修士学位論文

題目

SBOM 利活用に関する課題抽出及び C/C++ のソフトウェアに対する SBOM 生成手法の開発

指導教員
肥後 芳樹 教授

報告者
音田 渉

令和 7 年 1 月 28 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

令和 6 年度 修士学位論文

SBOM 利活用に関する課題抽出及び C/C++ のソフトウェアに対する SBOM 生成手法の開発

音田 渉

内容梗概

現在のソフトウェア開発では多くの外部ライブラリを活用するが, サイバーセキュリティ上や著作権法上のリスクが伴う. 対策にあたっては依存関係の適切な管理が重要となるため, ソフトウェア部品表（SBOM; Software Bill of Materials）の活用が奨励されているが, その普及は未だ不十分である.

本研究では，SBOM 普及を妨げる原因を調査し, そこで明らかとなった課題を解決する手法を開発した. まず主要な開発者向け Q&A サイトである Stack Overflow において SBOM 利活用に関する質問を調査した結果, 一般の開発者が SBOM を利用するにあたり実際に直面した課題は「SBOM ツールのユースケースの網羅性に不足がある」,「各種要件を満たす SBOM を生成できない」,「SBOM ツールに不具合がある, または利用方法が不明瞭である」の 3 つであった. そこで, 既存手法ではサポートされないユースケースである C/C++ で開発されたソフトウェアに対し, 米国政府機関 NTIA が定める要件を満たす SBOM を自動生成する手法を開発した. 既存手法がパッケージマネージャの持つ依存情報やメタデータに依存する中で, パッケージマネージャが使用されない C/C++ を対象とするにあたり, ビルド処理から得られる情報と, その結果生成されるバイナリに含まれるメタデータに着目した. 本手法を著名なネットワークツール curl と Web サーバソフトウェア Apache HTTP Server に適用したところ，curl ではビルド時に組み込まれる依存関係 9 件と実行時の依存関係 27 件を特定し，Apache HTTP Server ではそれぞれ 17 件と 7 件を特定し, 全てについて NTIA 要件を満たす SBOM を実用的な実行時間で生成できた. また, 本手法のメタデータ抽出能力の汎用性を検証するため，Ubuntu サーバ，Debian 仮想マシン，Linux Mint 仮想マシンにインストールされた全パッケージ 984 件，1677 件，2523 件についてメタデータ抽出処理を適用したところ, 全パッケージについて NTIA 要件を満たす SBOM を生成できたことから, 高い汎用性を持つことを示した.

主な用語

SBOM
SPDX
Software Supply Chain
Stack Overflow
C/C++

目次

1 はじめに

現在のソフトウェア開発は，実現する機能を全て自力で実装するのではなく，多くの外部ライブラリを活用して行われる．これにより，開発費用の削減や開発期間の短縮に加え，既に洗練されているソフトウェア資源を活用し，堅牢かつ高機能なソフトウェアを開発できる [24, 36]．実際に，調査した 1067 件のコードベースのうち 96% がオープンソース由来のコードを含んでおり，スキャンしたソースコードやファイルのうち 77% がオープンソース由来だったという報告がある [5]．一方で，外部ライブラリの利用にはサプライチェーンリスクと呼ばれる，サイバーセキュリティ上や著作権法上のリスクが伴う．

1.1 サイバーセキュリティ上のリスク

開発しているソフトウェアが依存する外部ライブラリに脆弱性が見つかった場合，それが依存元にも波及して悪意のある利用者による攻撃の対象となるリスクがある [44]．また，開発しているソフトウェアが依存する外部ライブラリの開発者が悪意のある処理を埋め込むリスクや，外部ライブラリの開発者が悪意を持った人物による乗っ取り被害にあうリスクも存在する [32]．

実際に，SolarWinds 社製ソフトウェア Orion の更新データにサプライチェーン攻撃によってマルウェアが埋め込まれ，Orion を利用する多数の米国省庁や民間企業がサイバー攻撃を受ける事件が 2020 年に発生した [45] ほか，サーバの遠隔管理などに利用されるソフトウェア OpenSSH が依存するライブラリ XZ Utils に対し，悪意を持った開発者が OpenSSH で発動するバックドアを埋め込む事件も 2024 年に発生した [7]．

1.2 著作権法上のリスク

ソフトウェアには著作権が存在し，各開発者が利用条件を定めてライセンスしているが，その条件の把握漏れにより条件に反した利用を行うことや，互いに条項が衝突するライセンスが設定されたライブラリを導入することなどにより，著作権侵害となる法的リスクも存在する．

実際に，2002 年に EPSON KOWA 社がリリースした Linux 用高品質プリンタドライバ及びスキャナドライバについて，GPL と非 GPL のソースコード及び非公開のバイナリについて個々のライセンスを明確にせずに 2 次配布を行ったことにより，第 3 者から GPL の条件と異なっているとの指摘を受けて対応を行った例や，2017 年に Panasonic Avionics 社が開発した航空機内エンターテイメント（IFE）ソフトウェアについて，米国の CoKinetic Systems 社から GPL 違反による賠償を求める訴訟を提起された例がある [6]．

1.3 サプライチェーンリスクへの対応策

サプライチェーンリスクは早期の把握と対応が重要だが，現在のソフトウェア開発で利用する外部ライブラリの数は，その外部ライブラリが依存する別のライブラリのような推移的な依存関係も考慮すると膨大となり [21]，適切な管理は難しい．この問題に対処するために，ソフトウェア部品表（SBOM;

Software Bill of Materials)の活用が奨励されているが，SBOM の普及は未だ不十分だと指摘されている [31, 46].

そこで，SBOM 普及を妨げる原因を明らかにするため，まず主要な開発者向け Q&A サイトである Stack Overflow において SBOM 利活用に関する質問を調査した．その結果，一般の開発者が SBOM を利用するにあたり実際に直面した課題は「SBOM ツールのユースケースの網羅性に不足がある」，「各種要件を満たす SBOM を生成できない」，「SBOM ツールに不具合がある，または利用方法が不明瞭である」の 3 つであることを明らかにした．

この調査結果を受けて，SBOM 普及に資するため，既存ツールではサポートされないユースケースである C 言語及び C++ 言語で開発されたソフトウェアに対し，米国政府機関 NTIA が定める要件 [42] を満たす SBOM を自動生成する手法を開発した．パッケージマネージャが存在しない C/C++ のソフトウェアに対して SBOM を生成するにあたり，ビルド処理から得られる情報と，その結果生成されるバイナリに含まれるメタデータに着目した．本手法を著名なネットワークツール curl と Web サーバソフトウェア Apache HTTP Server に適用したところ，curl ではビルド時に組み込まれる依存関係 9 件と実行時の依存関係 27 件を特定し，Apache HTTP Server ではそれぞれ 17 件と 7 件を特定し，全てについて NTIA 要件を満たす SBOM を実用的な実行時間で生成できたほか，要件外の項目も高い水準で抽出できた．また，本手法のメタデータ抽出能力の汎用性を検証するため，Ubuntu サーバ，Debian 仮想マシン，Linux Mint 仮想マシンにインストールされた全パッケージ 984 件，1677 件，2523 件についてメタデータ抽出処理を適用したところ，全パッケージについて NTIA 要件を満たす SBOM を生成でき，NTIA 要件外のメタデータについても高い水準で抽出できたことから，本手法のパッケージメタデータ抽出処理は高い汎用性を持ち，NTIA 要件を超える有用性の高い SBOM を生成できることを示した．一方，本手法では NTIA 要件外である著作権情報に関して，機械可読形式で記述されたものだけを解析対象としているため，`copyrightText` 項目をはじめとする著作権情報の抽出に不十分さが残る．機械可読形式でない情報も解析対象とする手法を開発し，`copyrightText` 項目やライセンス情報を拡充することが，今後の課題である．

2 背景

本節では，ソフトウェア部品表（SBOM）に関する概要，各国の SBOM に関する動向，SBOM の構成要素，そして SBOM に関する既存研究と既存ツールに関して述べ，現状の到達点及び課題を示す．

2.1 ソフトウェア部品表（SBOM）

SBOM は，ソフトウェアの構築に用いられるライブラリなどのソフトウェア部品の正式かつ機械可読な表であり，各ソフトウェア部品のライセンス・バージョン・ベンダなどの詳細や，部品間のサプライチェーン関係の情報を含む [18,42]．SBOM に記述された情報を活用すると，利用しているソフトウェアの依存情報から，1 節で述べたサプライチェーンリスクを迅速に検出・対応できる [22]．

SBOM を表現する方法として，最も主要なものは SPDX と CycloneDX の 2 つの形式である [2]．いずれの形式も，実際に使用するファイルフォーマットとしては JSON (JavaScript Object Notation) や XML (Extensible Markup Language) などを用いる．

SPDX (System Package Data Exchange)

Linux Foundation が策定する ISO/IEC 5962:2021 国際標準規格であり，開発フローや業務におけるコンプライアンス，透明性確保を念頭に設計されている [25]．

