 0 {
635     vulnerability := pendingVulnerabilities[len(pendingVulnerabilities)
636 -1]
637     pendingVulnerabilities = pendingVulnerabilities[:len(
638     pendingVulnerabilities)-1]
639
640     vulnerability.BOMRef = namespacedBOMRefWithNamespace(bomRefNamespace,
641     vulnerability.BOMRef)
642
643     if vulnerability.Affects != nil {
644       for j := range *vulnerability.Affects {
645         affect := &(*vulnerability.Affects)[j]
646         affect.Ref = bomRefNamespace
647       }
648     }
649   }
650 }
```

```
645
646 // containsComponent checks if a component already exists in the slice
647 func containsComponent(components []cyclonedx.Component, target cyclonedx
648 .Component) bool {
649     for _, comp := range components {
650         if comp.BOMRef == target.BOMRef {
651             return true
652         }
653     }
654     return false
655 }
656 // containsService checks if a service already exists in the slice
657 func containsService(services []cyclonedx.Service, target cyclonedx.
658 Service) bool {
659     for _, svc := range services {
660         if svc.BOMRef == target.BOMRef {
661             return true
662         }
663     }
664 }
```

Listing 9.1: Quellcode cyclonedx_merge.go

Anhang J: Quellcode convert_impl.go

```
1 package converter
2
3 import (
4     "context"
5     "fmt"
6     "log"
7     "os"
8     "path/filepath"
9     "strings"
10
11    "ocm.software/open-component-model/bindings/go/oci"
12 )
13
14 // ConvertOCMToSBOM orchestrates reading components from a CTF,
15 // generating SBOMs for their
16 // resources via Syft, merging a per-component via CycloneDX CLI (default
17 // ), and optionally
18 // converting the final output format.
19 func (c *CLIConverter) ConvertOCMToSBOM(cftPath string, componentName
20     string, targetFormat SBOMFormat, mergeTool string) ([]byte, error) {
21     repo, err := createRepository(cftPath)
22     if err != nil {
23         return nil, fmt.Errorf("error creating repository: %w", err)
24     }
25
26     // Process all components recursively starting at the given component (
27     // use version 1.0.0 for now) TODO: support version selection
28     allComponentSBOMPaths, err := c.processAllComponents(repo,
29     componentName, "1.0.0", targetFormat, mergeTool)
30     if err != nil {
31         return nil, fmt.Errorf("error processing components: %w", err)
32     }
33     if len(allComponentSBOMPaths) == 0 {
34         log.Println("No SBOMs were generated for any components. Result will
35         be empty.")
36         return []byte{}, nil
37     }
38
39     // The first entry is the fully merged SBOM for the root component (
40     // parent stays root).
41     rootComponentSbomPath := allComponentSBOMPaths[0]
42     log.Printf("Final merged SBOM at: %s", rootComponentSbomPath)
43
44     // Optionally convert to the desired output format
```

```
38     return c.convertFinalSBOM(rootComponentSbomPath, targetFormat)
39 }
40
41 // processAllComponents traverses the component hierarchy, generates
42 // SBOMs for each component's
43 // resources, then merges bottom-up so each parent includes its children.
44 // The returned slice's
45 // first element is the root component's fully merged SBOM path.
46 // go
47 func (c *CLIConverter) processAllComponents(repo oci.
48     ComponentVersionRepository, componentName, componentVersion string,
49     outputFormat SBOMFormat, mergeTool string) ([]string, error) {
50 type void = struct{}
51
52 processor := NewComponentProcessor(c)
53 merger := NewComponentSbomMerger(c)
54
55 // Graph and bookkeeping
56 type compKey struct{ Name, Version string }
57 id := func(n, v string) string { return fmt.Sprintf("%s:%s", n, v) }
58
59 rootID := id(componentName, componentVersion)
60
61 visited := make(map[string]bool)
62 queue := []compKey{{Name: componentName, Version: componentVersion}}
63
64 // Store descriptors, child relations, reverse parent relations, and
65 // per-node SBOM paths
66 descriptors := make(map[string]interface{}) // kept for future use;
67 // not needed for name/version
68 childrenOf := make(map[string][]string) // parentID -> []childID
69 parentsOf := make(map[string]map[string]void) // childID -> set(
70 // parentID)
71 resourceSBOMPath := make(map[string]string) // nodeID -> path of SBOM
72 // for that node's resources
73 resultPath := make(map[string]string) // nodeID -> merged (node
74 // + subtree) SBOM path
75 remainingChildren := make(map[string]int) // nodeID -> remaining
76 // children to be processed
77 allNodes := make(map[string]void)
78
79 ctx := context.Background()
80
81 // 1) Build graph (BFS) and generate resource-only SBOMs for each
82 // component
83 for len(queue) > 0 {
```

```
73     curr := queue[0]
74     queue = queue[1:]
75     currID := id(curr.Name, curr.Version)
76     if visited[currID] {
77         continue
78     }
79     visited[currID] = true
80
81     log.Printf("Processing component (graph build): %s", currID)
82     runtimeDesc, err := repo.GetComponentVersion(ctx, curr.Name, curr.
83     Version)
84     if err != nil {
85         log.Printf("Warning: could not get component version for %s: %v",
86         currID, err)
87         // Skip if descriptor cannot be obtained
88         continue
89     }
90
91     // Generate and store resource-only SBOM for this component
92     mergedSBOMPath, err := processor.ProcessComponent(runtimeDesc,
93     outputFormat, mergeTool)
94     if err != nil {
95         log.Printf("Warning: error processing component %s: %v", currID,
96         err)
97         // Keep going; if no SBOM, children might still produce results
98     }
99
100    // Remember descriptor and children
101    descriptors[currID] = runtimeDesc
102    allNodes[currID] = struct{}{}
103
104    for _, ref := range runtimeDesc.Component.References {
105        childID := id(ref.Component, ref.Version)
106        childrenOf[currID] = append(childrenOf[currID], childID)
107
108        if _, ok := parentsOf[childID]; !ok {
109            parentsOf[childID] = make(map[string]void)
110        }
111        parentsOf[childID][currID] = struct{}{}
112
113        if !visited[childID] {
```

```
114         queue = append(queue, compKey{Name: ref.Component, Version: ref.
115                         Version})
116     }
117 }
118
119 if len(allNodes) == 0 {
120     log.Printf("No components discovered from root %s:%s", componentName,
121               componentVersion)
122     return nil, nil
123 }
124
125 // Initialize remaining children counts
126 for nid := range allNodes {
127     remainingChildren[nid] = len(childrenOf[nid])
128 }
129
130 // 2) Bottom-up merging: process leaves first, propagate upwards
131 processed := make(map[string]bool)
132 batch := make([]string, 0)
133 for nid := range allNodes {
134     if remainingChildren[nid] == 0 {
135         batch = append(batch, nid)
136     }
137 }
138
139 for len(processed) < len(allNodes) && len(batch) > 0 {
140     next := make([]string, 0)
141     for _, nid := range batch {
142         if processed[nid] {
143             continue
144         }
145         // Files to merge: this component's resource SBOM + all children's
146         // merged SBOMs
147         files := make([]string, 0, 1+len(childrenOf[nid]))
148         if p := resourceSBOMPath[nid]; p != "" {
149             files = append(files, p)
150         }
151         for _, cid := range childrenOf[nid] {
152             if cp := resultPath[cid]; cp != "" {
153                 files = append(files, cp)
154             }
155         }
156         // If no files collected, nothing to produce
```

```
157     if len(files) == 0 {
158         log.Printf("Warning: no SBOM inputs collected for %s; skipping
159         merge for this node", nid)
160         processed[nid] = true
161         for parentID := range parentsOf[nid] {
162             remainingChildren[parentID]--
163             if remainingChildren[parentID] == 0 && !processed[parentID] {
164                 next = append(next, parentID)
165             }
166             continue
167         }
168
169         // If only one file, it is already the final SBOM for this node
170         finalPath := files[0]
171         if len(files) > 1 {
172             // Derive component name/version from node ID "name:version" to
173             // avoid invalid type assertion
174             compName, compVersion := "", ""
175             if parts := strings.SplitN(nid, ":", 2); len(parts) == 2 {
176                 compName, compVersion = parts[0], parts[1]
177             } else if len(parts) == 1 {
178                 compName = parts[0]
179             }
180
181             outPath, err := merger.ComponentSbomMerge(c.TempDir, files,
182             mergeTool, compName, compVersion)
183             if err != nil {
184                 log.Printf("Warning: merge failed for %s, using first input: %v
185                 ", nid, err)
186             } else {
187                 finalPath = outPath
188             }
189         }
190
191         resultPath[nid] = finalPath
192         processed[nid] = true
193
194         // Notify parents
195         for parentID := range parentsOf[nid] {
196             remainingChildren[parentID]--
197             if remainingChildren[parentID] == 0 && !processed[parentID] {
198                 next = append(next, parentID)
199             }
200         }
201     }
```

```
199     batch = next
200 }
201
202 // 3) The root's merged SBOM already contains the full hierarchy;
203 //      return it first.