CycloneDX

サイバーセキュリティを専門に取り扱う団体 OWASP が策定する ECMA-424 国際標準規格であり，ソフトウェア部品，外部サービス，それらの関係性を表現可能であるなど，セキュリティやサプライチェーン部品の解析を念頭に設計されている [33]．

2.2 各国の SBOM に関する動向

近年，各国政府機関が SBOM 利用を推進するなど，急速に注目が集まっている．

2.2.1 米国

米国では，Biden 政権が大統領令 Executive Order on Improving the Nation's Cybersecurity を 2021 年に発令し，SBOM 利用を推進している [4]．具体的には，NIST (National Institute of Standards and Technology) に「ソフトウェア製品の購入者に SBOM を提供すること」などを内容として含むサプライチェーンセキュリティのための綱領作成を指示し，OMB (Office of Management and Budget) に対してこの綱領を各政府機関で遵守させるよう指示するものである．これを受けて，政府との取引を行う連邦政府省庁および請負業者は，SBOM の利用・提供が求められるようになっている．2025 年の Biden 政権から Trump 政権への移行に伴い，Biden 政権下で発行された大統領令の大規模廃止が実行されたが，本大統領令 14028 はその対象となっていない [43]．

2.2.2 EU

EU では，ENISA (European Union Agency for Cybersecurity) が IoT セキュリティにおける SBOM 利用を推奨する Guidelines for Securing the Internet of Things を公開した [41]. また，SBOM 利用を義務化する Cyber Resilience Act も 2024 年に発効した [14]．Cyber Resilience Act については，ほとんどの内容が 2027 年から適用される旨が条文内で定められている．

2.2.3 日本国内

日本では，経済産業省が「ソフトウェア管理に向けた SBOM の導入に関する手引」の初版を 2023 年に公開し [19]，改訂版を 2024 年に公開した [20]．SBOM に関する基本的な情報や実際に導入するにあたって認識・実施すべきポイントが示されており，現時点では法令は整備されていないが，SBOM に対する国内での認識の広がりを伺える．

2.3 SBOM の構成要素

SPDX や CycloneDX といった規格は形式を定めるのみで，特に含めるべき項目は規定していない．しかし，実際にサプライチェーンリスク対策で活用するためには一定の基準が必要である．この基準について，現時点では国際的な標準規格が存在しないが，米国の政府機関 NTIA (National Telecommunications and Information Administration) が先述の大統領令を受けて発行した文書 [42] で SBOM に含めるべき要素の最小要件を定めている．SPDX の Web サイトで準拠度を確認する NTIA Conformance Checker ツール[*1]が提供されているなど，この文書が事実上の標準となっている．具体的な事項と，現在の ISO/IEC 国際標準規格である SPDX v2 における項目との対応関係を表 1 に示す．SPDX の項目名は JSON スキーマに従っている．

表 1　The Minimum Elements for an SBOM

要素	概要	SPDX における項目
Supplier Name	部品を作成・定義・特定した者	packages > supplier
Component Name	供給者によって部品に付けられた名前	packages > name
Version of the Component	旧版との区別のために供給者が設定した識別子	packages > versionInfo
Other Unique Identifiers	部品の特定や DB 検索キーに使われる識別子 例：CPE, SWID tag, PURL	packages > externalRefs
Dependency Relationship	部品 X がソフトウェア Y に含まれるなどの依存関係	relationships
Author of SBOM Data	SBOM データを作成した者	creationInfo > creators
Timestamp	SBOM データを作成した日時	creationInfo > created

[*1] https://tools.spdx.org/app/ntia_checker/

2.4 既存調査

本節では，SBOM の現状に関する調査を通じて原因や改善案を示した研究を紹介する．

Linux Foundation は，2022 年に SBOM 準備度調査を実施した [18]. これは多種多様な業種・規模の 412 の組織を対象としたものであり，76% の組織が SBOM 導入に取り組んでいるものの，40% の組織が業界の SBOM 導入への積極性，39% が SBOM に含めるべき情報に関する業界合意の有無，37% が SBOM によって顧客にもたらされる価値の不明瞭さについて不安を抱いていることを明らかにした．

Xia らは SBOM の普及状況や課題についてのインタビュー調査を実施した [46]. これは SBOM を利用する開発者を対象としたものであり，SBOM 生成のインセンティブ，SBOM に含めるべき情報の合意，SBOM の情報を部分的に開示する仕組み，SBOM の内容検証，成熟した SBOM ツール，SBOM の認知度向上などの重要性を指摘した．

Stalnaker らはステークホルダーが SBOM の作成時や使用時に直面した課題と，課題への対処法を調査した [40]. これは SBOM コミュニティと SBOM 採用者，主要 OSS の貢献者，サイバーフィジカルシステム（CPS）の開発者及び研究者，人工知能・機械学習の開発者及び研究者，法務者を対象としたものであり，複雑な SBOM 仕様の目的別整理，ツールとビルドシステムの SBOM サポート，SBOM の内容検証，SBOM 導入のインセンティブが重要だと指摘した．

Nocera らは個人利用・業務利用共に最も主要なソフトウェア開発プラットフォームである GitHub 上の OSS に対して SBOM の利用状況の調査を行った [31]. その結果として，OSS があまり SBOM を導入していないこと，その中でもコード変更の都度 SBOM を更新し同梱するなどの NTIA 要件を満たしているものは少ないこと，CI/CD やビルドに統合できる SBOM 生成ツールの拡充が重要であることを指摘した. 加えて，イタリアのソフトウェアサプライチェーン管理の現状についてのインタビュー調査も実施した [30]. この調査は大規模な多国籍企業を含む 6 つのソフトウェア企業から異なる役職の実践者を募って行われたもので，ソフトウェア業界のソフトウェアサプライチェーン関連の課題に対する関心は高いものの，SBOM の普及は不十分であることを明らかにした. その上で，SBOM やソフトウェアサプライチェーン関連の法規制について，ソフトウェア産業が十分に理解していないことが一因ではないかと指摘した．

Kloeg らは直接 SBOM に関わるステークホルダーを B2B 顧客，システムインテグレータ，ソフトウェアベンダ，開発者の 4 つに分類してインタビューを実施し，それぞれが重視するサプライチェーンリスク，SBOM の利点，SBOM 導入の課題，SBOM 導入の動機を調査した [23]. その結果，SBOM の生成と利用を担う開発者と B2B 顧客に最も SBOM 導入の動機がなく，SBOM 普及を最も推進できるのはシステムインテグレータとソフトウェアベンダであることを明らかにし，システムインテグレータとソフトウェアベンダによる開発者と B2B 顧客への働きかけが重要だと指摘した．

Bi らは GitHub 上の 510 件の SBOM 関連プロジェクトから 4786 件の議論を調査し，SBOM を効果的に活用するにあたっての主要なトピック，課題，解決策を示し，また SBOM 生成に用いられる

ツールやフレームワークについても調査し，それぞれの強みや限界を明らかにした [3]. その結果から，SBOM の管理方法を課題として挙げるとともに，SBOM の 4 つのライフサイクルのうち管理フェーズの重要性や，既存の SBOM ソリューションにあるギャップを指摘した.

2.5 既存の SBOM 生成ツール

SBOM は小規模なソフトウェアであれば手作業で作成できるが，プロジェクト規模によっては数百，数千を超えるソフトウェア部品を使用していることも少なくない. こうしたプロジェクトでは手動作成が非現実的なため，SBOM 普及が推進される中で，これを自動生成するツールが開発されている.

Microsoft 社が開発している SBOM Tool [28] や Anchore 社が開発している Syft [1] は，共にパッケージマネージャがプロジェクト内に作成するファイルを解析して依存関係やメタデータを抽出し，対象プロジェクトの SBOM を自動生成するツールである. これらは Python，Java，JavaScript など多様な開発言語に対応するが，仕組み上パッケージマネージャの存在が必須であり，パッケージマネージャを使わずに開発されたソフトウェアに対しては SBOM を生成できない.

GitHub が Dependency graph 内で各リポジトリに提供している SBOM エクスポート機能 [16] は，リポジトリ中に存在するパッケージマネージャのファイルを GitHub が自動解析して生成した SBOM を，特定のボタンをクリックするだけでダウンロードできる機能である. これも同様に，パッケージマネージャを使わずに開発されたソフトウェアに対しては SBOM を生成できない.

debiantospdx [47] は，Debian Linux にインストールされた各パッケージについて，依存関係グラフの情報を含む SPDX 形式の SBOM を自動生成するツールである. このツールは，Debian Linux のパッケージリポジトリに存在するパッケージのみを対象とするため，リポジトリ上に存在しない開発中のソフトウェアに対しては SBOM を生成できない.

3　SBOM 利活用に関する課題抽出

一般のソフトウェア開発者が直面している課題を分析し，SBOM の普及に際して解決すべき技術課題を明らかにするため，主要な開発者向け Q&A サイトである Stack Overflow において，SBOM 利活用に関する質問を調査した．

3.1　既存調査の不足点

2.4 節で，SBOM の現状に関する調査を通じて原因や改善案を示した既存研究を紹介した．これらの調査は，業界における SBOM の現状と，今後取り組むべき課題に関して洞察を与えるものであるが，企業や組織に所属せず，SBOM 関連プロジェクトにも携わっていない一般のソフトウェア開発者が直面している課題について，明らかにしたものは現時点では存在しない．そのため，SBOM 普及をより強く推進するためには，一般のソフトウェア開発者が直面している課題を分析して，解決すべき技術課題を明らかにすることが必要だと考えた．

3.2　Stack Overflow

図 1　Stack Overflow における質問の例

Stack Overflow は，質問とそれに対する回答を投稿・検索できる開発者に特化した Q&A サイトであり，あるユーザの質問に不特定多数のユーザが回答する．Stack Overflow における質問の例を図 1 に示す[*2]．Stack Overflow における投稿は，タイトル，質問，複数個のタグ，そして質問に対する複数の回答で構成される．複数の回答のうち，質問の解決に最も貢献したと質問者が判断した回答 1 つについては，緑色のチェックマークで表される Accepted が付与される．ただし，どの回答でも質問が解決されなかった場合など，どの回答にも Accepted が付与されないこともある．

Stack Overflow は 2024 年 2 月現在，ユーザ数は約 2,200 万人，投稿は質問と回答を合わせて約 6,000 万件，そして 1 日に約 2,700 件の質問が投稿されており，広く普及していることがわかる [37]．新たなツールや技術の調査にも広く用いられ，新たなツールの評価にあたり Stack Overflow などの開発者コミュニティを利用すると答えた開発者は 61.3% と，無料トライアルを開始する，知り合いの開発者に質問するに続く第 3 位となっている [39]．

以上の事実から，SBOM を利活用する際に課題に直面した開発者は，その解決法について Stack Overflow で質問している可能性が高いと考えられる．

3.3 調査

SBOM フォーマットとして CycloneDX と SPDX を対象とし，Stack Overflow における SBOM 利活用に関する質問について，以下の内容で調査を行った．

調査 1：SBOM 利活用に関する質問の回答・解決状況

Stack Overflow に投稿された質問には有志の開発者が回答するが，必ずしも全ての質問に回答が付くとは限らず，また回答が付いたとしても解決に至るとは限らない．SBOM 利活用に関する質問について回答状況と解決状況を調べることで，SBOM 利用者が課題に直面した際に，Stack Overflow で質問すれば解決できる状況になっているかどうかを調査する．

調査 2：SBOM 利活用に関する質問数の推移

SBOM 利用を求める米国大統領令の発令や SPDX の ISO/IEC 国際標準化など，SBOM を取り巻く状況は変化している．その SBOM 普及への影響を調査するため，Stack Overflow における SBOM 利活用に関する質問数の推移を調べる．

調査 3：SBOM 利用者が抱いている課題

Stack Overflow 上の SBOM 利活用に関する質問内容を調べることで，実際に SBOM を利用する一般のソフトウェア開発者が抱いている課題を調査する．

調査にあたり，図 2 の手順で調査対象の質問を抽出する．まず，検索キーワードとして「SBOM」，「CycloneDX」，「SPDX」を選出した．大文字小文字は区別せず，タイトルまたは本文にこれらのキーワードのうち少なくとも 1 つが含まれる質問を抽出する．Stack Overflow には質問内容に応じてタグを設定する機能があるが，調査時点で SBOM 関連のタグはほとんど設定されていないため使用しな

[*2] https://stackoverflow.com/questions/69121175

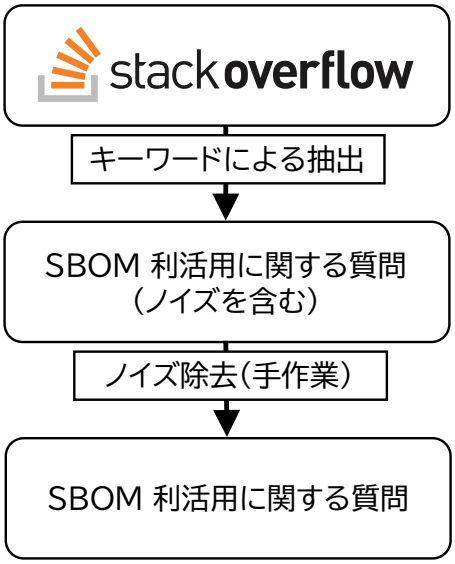

図 2　調査の流れ

い. その後, 利用しているフレームワークが SBOM にまつわる文字列を出力した, ソースコード中に SBOM に関する文字列が含まれていた, などといった SBOM 利活用に関する質問ではないもの（以下ノイズと呼ぶ）を取り除く.

キーワード「SBOM」と「CycloneDX」については Stack Overflow のサイト上の検索ボックスから, それぞれ `SBOM is:question`, `CycloneDX is:question` と入力して検索し, 得られたものからノイズを手作業で取り除く. 大文字小文字は区別されず, `is:question` によって質問のみが検索結果に表示される. ただし, 次のキーワード「SPDX」で用いるデータベースが 2023 年 9 月版であるため, 一貫性を保つために 2023 年 8 月中までの質問のみを対象とする.

キーワード「SPDX」については Stack Exchange Data Dump 2023-09-12[*3]を利用し, 後述するノイズパターン文字列を機械的に取り除いたのちに残ったノイズを手作業で取り除く. ここで, キーワード「SBOM」と「CycloneDX」については Web サイトから直接検索したのに対し, キーワード「SPDX」ではデータダンプを用いた理由は, 検索時にマッチした質問件数が前者はそれぞれ 57 件, 31 件だったのに対し, 後者は Stack Overflow の検索機能の上限である 500 件を超えたためである. Web サイト上の検索ボックスにもキーワード除外機能が存在するが, Stack Overflow 上では投稿が単語ごとにインデックスされているため, 複数の単語で構成されるノイズパターン文字列を除外する検索が正しく動作しなかった. キーワード「SPDX」における手動でのノイズ除去は, データダンプから機械的にノイズパターン文字列を取り除いた後に得られる投稿について, 質問ページを閲覧して行う.

表 2 に除外するノイズパターン文字列とその集計結果（重複を含む）を示す. ノイズパターン文字列は, ノイズを手作業で除外する過程で SBOM の利活用に無関係な投稿に頻出する文字列を抽出し,

[*3] `https://archive.org/details/stackexchange_20230912`

表 2　キーワード「SPDX」におけるノイズパターン文字列

ノイズパターン文字列	件数
`SPDX-License-Identifier`	1229
`License should be a valid SPDX license expression`	30
`spdy`	51
`spdx-correct`, `spdx-expression-parse`, `spdx-exceptions`, `spdx-license-ids`	62

それが SPDX 形式の SBOM に現れる文字列でないことを確認した上で選定した. 以下, 各ノイズパターン文字列の選定理由を説明する.

SPDX-License-Identifier

プログラムコード冒頭でライセンスを宣言する際に使用する統一的な記法であり, この宣言を含むプログラムコードを掲載する質問がマッチしていた. これは SPDX 規格の一部として Linux Foundation により定められている [26] が, これ自身は SBOM ではなく, また SPDX 形式の SBOM にも現れないため除外した.

License should be a valid SPDX license expression

JavaScript 向けのパッケージマネージャ Yarn が, パッケージに設定されたライセンス表記が誤っている場合に出力する警告メッセージであり, SBOM とは無関係なため除外した.

spdy

プログラム中で x 軸の速度を表す変数名として `spdx`（大文字小文字の組み合わせが異なるものも存在）を用いているケースがあり, これらのケースでは概ね y 軸の速度を表す `spdy` を伴うことから除外した.

spdx-correct, spdx-expression-parse, spdx-exceptions, spdx-license-ids

いずれもパッケージマネージャ npm 上のパッケージ名であるが, npm のエラーに関する質問で, 対象のプロジェクトの依存関係として含まれるためにログ中に現れ, マッチするケースがほとんどだったため取り除いた.

この処理の結果, キーワード「SBOM」では 27 件,「CycloneDX」では 28 件,「SPDX」では 6 件の質問が抽出された. また, 全体としては重複があるため 42 件となった. この準備の下で, 各調査を実施する.