203 rootMerged := resultPath[rootID]
204 if rootMerged == "" {
205     // Fallback: if root had no result, return any path we created
206     for _, p := range resultPath {
207         rootMerged = p
208         break
209     }
210 }
211 if rootMerged == "" {
212     log.Printf("No merged SBOM produced for root %s", rootID)
213     return nil, nil
214 }
215
216 // Keep return type the same; ensure the root SBOM is first.
217 return []string{rootMerged}, nil
218 }
219
220 // processAllComponentsOld traverses the component hierarchy and
221 // generates a merged SBOM for each component.
221 func (c *CLIConverter) processAllComponentsOld(repo oci.
222     ComponentVersionRepository, componentName, componentVersion string,
223     outputFormat SBOMFormat, mergeTool string) ([]string, error) {
222 var allComponentSBOMPaths []string
223 processed := make(map[string]bool)
224 queue := []struct{ ComponentName, Version string }{{ComponentName:
225     componentName, Version: componentVersion}}
226
226 processor := NewComponentProcessor(c)
227
228 for len(queue) > 0 {
229     curr := queue[0]
230     queue = queue[1:]
231     id := fmt.Sprintf("%s:%s", curr.ComponentName, curr.Version)
232     if processed[id] {
233         log.Printf("Component %s already processed, skipping", id)
234         continue
235     }
236
237     log.Printf("Processing component: %s", id)
238     runtimeDesc, err := repo.GetComponentVersion(context.Background(),
239         curr.ComponentName, curr.Version)
```

```
239     if err != nil {
240         log.Printf("Warning: could not get component version for %s: %v",
241             id, err)
242         processed[id] = true
243         continue
244     }
245
246     // Process the current component using the runtime descriptor
247     mergedSBOMPath, err := processor.ProcessComponent(runtimeDesc,
248             outputFormat, mergeTool)
249     if err != nil {
250         log.Printf("Warning: error processing component %s: %v", id, err)
251     }
252     if mergedSBOMPath != "" {
253         allComponentSBOMPaths = append(allComponentSBOMPaths,
254             mergedSBOMPath)
255         log.Printf("SBOM for component %s at %s", id, mergedSBOMPath)
256     }
257
258     processed[id] = true
259
260     // Enqueue referenced components using fully-qualified component name
261     // from runtime descriptor
262     for _, ref := range runtimeDesc.Component.References {
263         refID := fmt.Sprintf("%s:%s", ref.Component, ref.Version)
264         if !processed[refID] {
265             log.Printf("Queue referenced component %s", refID)
266             queue = append(queue, struct{ ComponentName, Version string }{
267                 ComponentName: ref.Component, Version: ref.Version})
268         }
269     }
270
271     return allComponentSBOMPaths, nil
272 }
273
274 // convertFinalSBOM converts the final SBOM into the target format when
275 // required.
276 func (c *CLIConverter) convertFinalSBOM(sourceSBOMPath string,
277             targetFormat SBOMFormat) ([]byte, error) {
278     // If CycloneDX JSON is desired, we can just read the merged file
279     // directly.
280     if strings.ToLower(string(targetFormat)) == "cyclonedx-json" {
281         log.Println("No format conversion needed; reading merged SBOM
282             directly")
283         return os.ReadFile(sourceSBOMPath)
```

```
276     }
277
278     if c.CycloneDXCLIPath == "" {
279         return nil, fmt.Errorf("CycloneDX CLI path not set or found, but
280         required for conversion")
281     }
282
283     convertedSBOMPath := filepath.Join(c.TempDir, "finalConverted_sbom.
284         json")
285     outputFormatArg := string(targetFormat)
286     outputVersion := "1.6" // default for CycloneDX
287
288     if strings.HasSuffix(outputFormatArg, "-yaml") {
289         convertedSBOMPath = filepath.Join(c.TempDir, "finalConverted_sbom.
290         yaml")
291         outputFormatArg = "yaml"
292     } else if strings.HasSuffix(outputFormatArg, "-json") {
293         outputFormatArg = "json"
294     }
295
296     if strings.HasPrefix(string(targetFormat), "spdx") {
297         outputVersion = "2.3" // default for SPDX
298     }
299
300     args := []string{
301         "convert",
302         "--input-file", sourceSBOMPath,
303         "--output-file", convertedSBOMPath,
304         "--output-format", outputFormatArg,
305         "--output-version", outputVersion,
306     }
307     log.Printf("Running CycloneDX CLI convert: %s %s",
308             c.CycloneDXCLIPath,
309             strings.Join(args, " "))
310     if _, err := c.runCommand(c.CycloneDXCLIPath, args...); err != nil {
311         return nil, fmt.Errorf("failed to convert SBOM to %s: %w",
312             targetFormat, err)
313     }
314
315     return os.ReadFile(convertedSBOMPath)
316 }
```

Listing 10.1: Quellcode convert_impl.go

Literaturverzeichnis

- Aghamanoukjan, Anahid u. a. (2009). „Qualitative Interviews“. Deutsch. In: *Qualitative Marktorschung: Konzepte – Methoden – Analysen*. Hrsg. von Renate Buber und Hartmut H. Holzmüller. Wiesbaden: Gabler, S. 415–436. DOI: 10.1007/978-3-8349-9441-7_26. URL: https://doi.org/10.1007/978-3-8349-9441-7_26 (besucht am 18.09.2025).
- Balliu, Musard u. a. (Nov. 2023). „Challenges of Producing Software Bill Of Materials for Java“. Englisch. In: *IEEE Security & Privacy* 21.6. arXiv:2303.11102 [cs], S. 12–23. DOI: 10.1109/MSEC.2023.3302956. URL: <http://arxiv.org/abs/2303.11102> (besucht am 08.09.2025).
- Bogner, Alexander u. a. (2014). *Interviews mit Experten: Eine praxisorientierte Einführung*. Deutsch. Wiesbaden: Springer Fachmedien. DOI: 10.1007/978-3-531-19416-5. URL: <https://link.springer.com/10.1007/978-3-531-19416-5> (besucht am 18.09.2025).
- Bowen, Glenn A. (Aug. 2009). „Document Analysis as a Qualitative Research Method“. Englisch. In: *Qualitative Research Journal* 9.2, S. 27–40. DOI: 10.3316/QRJ0902027. URL: <http://www.emerald.com/qrj/article/9/2/27-40/360733> (besucht am 17.09.2025).
- Brooks Jr, Frederick (Apr. 1987). „No Silver Bullet—Essence and Accidents of Software Engineering“. Englisch. In: *IEEE Computer* 20, S. 10–19. DOI: 10.1109/MC.1987.1663532.