調査 1

得られた質問について, 回答の有無や解決状況を調べる. Accepted 回答が存在する質問を

解決，存在しない質問を未解決とみなす．

調査 2

得られた質問について，投稿日時を元に各年ごとに分類する．投稿日時は最終更新ではなく，Stack Overflow のサイト上で「asked Mar 10, 2021 at 10:35」のように記載される初出の日時を使用する．

調査 3

各質問の内容から技術課題を読み取り，頻繁に質問されている内容や未解決と思われる技術課題を，筆頭著者の主観により抽出する．

3.4 調査結果と考察

3.4.1 調査 1：SBOM 利活用に関する質問の回答・解決状況

「回答あり（解決）」，「回答あり（未解決）」，「回答なし」に分類した結果と，2024 年 2 月 1 日現在の Stack Overflow 全体の状況 [38] を図 3 に示す．SBOM に関する質問が少ない 2020 年以前の質問は，調査の公平性のため除外している．SBOM 利活用に関する質問全 40 件中，回答あり（解決）は 6 件（15.0%），回答あり（未解決）は 19 件（47.5%），回答なしは 15 件（37.5%）であった．

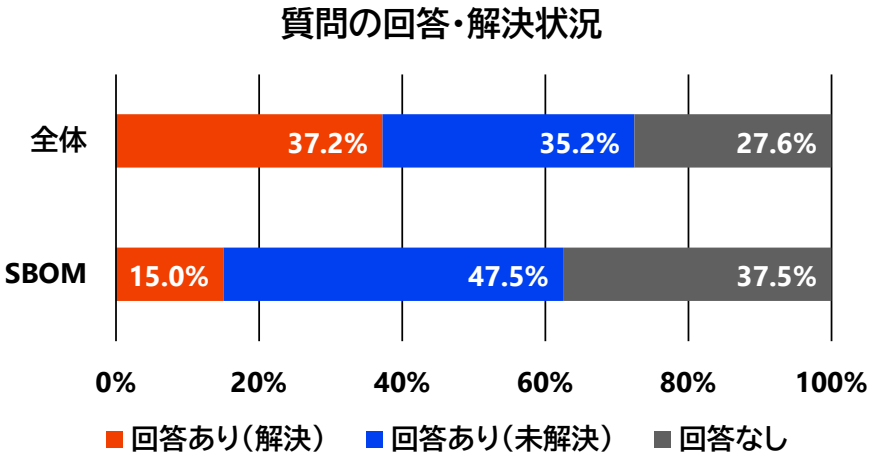

図 3　質問の回答・解決状況

Stack Overflow 全体における回答なしの質問の割合が 27.6% であることを考えると，SBOM 利活用に関する質問の回答状況は概ね平均的と言える．一方，Stack Oveflow 全体における解決済み質問の割合が 37.2% であることを考えると，SBOM 利活用に関する質問の解決状況は極めて低い水準であり，SBOM 利用者が課題に直面した際に Stack Overflow で質問することでは解決が難しい状況だと言える．解決済みの質問が少ない要因として，3.4.3.3 節も考慮すると，SBOM 利活用に関する知見が不十分で質問に回答できるソフトウェア開発者が不足していることや，SBOM 利活用

の技術が未熟で質問時点では技術的に解決不可能な事項が多いことが考えられる.

3.4.2　調査 2: SBOM 利活用に関する質問数の推移

年間の新規質問数について，2019 年から 5 年間の推移を図 4 に示す. ただし，2023 年は 9 月時点の件数である．2018 年以前は，3.3 節の手順で抽出された質問件数が 2012 年のみ 1 件, それ以外は 0 件であった.

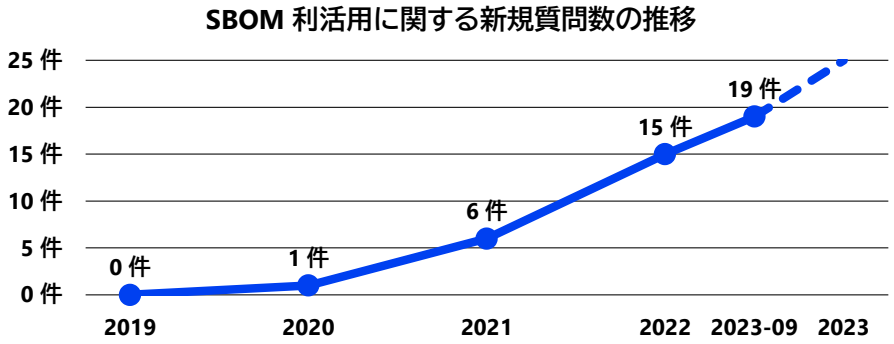

図 4　SBOM 利活用に関する新規質問数の推移

Stack Overflow には 2024 年 2 月現在, 全体で約 2400 万件の質問が存在し，1 日に約 2,700 件の質問が投稿されていることを考えると, 本調査で抽出された SBOM 利活用に関する質問は 4 年間を通して少数である. しかし，2020 年から 2023 年にかけて着実に新規質問数が増加しており, 特に SBOM 利用を求める米国大統領令の発令や SPDX の ISO/IEC 国際標準化が行われた 2021 年以降に, 因果関係は断定できないがその傾向が加速していることがわかる.

3.4.3　調査 3: SBOM 利用者が抱いている課題

大きく以下の課題が挙げられることがわかった.

1. SBOM ツールのユースケースの網羅性に不足がある（7 件）
2. 各種要件を満たす SBOM を生成できない（4 件）
3. SBOM ツールに不具合がある, または利用方法が不明瞭（19 件）

以降, それぞれについて代表的なものを抜粋して紹介する.

3.4.3.1　SBOM ツールのユースケースの網羅性に不足がある

How do I generate a Cyclonedx bom for a Java project built with Ant?[*4] **(2022)**

[*4] https://stackoverflow.com/questions/71605182

I'm already generating boms and using them with Dependency Track for some projects built with Gradle. There's a CycloneDx Gradle plugin that works well for that. However I'm also working with many older Java projects that are built with Ant. I've not been able to find an Ant tool to generate the boms anywhere. Is there one out there? If not, what's the best way to generate the bom files?…

これは，Gradle を用いて開発されている Java プロジェクトに対しては SBOM 生成経験があるソフトウェア開発者が，Apache Ant を用いて開発されている Java プロジェクトに対する SBOM 生成ツールを見つけられず，生成方法を求めている質問である．CycloneDX Core (Java)[*5]を使えば生成できる可能性があると指摘している回答があるが Accepted ではない．Gradle に対して Apache Ant は旧式のプロジェクト管理システムであるために，SBOM 生成ツールが開発されていないと考えられる．

Is there any tool through which we can generate SBOM report (SPDX / CycloneDX) for Windows programs?[*6] **(2022)**

…There are many tools available which can scan Linux OS packages and application packages (e.g java, maven, .net) like Trivy, Syft, whitesource but it looks like there is no tool available which can generate SBOM report for the applications installed on Microsoft Windows.…

これは，Linux においては Syft などのツールを用いればインストールされたソフトウェアの SBOM を生成できるが，Windows ではそれを行えるツールが見つからないという質問である．こちらも回答があるが Accepted ではない．Linux では APT や DNF といったパッケージマネージャを使ってソフトウェアをインストールすることが多い一方，Windows ではパッケージマネージャを使わずに直接インストールすることが多い．そのため，Windows では集権的にソフトウェアのメタデータが管理されておらず，個別のソフトウェアがシステムに登録する情報しか抽出できない背景から，Windows に対応する SBOM 生成ツールの開発が追い付いていないと考えられる．

Generate Software Bill Of Material SBOM for AOSP[*7] **(2022)**

I am working with AOSP 9 and 11 to build a android based embedded system. For several review task I require a list of all software packages, libraries, etc. that are used to build and/or get installed in the final image plus their versions. Is there a good and reliable way to generate this SBOM for AOSP?

これは，Google 社が開発するスマートデバイス向け OS である Android 9, 11 ベースの組み込

[*5] https://github.com/CycloneDX/cyclonedx-core-java
[*6] https://stackoverflow.com/questions/73648096
[*7] https://stackoverflow.com/questions/73879514

みシステムを開発するソフトウェア開発者が，Android のソースコードである AOSP (Android Open Source Project) のビルド時に使用されるソフトウェアパッケージや，最終成果物に含まれるソフトウェアパッケージに関する SBOM を生成する方法を求めている質問である．回答には Accepted ではないが，Android 14 で SBOM 生成機能が追加されたというものがある．Android は 1 年に 1 度新たなバージョンがリリースされるが，実際に Android を搭載する機器には古いバージョンが搭載されることも多い．この質問は，古いシステムを対象に SBOM を生成するための技術の不足を表していると考えられる．

3.4.3.2 各種要件を満たす SBOM を生成できない

> **NTIA minimum SBOM requirement tool[*8] (2023)**
> I am trying to generate SBOM for Java, Python, ios (Swift) and Android (kotlin) project. I need to follow the NTIA guidelines for minimum element for SBOM (`https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf`). These elements are Supplier Name, Component Name, Version of the component, Other identifier, dependency relationship, Author of SBOM data and timestamp. I tried command line SNYK and SYFT to generate SBOM for my projects Java and Python projects in SPDX format. Neither of the tools generates Supplier name. Syft did not generate the SBOM for my ios project. Still to perform SNYK test for ios project. Have anyone used some other tools to generate SBOM with minimum NTIA required fields. Is there any tool to generate report in word or pdf format based on the generated json?

これは，開発者が Java と Python について SBOM 生成を試みたものの，NTIA 要件を満たす SBOM を生成するツールを見つけられず，そのようなツールを求めている質問である．この要件は 2021 年 7 月に発行されたが，追従できていないツールが多く存在することや，ツールの実装において要件に定められる情報の入手が困難などの課題があることを表す質問だと考えられる．

> **How to include Open Source license in GitHub SBOM export?[*9] (2023)**
> Is there a way to include the Open Source license in the GitHub SBOM export? If not is there a way to easily generate a list of dependencies that includes the name, version, and open source license for a GitHub repo? I tried the Export SBOM under dependencies but this does not seem to include the Open Source license for the dependencies.

これは，依存パッケージのライセンス情報が GitHub の SBOM 出力機能に含まれないため，これを含める方法を求めている質問である．コンプライアンス管理などの目的でライセンス情報が必要な状

[*8] `https://stackoverflow.com/questions/76103711`
[*9] `https://stackoverflow.com/questions/76962834`

況において，GitHub の SBOM 出力機能の不足を表す質問だと考えられる．

3.4.3.3 SBOM ツールに不具合がある，または利用方法が不明瞭

How to create a BOM file in a Flutter project[*10] **(2023)**

I'm trying to create a BOM file for the Android portion of a Flutter project for security scanning. I added org.cyclonedx.bom (a gradle plugin) to gradle and I'm running the cyclonedxBom gradle task, but I'm getting an error:…

これは，Flutter を用いるソフトウェア開発者が CycloneDX 形式の SBOM 生成を試みたものの，エラーが出て生成できなかったという質問である．この他にも，エラーの解決法や，SBOM ツールの使い方に関する質問が多く存在した．

CycloneDx is not able to find project.assets.json file in correct path[*11]**(2022)**

CI/CD サービスである Azure Pipeline でアプリケーション開発フレームワーク.NET を使ったプロジェクトの SBOM を作ろうとしているが，CycloneDX 生成ツールが `project.assets.json` ファイルを探す場所が間違っており，生成されないという質問．これは別のパスを指定する機能の認知不足を表している．

How to extract nix derivations from package files?[*12]**(2022)**

パッケージマネージャ Nix から SPDX を作るツール Nixbom を見つけたが，このツールが必要とするファイルを得る方法が分からないという質問．ツールの使い方の認知不足を表している．

I installed Micorosoft Sbom-tool, but it cannot detect any packages[*13]**(2023)**

Microsoft 社製の SBOM 生成ツールが何のパッケージも検出せず，どうすれば良いかという質問．質問内容だけからは判断できないが，ツールの不具合であるか，あるいはツールの使い方の認知不足を表している．

cycloneDx Bom is not fetching all dependencies of my project[*14]**(2023)**

CycloneDX 生成ツールが.NET を使ったプロジェクトの完全な依存関係を抽出せず，どうすれば良いかという質問．質問内容だけからは判断できないが，ツールの不具合であるか，あるいはツールの使い方の認知不足を表している．

cyclonedxBom gradle plugin is not analyzing my dependencies[*15]**(2023)**

CycloneDX 生成ツール `cyclonedx-gradle-plugin` が Android プロジェクトの依存関係

[*10] `https://stackoverflow.com/questions/76515339`
[*11] `https://stackoverflow.com/questions/71073665`
[*12] `https://stackoverflow.com/questions/73022368`
[*13] `https://stackoverflow.com/questions/75576917`
[*14] `https://stackoverflow.com/questions/76051534`
[*15] `https://stackoverflow.com/questions/76333776`

を全く抽出できず，どうすれば良いかという質問．質問内容だけからは判断できないが，ツールの不具合であるか，あるいはツールの使い方の認知不足を表している．

BOM does not confirm to the CycloneDX BOM standard[*16]**(2023)**

Java で開発されたレガシーシステムの SBOM をビルドツール Gradle の CycloneDX 用プラグインで作成しているが，1 つのプロジェクトだけ生成中に標記のエラーが出力される問題があり，どうすれば良いかという質問である．これは，ツールの不具合の可能性が高いと考えられる．

これらの質問は，SBOM ツールが未熟なために不具合や機能不足によって正しく SBOM 生成できないケースが存在することや，SBOM 生成に関するドキュメントや知見が十分に蓄積されておらず，ソフトウェア開発者が SBOM ツールの基本的な利用方法を理解できていないことを示唆していると考えられる．

3.5 妥当性への脅威

3.5.1 ノイズの除外方法

本調査では，キーワード「SPDX」を含む質問の調査において表 2 のキーワードを除外した．しかし，これにより本来除外されるべきではない，ノイズに該当しない SBOM 利活用に関する質問が除外された可能性がある．ただし，除外キーワードの選定にあたっては，実際にそのキーワードを含む投稿を複数抽出して確認し，該当の除外キーワードを含む投稿がノイズである可能性が高いことを確認しているため，これによる妥当性への脅威は小さいと考えられる．

3.5.2 調査数

本調査において，条件を満たすとして抽出された質問数は全体で 42 件と少数だった．SBOM が未だ十分に普及していないことが原因と考えられるが，調査結果の正確性に悪影響を与えている可能性がある．一方，Stack Overflow 以外にも Quora[*17]などの一般の Q&A サイトが存在するが，調べた範囲では SBOM に携わる開発者や管理者が，それらを利用して問題解決を活発に行っている様子はみられなかった．

3.6 結論

開発者向け Q&A サイト Stack Overflow に投稿された，SBOM 利活用に関する質問の回答・解決状況（調査 1），SBOM 利活用に関する質問数の推移（調査 2），SBOM 利用者が抱いている課題（調査 3）の調査を行った．

[*16] https://stackoverflow.com/questions/76501670

[*17] https://www.quora.com/

調査 1 では，解決済みの質問は 15.0% と Stack Overflow 全体における 37.2% に対して極めて低い水準であり，SBOM 利用者が課題に直面した際に Stack Overflow で質問することでは解決が難しい状況であることがわかった．調査 2 では，SBOM 利用を求める米国大統領令や SPDX の ISO/IEC 国際標準化などがなされた 2021 年以降，新規質問数の増加が加速していることがわかった．調査 3 では，SBOM 利用者が抱いている課題が「SBOM ツールのユースケースの網羅性に不足がある」，「各種要件を満たす SBOM を生成できない」，「SBOM ツールに不具合がある，または利用方法が不明瞭である」の 3 つであることがわかった．

今回の結果から，SBOM に対する潜在的な疑問・質問は多く存在すると思われるが，Stack Overflow への投稿数は少ない．Stack Overflow が投稿先として適切でないと思われているためか，SBOM が十分に普及していないためかなど，今後の理由の分析が必要である．そのために，質問数が増加傾向にある Stack Overflow の調査を継続するとともに，Reddit[*18]のような Stack Overflow 以外の Web サイトでも調査を行い，SBOM 利活用の最新のトレンドを追跡することが今後の課題である．

[*18] https://www.reddit.com/

4　C/C++ のソフトウェアに対する SBOM 生成の課題

3 節の調査から，SBOM ツールのユースケースの網羅性不足や，各種要件を満たす SBOM を生成できないことが，SBOM 普及の妨げとなっていた．そこで，SBOM 普及に資するため，既存ツールではサポートされないユースケースである C 言語及び C++ 言語（以下，C/C++ と省略する）で開発されたソフトウェアに対し，NTIA 要件 [42] を満たす SBOM を自動生成する手法を開発した．本節では，C/C++ の特性と既存研究について述べ，SBOM 生成にかかる課題を示す．

4.1　C/C++

C 言語はコンパイル型のプログラミング言語で，1970 年代に登場した．C++ 言語は C 言語を拡張する形で開発されたプログラミング言語で，1980 年代に登場した．共に歴史が長く，また実行速度が速く軽量であり，低水準な処理を記述できるなどの特性を備えていることから，多数の後継のプログラミング言語が登場した現在でも広く利用されている．実際に GitHub が 2024 年に行った調査では，GitHub 上の C 言語で開発されているプロジェクトに貢献するユーザ数が全言語中 9 位，C++ 言語では 6 位だった [17] ほか，Stack Exchange が 2024 年に行った調査でも，C 言語が 10 番目，C++ 言語が 9 番目に多くのユーザに利用されていると報告されており [39]，両言語とも両調査で上位 10 位に入る主要なプログラミング言語である．また，統計などの分野で幅広く使用されている Python の数値計算ライブラリ NumPy[*19]など，後継言語の中にも C/C++ の部品に依存するものが多く存在する．

しかし，Python であれば pip，Java であれば Maven，JavaScript (Node.js) であれば npm というように，後継の言語には依存関係とそのメタデータを管理するパッケージマネージャが存在するが，C/C++ はその歴史の長さからパッケージマネージャが登場する以前より使用されており，プロジェクトごとに独自の方法でビルド手順や依存関係を管理している．ビルドに使用するツールやスクリプトの内容や記法が千差万別なことに加え，依存関係やメタデータも多くの場合，プロジェクトの Readme ファイルなどの文書に記載されているだけで機械可読でないため，依存関係とメタデータを容易に抽出する方法が存在しない．現在では Conan[*20]と呼ばれる C/C++ 用のパッケージマネージャが開発されているが，これを使わずに構築されたプロジェクトを Conan を使うように修正するには膨大な作業を要するため，現実的ではない．こうした事情から，2.5 節で見たように Python, Java, JavaScript といったプログラミング言語で開発されたソフトウェアに対しては，パッケージマネージャが取り扱うファイルを情報源として SBOM を自動で生成するツールが既に登場している一方で，パッケージマネージャを用いない C/C++ のソフトウェアに対して容易に SBOM を生成する方法は未だ存在しない．

[*19] https://numpy.org/
[*20] https://conan.io/

4.2 C/C++ の依存関係抽出に関する既存研究

Na らは CNEPS と呼ばれる，C/C++ のオープンソース部品への依存関係を抽出する手法を開発した [29]. これはソースコード解析を用いて`#include` などのライブラリを取り込むコードを分析し，モジュールと呼ばれる新たな粒度を導入して依存関係グラフを構築し，C/C++ のコードベースに含まれる依存関係を抽出するものである. しかし，この手法はコードベースを部品単位に切り出すものであり，SBOM を構築するために必要なメタデータを得られないほか，本体のコードベースの一部をライブラリとして誤認識する場合もあり，現時点では SBOM の自動生成への応用は難しい.