- Carmody, Seth u. a. (Feb. 2021). „Building resilient medical technology supply chains with a software bill of materials“. Englisch. In: *npj Digital Medicine* 4.1. Publisher: Nature Publishing Group, S. 34. DOI: 10.1038/s41746-021-00403-w. URL: <https://www.nature.com/articles/s41746-021-00403-w> (besucht am 19.08.2025).
- CISA (Dez. 2021). *Mitigating Log4Shell and Other Log4j-Related Vulnerabilities*. Englisch. URL: <https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-356a> (besucht am 22.09.2025).
- (Aug. 2025). „Minimum Elements for a Software Bill of Materials (SBOM)“. Englisch. In.
- CycloneDX Core Working Group (Sep. 2024). *Authoritative Guide to SBOM (Second Edition)*. Englisch. Technical Guide. OWASP Foundation, S. 75. URL: https://cyclonedx.org/guides/OWASP_CycloneDX-Authoritative-Guide-to-SBOM-en.pdf (besucht am 05.09.2025).
- Ecma Technical Committee 54 (Mai 2024). *ECMA-424: CycloneDX Bill of Materials (BOM) Standard*. Englisch. BOM Standard “CycloneDX”. Geneva, Switzerland. URL: <https://ecma-international.org/publications-and-standards/standards/ecma-424/> (besucht am 10.09.2025).

- European Parliament and Council (Dez. 2022). *Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cybersecurity across the Union, amending Regulation (EU) No 910/2014 and Directive (EU) 2018/1972, and repealing Directive (EU) 2016/1148*. Englisch. URL: <https://eur-lex.europa.eu/eli/dir/2022/2555/2022-12-27/eng> (besucht am 08.09.2025).
- (Okt. 2024). *Cyber Resilience Act – Cybersicherheit EU-weit gedacht*. Englisch. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/Management_Blitzlicht/Management_Blitzlicht_CRA.pdf.
- Flick, Uwe (2009). *An introduction to qualitative research*. Englisch. 4. ed., repr. Los Angeles, Calif.: SAGE Publications Ltd.
- Freeman, Robert Edward (1984). *Strategic Management: A Stakeholder Approach*. Englisch. Pitman series in business and public policy. Boston [u.a.] : Pitman.
- Hendrick, Stephen (Jan. 2022). *Software Bill of Materials (SBOM) and Cybersecurity Readiness*. Englisch. Techn. Ber. The Linux Foundation. DOI: 10.70828/RYTL5793. URL: <https://www.linuxfoundation.org/research/the-state-of-software-bill-of-materials-sbom-and-cybersecurity-readiness> (besucht am 16.08.2025).
- Herrmann, Andrea (2022). „Ermitteln von Anforderungen“. Deutsch. In: *Grundlagen der Anforderungsanalyse: Standardkonformes Requirements Engineering*. Hrsg. von Andrea Herrmann. Wiesbaden: Springer Fachmedien, S. 25–80. DOI: 10.1007/978-3-658-35460-2_4. URL: https://doi.org/10.1007/978-3-658-35460-2_4 (besucht am 18.09.2025).
- Hevner u. a. (2004). „Design Science in Information Systems Research“. Englisch. In: *MIS Quarterly* 28.1, S. 75. DOI: 10.2307/25148625. URL: <https://www.jstor.org/stable/10.2307/25148625> (besucht am 17.09.2025).
- ISO/IEC/IEEE 29148 (Nov. 2018). *ISO/IEC/IEEE 29148:2018*. Englisch. (Besucht am 19.09.2025).
- Jaatun, Lars Andreassen u. a. (Dez. 2023). „Software Bill of Materials in Critical Infrastructure“. Englisch. In: *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Naples, Italy: IEEE, S. 319–324. DOI: 10.1109/CloudCom59040.2023.00059. URL: <https://ieeexplore.ieee.org/document/10475810/> (besucht am 19.08.2025).
- Kağızmandere, Ömercan und Halil Arslan (Juni 2024). „Vulnerability analysis based on SBOMs: A model proposal for automated vulnerability scanning for CI/CD pipelines“. Englisch. In: *International Journal of Information Security Science* 13.2, S. 33–42. DOI: 10.55859/ijiss.1455039. URL: <http://dergipark.org.tr/en/doi/10.55859/ijiss.1455039> (besucht am 19.08.2025).