5　C/C++ のソフトウェアに対する SBOM 生成手法の開発

パッケージマネージャが存在しない C/C++ のソフトウェアに対して SBOM を生成するにあたり，ビルド処理から得られる情報と，その結果生成されるバイナリに含まれるメタデータに着目した．Ubuntu を含む Debian 系の Linux ディストリビューションにおいて，GCC でビルドする C/C++ のソフトウェアを対象とする．対象 OS としてこれらを選定した理由は，現在 Web 上に公開されているサーバのうち 55.4% が Linux を採用しており，そのうち Ubuntu と Debian が上位 2 位を占めているなど，最もよく使われている OS だからである [35].

本節では，開発した手法について，C/C++ で開発されたソフトウェアから依存関係を抽出し，そのメタデータを抽出して SBOM を生成するまでの一連の流れを説明する．

5.1　依存ファイル群の抽出

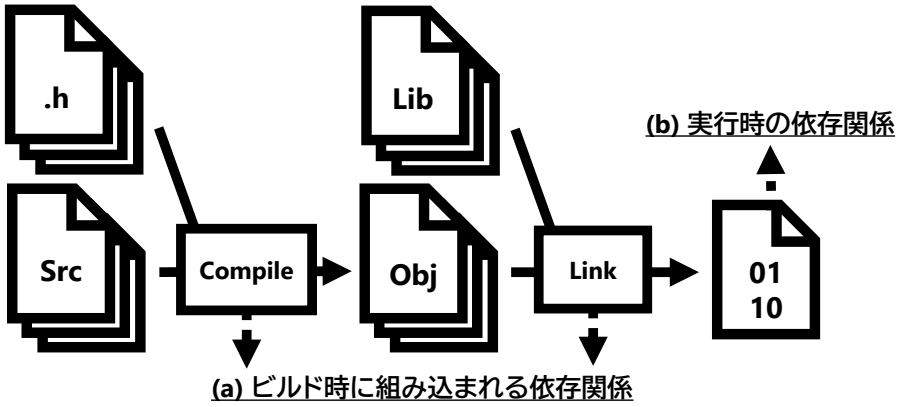

図 5　C/C++ のビルド処理の流れ

C/C++ のビルド処理は，図 5 の手順で行われる．まず，C/C++ で書かれたソースコード（.c ファイルや .cpp ファイルなど）及びヘッダファイル（.h ファイルや .hpp ファイルなど）を読み込んでコンパイル処理を行い，機械語のオブジェクトファイル（.o ファイル）に変換する．そして，得られたオブジェクトファイルが必要とする外部の関数シンボルと，それを提供する事前に用意されているバイナリライブラリを照合するリンク処理を行い，静的ライブラリ（.o ファイルや .a ファイルなど）はビルド成果物バイナリに埋め込み，動的ライブラリ（.so ファイルなど，共有ライブラリとも呼ぶ）については，実行時に必要なバイナリライブラリとシンボルの情報をビルド成果物バイナリ内にメタデータとして記録する．こうして，実行可能な成果物バイナリファイルを生成する．

コンパイル処理で読み込むヘッダファイルには，コードベース内部のもの以外にシステムにインストールされたヘッダライブラリも含まれる．加えて，リンク処理で読み込むバイナリライブラリにも，同様にシステムにインストールされたものが含まれる．また，ビルド成果物バイナリ内にメタデータとして記録されたバイナリライブラリについても，システムにインストールされたものが含まれている．

以上のことから，以下の通り依存関係を抽出できる．

(a) **ビルド処理中のコンパイラ・リンカの動作を観測すれば，ビルド時に組み込まれる依存関係を抽出できる**

(b) **ビルド成果物バイナリのメタデータを解析すれば，実行時の依存関係を抽出できる**

表 3 に示す米国の政府機関 CISA (Cybersecurity and Infrastructure Security Agency) が公開した文書 [8] で示された SBOM の分類によると，(a) の情報に基づいて生成される SBOM は Build SBOM に該当し，(b) の情報に基づいて生成される SBOM は Analyzed SBOM に該当する．この 2 つの SBOM は相補的に機能する．(a) ではヘッダファイル，静的リンク，動的リンクが全て依存関係として抽出される．しかし，動的リンクについてはビルド環境に存在するライブラリバージョンが抽出されるが，ビルドしたソフトウェアを実行する際には実行環境に存在するライブラリが使われ，この両者は必ずしも一致しない．また，動的リンクしたライブラリの推移的依存関係については，このステップでは抽出されない．一方，(b) では動的リンクだけが依存関係として抽出され，既にコンパイルされて成果物バイナリの一部をなしているヘッダファイルと静的リンクは抽出されない．しかし，実際に使用される動的ライブラリのバージョンと，動的ライブラリの推移的依存関係が抽出される．

表 3　SBOM の分類

種別	定義
Design	設計段階で作成する SBOM
Source	プロジェクトから作成する SBOM
Build	**ビルド時に作成する SBOM**
Analyzed	**成果物から作成する SBOM**
Deployed	稼働環境全体の SBOM
Runtime	システムの動作を観測して作成する SBOM

5.1.1　ビルド時に組み込まれる依存関係の抽出

ビルド時に組み込まれる依存関係は，ビルド時に実行されるコマンドの詳細なログ出力から得られる．まず，ビルド時にコンパイラである gcc あるいは g++ に対して -H オプションを渡すことで，Listing 1 の通りコンパイル処理で用いるヘッダファイルが全て出力される（抜粋）．この例では，OpenSSL ライブラリ[*21]が読み込まれている．

Listing 1　ビルド時に使用されるヘッダファイル

```
1 . /usr/include/openssl/ssl.h
2 .. /usr/include/openssl/comp.h
```

[*21] https://openssl-library.org/

```
3 ... /usr/include/openssl/crypto.h
4 .... /usr/include/openssl/safestack.h
```

また，リンカである ld に対して -t オプションを渡すことで，Listing 2 の通りリンク処理で用いるバイナリライブラリが全て出力される（抜粋）．コンパイラフロントエンドとして gcc あるいは g++ で一括でリンク処理まで実行しているソフトウェアでは，-Wl,-t オプションを渡すことで内部的に ld にオプションが渡される．この例では，libpsl ライブラリ[*22]と OpenSSL ライブラリが読み込まれている．

Listing 2　ビルド時に使用されるバイナリライブラリ

```
1 /usr/lib/gcc/x86_64-linux-gnu/13/../../../x86_64-linux-gnu/libpsl.so
2 /usr/lib/gcc/x86_64-linux-gnu/13/../../../x86_64-linux-gnu/libssl.so
3 /usr/lib/gcc/x86_64-linux-gnu/13/../../../x86_64-linux-gnu/libcrypto.so
```

ただし，出力には外部ライブラリでない，コードベースの一部をなすファイルや，コードベースをコンパイルした結果出力されたオブジェクトファイルも含まれるので，これらは除外する．

5.1.2　実行時の依存関係の抽出

実行時に呼び出される依存関係は，実行可能バイナリファイルにおいて実行に必要な周辺情報を記載した「ヘッダ領域」に記録されている．主要な Linux ディストリビューションに最初からインストールされている ldd スクリプトを用いると，ヘッダ領域を解析して実行時に読み込むバイナリライブラリが Listing 3 の通り全て出力される（抜粋）．この例では，libpsl ライブラリと OpenSSL ライブラリを必要としている．

Listing 3　実行時に使用されるバイナリライブラリ

```
1 libpsl.so.5 => /lib/x86_64-linux-gnu/libpsl.so.5 (0x00007753cb293000)
2 libssl.so.3 => /lib/x86_64-linux-gnu/libssl.so.3 (0x00007753caf56000)
3 libcrypto.so.3 => /lib/x86_64-linux-gnu/libcrypto.so.3 (0x00007753caa00000)
```