- Kaiser, Robert (2021). *Qualitative Experteninterviews: Konzeptionelle Grundlagen und praktische Durchführung*. Deutsch. Elemente der Politik. Wiesbaden: Springer Fachmedien. DOI: 10.1007/978-3-658-30255-9. URL: <https://link.springer.com/10.1007/978-3-658-30255-9> (besucht am 19.09.2025).
- Kano, Noriaki u. a. (1984). „Attractive Quality and Must-Be Quality“. In: *The Journal of the Japanese Society for Quality Control* 14.2, S. 39–48.
- Kober, Ingo (Nov. 2023). *Embracing the Future of Software Delivery: The Open Component Model in the Era of DevSecOps*. Englisch. Section: DevOps and System Administration Blogs. URL: <https://community.sap.com/t5/devops-and-system-administration-blogs/embracing-the-future-of-software-delivery-the-open-component-model-in-the/ba-p/13580219> (besucht am 15.09.2025).
- Linux Foundation (2025). *The Linux Foundation Announces the Launch of NeoNephos to Advance Digital Autonomy in Europe*. Englisch. URL: <https://linuxfoundation.eu/newsroom/the-linux-foundation-announces-the-launch-of-neonephos-to-advance-digital-autonomy-in-europe> (besucht am 15.09.2025).
- Mayring, Philipp (2015). *Qualitative Inhaltsanalyse: Grundlagen und Techniken*. Deutsch. 12., vollständig überarbeitete und aktualisierte Aufl. Beltz Pädagogik. Weinheim: Beltz.
- Mens, Tom (Aug. 2016). *Research trends in structural software complexity*. Englisch. arXiv:1608.01533 [cs]. DOI: 10.48550/arXiv.1608.01533. URL: <http://arxiv.org/abs/1608.01533> (besucht am 12.09.2025).
- Meuser, Michael und Ulrike Nagel (2009). „Das Experteninterview — konzeptionelle Grundlagen und methodische Anlage“. Deutsch. In: *Methoden der vergleichenden Politik- und Sozialwissenschaft*. Hrsg. von Susanne Pickel u. a. Wiesbaden: VS Verlag für Sozialwissenschaften, S. 465–479. DOI: 10.1007/978-3-531-91826-6_23. URL: http://link.springer.com/10.1007/978-3-531-91826-6_23 (besucht am 17.09.2025).
- Mitchell, Ronald K. u. a. (Okt. 1997). „Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts“. Englisch. In: *The Academy of Management Review* 22.4, S. 853. DOI: 10.2307/259247. URL: <http://www.jstor.org/stable/259247?origin=crossref> (besucht am 16.09.2025).
- National Vulnerability Database (2025). *CVE-2021-44228 Detail*. Englisch. URL: <https://nvd.nist.gov/vuln/detail/CVE-2021-44228> (besucht am 22.09.2025).
- NTIA (Dez. 2021). *The Minimum Elements For a Software Bill of Materials (SBOM)*. Englisch. Guidance Document Executive Order 14028. Washington, D.C.: U.S. Department of Commerce, National Telecommunications und Information Administration, S. 11. URL: <https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom> (besucht am 08.09.2025).

- NTIA Framing Working Group (Okt. 2021). *Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM) - Second Edition*. Englisch. URL: https://www.ntia.gov/files/ntia/publications/ntia_sbom_framing_2nd_edition_20211021.pdf (besucht am 01.09.2025).
- NTIA Standards and Formats Working Group (Okt. 2019). *Survey of SBOM Formats and Standards*. Englisch. URL: https://www.ntia.gov/files/ntia/publications/ntia_sbom_formats_and_standards_whitepaper_-_version_20191025.pdf?utm_source=chatgpt.com (besucht am 10.09.2025).