ただし，出力には Linux kernel がプログラムの実行時に内部的に使用する vdso と ld.so も含まれる [13]．これらは依存ライブラリではないので，linux-vdso または ld-linux を名称中に含む依存関係は除外する．

5.2　ファイルパスの正規化

5.1 節で依存ファイルのパスが得られたが，2.3 節で説明した NTIA 要件を満たす SBOM を生成するためには，これらのライブラリファイルに対応するメタデータを抽出する必要がある．そのために，まず抽出したファイルパスを実際にシステム上に存在する具体的なファイルにマッピングする．

この時点では，ファイルパスは以下の状態のものが混在している．

- /./ や /../ や // といった冗長なパスが含まれる

[*22] https://github.com/rockdaboot/libpsl

- 実体のファイルではなくシンボリックリンクを指している
- ファイル名が完全ではない（例：実際に存在するファイルは `libpsl.so.5` であるところ，出力には `libpsl.so` と記載されている）

これでは出自パッケージを特定できないため，以下の処理を抽出した全ファイルパス文字列に対して適用した上で，重複するものを取り除く．

1. ファイルシステム上での解決を試み，成功すればそれを最終結果とする
2. 失敗すれば，ファイルシステム上で，辞書順でファイルパス文字列で始まる最も早いファイルを探す
3. 見つかったファイルについてファイルシステム上で解決し，それを最終結果とする

この処理の結果，例えば Listing 2 のファイルパスは Listing 4 のように変換される．

Listing 4　正規化後のファイルパス

```
1 /usr/lib/x86_64-linux-gnu/libpsl.so.5.3.4
2 /usr/lib/x86_64-linux-gnu/libssl.so.3
3 /usr/lib/x86_64-linux-gnu/libcrypto.so.3
```

5.3　パッケージへの対応付け

5.2 節で正規化したファイルパスを用いて，それを提供する OS 上のパッケージと対応付ける．

このステップでは，Debian 系 OS でパッケージを管理する `dpkg` ツールが持っているデータベースに問い合わせる．具体的には，正規化されたファイル名を `dpkg-query -S` に渡すことでパッケージ名が得られるので，これを全ファイルに対して行う．1 つのパッケージが複数のファイルを提供している場合は重複が生じるため，重複を取り除く処理も行う．CPU アーキテクチャに依存するパッケージについては，どのアーキテクチャ向けのパッケージがインストールされているかの情報も得られるため，この情報も保持する．

加えて，メタデータ抽出において実際にインストールされているバージョンに基づくメタデータを得るために，インストール済みのバージョンを特定する．具体的には，パッケージ名を `dpkg-query -W`[*23] に渡すことでインストールされたバージョンが得られるので，これを全パッケージに対して行う．

この処理の結果，例えば Listing 4 は，`libpsl5t64:amd64=3.0.13-0ubuntu3.4` と `libssl3t64:amd64=3.0.13-0ubuntu3.4` のようにパッケージ名とバージョンに変換される．

5.4　OS バージョンの抽出

NTIA 要件を満たすためにはパッケージに Other Unique Identifiers を与える必要がある．これは CPE，SWID tag，PURL などが認められているが，ここでは他の SBOM 生成ツールで広く使われて

[*23] 類似のコマンドとして dpkg-query -l があるが，これは出力が機械可読形式でない．

いる PURL (Package URL)[*24]を採用する．Debian 系 OS のパッケージについては生成方法が公式に規定されており，以下の情報が必要となる [34].

- debian，ubuntu などのベンダ名（ディストリビューション名）
- パッケージ名
- バージョン
- 対象 CPU アーキテクチャ（CPU アーキテクチャに依存するパッケージのみ）
- bookworm，noble などの OS バージョン（コードネーム）

このうち，パッケージ名，バージョン，対象 CPU アーキテクチャについてはここまでのステップで抽出できているので，ベンダ名と OS バージョンの取得が必要となる．Debian 系 OS では/etc/os-release にこれらの情報が記載されているため，このファイルを解析して ID= で始まる行からベンダ名，VERSION_CODENAME= で始まる行から OS バージョンを，それぞれ小文字化して抽出する．Listing 5 に Ubuntu 24.04 におけるファイル内容の例を示す．

Listing 5　/etc/os-release ファイルの例

```
1  PRETTY_NAME="Ubuntu 24.04.1 LTS"
2  NAME="Ubuntu"
3  VERSION_ID="24.04"
4  VERSION="24.04.1 LTS (Noble Numbat)"
5  VERSION_CODENAME=noble
6  ID=ubuntu
7  ID_LIKE=debian
8  HOME_URL="https://www.ubuntu.com/"
9  SUPPORT_URL="https://help.ubuntu.com/"
10 BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
11 PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
12 UBUNTU_CODENAME=noble
13 LOGO=ubuntu-logo
```

あとは得られた文字列を組み合わせるが，PURL は URL 規格に沿った文字列とする必要があるため，URL で使えない文字は % エンコードする必要がある．以上の処理の結果，得られる PURL の一例を Listing 6 に示す．

Listing 6　PURL の例

```
1  pkg:deb/ubuntu/libgmp10@2%3A6.3.0%2Bdfsg-2ubuntu6?arch=amd64?distro=noble
```

5.5　パッケージメタデータの抽出

以降のステップでは，以上の準備をもとに，実際にパッケージメタデータを抽出する．Debian パッケージの SBOM を生成する先行研究 [47] では，各パッケージについてソースパッケージをダウンロードし，その中身を ScanCode Toolkit[*25]で解析する手法を用いていたが，これは SBOM 生成に

[*24] https://github.com/package-url/purl-spec/
[*25] https://aboutcode.org/scancode/

ネットワーク帯域やディスク領域などの空間的コストに加え，ダウンロード時間や外部ツールの実行時間といった時間的コストを要する．そのため，本手法ではソースパッケージを利用せずシステム上に既に存在する情報だけを参照し，外部ツールに依存せず単独で解析することで，空間的コストや時間的コストを削減する．

ここでは，Debian 系 OS のパッケージマネージャである APT に問い合わせ，著作権情報以外のパッケージメタデータを抽出する．著作権情報は APT が管理していないため次節で抽出する．

まず，`apt-get download --print-uris`[*26]にパッケージ名とバージョンを与えて，パッケージのダウンロード URL とアーカイブファイル名を取得する．これらの内容は NTIA 要件では求められていないが，SPDX 規格で定義される特に有用性の高い項目に絞ったサブセット規格 SPDX Lite [27] に存在する項目であるため，SBOM の有用性を高める目的で抽出する．

次に，`apt-cache show` にパッケージ名とバージョンを与えて実行する．すると Listing 7 に示すような出力が得られるので，これを解析して以下の SPDX 項目に対応する情報を抽出する．以下の項目において，NTIA 要件で求められる項目に NTIA，SPDX Lite に含まれる項目に SPDX Lite と記している．

supplier (NTIA, SPDX Lite)

パッケージ供給者．`Maintainer:` から始まる行を抽出する．出力ではメールアドレスは <> で囲まれるが，SPDX 規格では () で囲むよう規定しているため，その変換を行う．

originator

上記のパッケージ供給者がソースコードを入手した，元のパッケージの供給者．`Original-Maintainer:` から始まる行を抽出する．Ubuntu のように上流のディストリビューションをベースとして構築されたディストリビューションでは，上流のパッケージを調整して配布していることがあり，その場合にこの項目が記録される．

homepage (SPDX Lite)

パッケージ供給元のホームページ．`Homepage:` から始まる行を抽出する．

summary

パッケージの要約説明．`Description-en:` から始まる行を抽出する．

description

パッケージの詳細説明．`Description-en:` の次行以降を抽出する．1 つの空白から始まる行は通常行，1 つの空白とドットで始まる行は空行，2 つ以上の空白で始まる行は整形済み文字列と定められている [10] ため，これに従って解析する．空白から始まらない行に到達すれば，それは次の項目なので終了する．

checksums

パッケージのアーカイブファイルのチェックサム値で，パッケージが改竄されていないことを

[*26] apt コマンドは人間がインタラクティブに使用するコマンドとして設計されており，今後のインタフェースの互換性が保証されないため，ツールによる自動実行では旧来の apt-get コマンドや apt-cache コマンドの利用が推奨されている [12]．

確認するために用いられる．SPDX では複数のアルゴリズムで計算した値を掲載できるため，取得できる全てを抽出する．MD5 アルゴリズムのものは `MD5sum:`，SHA1 は `SHA1:`，SHA256 は `SHA256:`，SHA512 は `SHA512:` から始まる行を抽出する．