- O'Donoghue, Eric u. a. (Juni 2025). *Software Bill of Materials in Software Supply Chain Security A Systematic Literature Review*. Englisch. arXiv:2506.03507 [cs]. DOI: 10.48550/arXiv.2506.03507. (Besucht am 10.09.2025).
- OCM Working Group (2025a). *Open Component Model*. Englisch. URL: <https://ocm.software/> (besucht am 11.09.2025).
- (2025b). *Open Component Model Specification*. Englisch. URL: <https://github.com/open-component-model/ocm-spec> (besucht am 11.09.2025).
- OWASP Foundation (Nov. 2024). *Specification Overview / CycloneDX*. Englisch. URL: <https://cyclonedx.org/specification/overview/> (besucht am 10.09.2025).
- (2025a). *Software Bill of Materials (SBOM) / CycloneDX*. Englisch. URL: <https://cyclonedx.org/capabilities/sbom/> (besucht am 10.09.2025).
- (2025b). *CycloneDX Bill of Materials Standard / CycloneDX*. Englisch. URL: <https://cyclonedx.org/> (besucht am 08.09.2025).
- Rupp, Chris und SOPHIST (2021). *Requirements-Engineering und -Management: das Handbuch für Anforderungen in jeder Situation*. Deutsch. 7., aktualisierte und erweiterte Auflage. Hanser eLibrary. München: Hanser. DOI: 10.3139/9783446464308.
- SPDX Workgroup (2025). *Software Package Data Exchange*. Englisch. URL: <https://spdx.org/> (besucht am 10.09.2025).
- U.S. Office of the Director of National Intelligence u. a. (Dez. 2023). *Securing the Software Supply Chain: Recommended Practices for Software Bill of Materials Consumption*. Englisch.
- Wieringa, Roel J. (2014). „Research Goals and Research Questions“. Englisch. In: *Design Science Methodology for Information Systems and Software Engineering*. Hrsg. von Roel J. Wieringa. Berlin, Heidelberg: Springer, S. 13–23. DOI: 10.1007/978-3-662-43839-8_2. URL: https://doi.org/10.1007/978-3-662-43839-8_2 (besucht am 14.07.2025).
- Xia, Boming u. a. (Feb. 2023). *An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead*. Englisch. arXiv:2301.05362 [cs]. DOI: 10.48550/arXiv.2301.05362. URL: <http://arxiv.org/abs/2301.05362> (besucht am 19.08.2025).

- Zahan, Nusrat (Juli 2023). „Software Supply Chain Risk Assessment Framework“. Englisch. In: *Proceedings of the 45th International Conference on Software Engineering: Companion Proceedings*. ICSE '23. Melbourne, Victoria, Australia: IEEE Press, S. 251–255. DOI: [10.1109/ICSE-Companion58688.2023.00068](https://doi.org/10.1109/ICSE-Companion58688.2023.00068). (Besucht am 10.09.2025).

Verzeichnis sonstiger Quellen

- Anchore (Sep. 2025). *Grype Vulnerability Scanner*. URL: <https://github.com/anchore/grype> (besucht am 10.09.2025).
- Aqua Security Software Ltd. (Mai 2021). *Trivy Open Source Vulnerability Scanner*. URL: <https://www.aquasec.com/products/trivy/> (besucht am 10.09.2025).
- Bressers, Josh (Sep. 2025). *Experteninterview über Scanner (Grype)*. Englisch.
- Hohage, Karsten u. a. (Aug. 2023). *Open Component Model (OCM) – Describe, Transport, Deploy - The Open Source Way*. Englisch. URL: <https://podcast.opensap.info/open-source-way/2023/08/30/open-component-model-ocm-describe-transport-deploy/> (besucht am 11.09.2025).
- Linux Foundation (Mai 2023). *Delivering Secure & Compliant Software Components with the Open Component Model & GitOps - Dan Small*. Englisch. URL: <https://www.youtube.com/watch?v=LBD4EYDY1CU> (besucht am 11.09.2025).
- Möller, Jakob (Sep. 2025a). *Experteninterview zum Open Component Model*. Deutsch.
- (31. Juli 2025b). *Hierarchical SBOM Merging from OCM Resources and Component References*. GitHub issue. Repository: open-component-model/ocm-project; Status: closed. open-component-model. URL: <https://github.com/open-component-model/ocm-project/issues/572> (besucht am 23.09.2025).
- Springett, Josh (Sep. 2025). *Experteninterview zu CycloneDX*. Englisch.

Ehrenwörtliche Erklärung zur Selbständigkeit

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema

Von der Software Bill of Delivery (SBoD) zur Software Bill of Materials (SBOM)

Ein *Proof of Concept* zur Konvertierung von Open Component Model Deskriptoren in den CycloneDX SBOM Standard

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

23.09.2025

Ort, Datum



Unterschrift

Erklärung und Nutzungsdokumentation zum Einsatz von KI-basierten Werkzeugen

Erklärung

Zur Verwendung KI-gestützter Werkzeuge erkläre ich in Kenntnis des Hinweisblatts „Hinweise zum Einsatz von KI-basierten Werkzeugen bei der Anfertigung wissenschaftlicher Arbeiten, u.a. im prüfungsrechtlichen Kontext“ Folgendes:

- Ich habe mich aktiv über die Leistungsfähigkeit und Beschränkungen der in meiner Arbeit eingesetzten KI-Werkzeuge informiert.
- Bei der Anfertigung der Arbeit habe ich durchgehend eigenständig und beim Einsatz KI-gestützter Werkzeuge maßgeblich steuernd gearbeitet.
- Insbesondere habe ich die Inhalte entweder aus wissenschaftlichen oder anderen zugelassenen Quellen entnommen und diese gekennzeichnet oder unter Anwendung wissenschaftlicher Methoden selbst entwickelt.
- Mir ist bewusst, dass ich als Autor/in der Arbeit die volle Verantwortung für die in ihr gemachten Angaben und Aussagen trage.
- Soweit ich KI-gestützte Werkzeuge zur Erstellung der Arbeit eingesetzt habe, sind diese jeweils mit dem Produktnamen, den formulierten Eingaben (Prompts), der Einsatzform sowie der entsprechenden Seiten- bzw. Bereichsreferenzierung auf die Arbeit im KI-Verzeichnis am Ende der Arbeit vollständig ausgewiesen und im Text belegt (z.B. als Fußnote).

23.09.2025

Ort, Datum



Unterschrift der/des Studierenden

KI-Verzeichnis

Das KI-Verzeichnis ist in der vorgegebenen tabellarischen Form zu führen und abzugeben. Es empfiehlt sich, das Verzeichnis bereits ab Beginn des Erarbeitens und Schreibens der Arbeit fortlaufend zu führen.

KIgestütztes Werkzeug	Prompts (Kurzbeschreibung)	Einsatzform	Seiten
DeepL Write	Kein Prompt	Rechtschreibung, Grammatik und leichte Stilglättung finaler eigener Texte	Gesamte Arbeit
DeeperWrite	Kein Prompt	Stilglättung/-Lesbarkeitscheck einzelner Abschnitte	Gesamte Arbeit
GPT-5	„Mache mir mehrere Vorschläge für Forschungsfragen zu meiner Arbeit.“	Ideengenerierung/Strukturierung (Forschungsfragen)	S. 1–2
GPT-5	„Erstelle mir eine Gliederung für mein Thema: {Thema}.“	Strukturierung (Brainstorming zur Gliederung; Auswahl/Überarbeitung durch Autor)	Gliederung
GPT-5 Thinking	„Was hältst du von meiner Arbeit, wo siehst du Verbesserungspotenzial: {Doc}?“	Kritisches Lektorat/Feedback (inhaltliche Hinweise, Kohärenz)	Gesamte Arbeit
GPT-5 Thinking	„Bitte vor Abgabe prüfen: Showstopper/-Prüfungsrisiken + Noteinschätzung: {Doc}“	Strukturierend/überarbeitend (Checkliste, Diffs, Priorisierung)	Gesamte Arbeit