Listing 7　apt-cache show の出力例

```
Package: libssl3t64
Architecture: amd64
Version: 3.0.13-0ubuntu3.4
Multi-Arch: same
Priority: required
Section: libs
Source: openssl
Origin: Ubuntu
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Debian OpenSSL Team <pkg-openssl-devel@alioth-lists.debian.net>
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Installed-Size: 6608
Provides: libssl3 (= 3.0.13-0ubuntu3.4)
Depends: libc6 (>= 2.38)
Breaks: libssl3 (<< 3.0.13-0ubuntu3.4)
Replaces: libssl3
Filename: pool/main/o/openssl/libssl3t64_3.0.13-0ubuntu3.4_amd64.deb
Size: 1940128
MD5sum: 3117130af68b84c124a7461e9b4209d3
SHA1: af6b1bc382457b3033d6286de33afaa86d6abf09
SHA256: 460131a068304561137c0447b6710438a80945202336f86f28ffab6891b1d1f1
SHA512: 42bff6df63229985c1a3f3ffda35d57fde14f64f5187b254f71aa16a83f5288e...(省略)
Homepage: https://www.openssl.org/
Description-en: Secure Sockets Layer toolkit - shared libraries
 This package is part of the OpenSSL project's implementation of the SSL
 and TLS cryptographic protocols for secure communication over the
 Internet.
 .
 It provides the libssl and libcrypto shared libraries.
Description-md5: 88547c6206c7fbc4fcc7d09ce100d210
Task: cloud-minimal, minimal, server-minimal
```

5.6　著作権情報の抽出

Debian 系 OS のパッケージは，著作権情報を `/usr/share/doc/パッケージ名/copyright` に必ず配置するよう定められている [9]. 標準の機械可読フォーマット [11] が定義されており，必須ではないが大部分のパッケージが従っているので，これに基づいて解析して著作権情報を抽出する．

5.6.1　copyright ファイルの構造

機械可読フォーマットは，stanza（節）とそれを構成する field（項目）の組み合わせで記述され，stanza 間は空行で分離される. 以下に stanza と field の一覧を，stanza については出現回数と内容，field については任意か必須か，形式，内容を併記して示す. 一例として libgmp10[*27]の copyright

[*27] `https://gmplib.org/`

ファイルを付録の Listing 12 に示す.

- **Header stanza (once):** パッケージ全体の著作権情報
  - **Format (required, single-line):** ファイルフォーマットを指定する. 現在は `https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/` とその HTTP 版のみ有効.
  - **Upstream-Name (optional, single-line):** 上流(入手元)で使われるソフトウェア名.
  - **Upstream-Contact (optional, line-based list):** 上流(入手元)の連絡先. メールアドレスなど.
  - **Source (optional, formatted text without synopsis):** 上流ソースコードの入手元. URL など.
  - **Disclaimer (optional, formatted text without synopsis):** Debian の一部でないパッケージについて, その理由.
  - **Comment (optional, formatted text without synopsis):** 補足情報.
  - **License (optional, formatted text with synopsis):** パッケージ全体のライセンス情報. 各ファイルのライセンスを組み合わせたものと比べて, 違っていたり簡略化されていたりする場合がある. 詳細な構文は後述.
  - **Copyright (optional, formatted text without synopsis):** パッケージ全体の著作権情報. 各ファイルの著作権情報を組み合わせたものと比べて, 違っていたり簡略化されていたりする場合がある.
- **Files stanza (repeatable):** ファイルごとの著作権情報
  - **Files (required, whitespace-separated list):** 本 stanza が対象とするファイル. ソースパッケージにおけるファイルパスのパターン形式で指定.
  - **Copyright (required, formatted text without synopsis):** 対象ファイルのみに適用される著作権情報.
  - **License (required, formatted text with synopsis):** 対象ファイルのみに適用されるライセンス情報. 詳細な構文は後述.
  - **Comment (optional, formatted text without synopsis):** 補足情報.
- **Stand-alone License stanza (optional, repeatable):** Header stanza や Files stanza で参照したライセンスの定義
  - **License (required, formatted text with synopsis):** 指定ライセンスの条文. 詳細な構文は後述.
  - **Comment (optional, formatted text without synopsis):** 補足情報.

値の形式は以下の通り.

Single-line values

1 行だけを使い，1 つの値を表す.

Whitespace-separated list

1 行以上に渡り，空白区切りで複数の値を表す.

Line-based list

1 行以上に渡り，1 行あたり 1 つの値を表す.

Formatted text with/without synopsis

`apt-cache show` における `Description-en:` と同様の形式で複数行の文字列を表す. with synopsis の場合，1 行目は特別な意味を持つ.

License field は，1 行目を synopsis として使用し，ライセンスの短縮名や，複数ライセンスや例外規定が適用される場合のライセンス式を記載する. そして，2 行目以降に本文を記載する. Header stanza や Files stanza においては，本文を省略して Stand-alone License stanza を参照することができる.

短縮名は copyright ファイル内で一貫して用いる必要があり，大文字小文字を区別せず，空白を含まない. また，複数のバージョンがある場合はハイフンに続く数字を末尾に付加し，あるバージョン以降の任意のバージョンを選択できる場合は末尾に + を付加する. 複数バージョンがあるライセンスでバージョン番号が省略されている場合は最も古いバージョンを意味し，バージョン番号末尾の `.0` の有無は区別せず同一とする. この短縮名については，一部がフォーマット仕様書内で定められている.

ライセンス式において，複数のライセンスのうち任意の一方に従えば良い場合は `or` を使い，両方に従う必要がある場合は `and` を使う. フォーマット仕様外だが `and/or` を使うファイルもあり，これは「任意の一方に従っても良いし，両方に従っても良い」という意味なので `or` と等価とみなす. 演算子の優先順位は `and` の方が強く，これを入れ替える場合にはカンマを用いて `A or B, and C` のように表現する. 複雑な式の途中に現れる場合に関しては厳密な定義がなされておらず曖昧性が残るが，実際のファイルの記述を精査した結果，該当のカンマから先頭方向に最も近いネストが同じ `and` 以降に対して適用されていると解釈する. ただし，フォーマット仕様書には記載がないものの，実際のファイルでは `A, B and C` のように自然言語の意味でカンマを使っている場合があるので，どちらの意味で使っているかを識別する必要がある. また，ライセンスの例外規定を指定する場合は 基底ライセンス `with` 例外名 `exception` と記載する. ただし，例外規定については標準名を使う必要がないため，機械的に既知の例外規定にマッピングするのは困難である.

5.6.2 SPDX の著作権関連項目

まず，パッケージメタデータとして以下の項目が存在する. 全て SPDX Lite の範囲内である.

- **licenseConcluded:** パッケージ全体のライセンスとして結論付けたライセンス式
- **licenseDeclared:** パッケージ中に宣言されたライセンス式
- **licenseComments:** 以上の項目に対する補足情報

- **copyrightText:** 著作権表記

そして，ライセンス式で独自ライセンスなどの SPDX License List[*28]に含まれないライセンスを記載できるようにするため，その本文などの情報を列挙する Other licensing information detected セクション（JSON スキーマでは `hasExtractedLicensingInfos`）があり，その要素は以下の項目で構成される．これも SPDX Lite の範囲内である．

- **licenseId:** パッケージメタデータ側から参照するために用いる，ライセンスに一意に割り当てた `LicenseRef-` から始まる識別子
- **name:** ライセンス名
- **extractedText:** ライセンス本文
- **comment:** 補足情報

SPDX におけるライセンス式は，SPDX License List 上に定義された短縮名あるいは独自に定義した LicenseRef のいずれかを項として，演算子 + （指定バージョン以上のライセンス），`WITH`（例外規定），`AND`（両方必須），`OR`（片方のみで良い）を適用する形で記述する．優先順位はこの順で + が最も強いが，() を使って変更できる．copyright ファイルとの類似点も見られるが，SPDX では `WITH` 演算子の例外規定は SPDX License Exceptions[*29]に掲載されているものだけが使用可能で，copyright ファイルのような自由度はない．また，copyright ファイルで定められている短縮名と SPDX License List で定められている短縮名の間に相違があることも注意が必要である．

5.6.3 解析

以上の仕様をもとに copyright ファイルの解析を行い，SPDX の項目に沿うよう変換する．

Header stanza からは Format，Comment，License，Copyright を読み込み，Comment と Copyright はパッケージの `licenseComments`，`copyrightText` にそれぞれ反映する．Format については対象の copyright ファイルがフォーマット仕様 [11] に従うことを確認するために使用し，満たさない場合はファイル解析を行わず，ファイル全文を単一のライセンス文書とみなして LicenseRef を発行し，`hasExtractedLicensingInfos` に出力する．License は後述の方法で解析し，パッケージの `licenseConcluded` と `licenseDeclared`，及び未知ライセンスがあった場合はそれらに LicenseRef を発行して `hasExtractedLicensingInfos` に出力する．copyright ファイル内に複数のライセンス式が記載されていた場合は，それらを全て `AND` 演算子で結合したものを `licenseConcluded` と `licenseDeclared` に設定する．演算子の作用対象を変化させないため，結合の際は括弧に囲われない `OR` 演算子を含む式に括弧を追加する．

Files stanza からは Copyright，License，Comment を読み込み，Header stanza と同様に処理す

[*28] `https://spdx.org/licenses/`

[*29] `https://spdx.org/licenses/exceptions-index.html`

る. それぞれ, 既にある場合は上書きせず追加する. Files field については, 本手法ではソースパッケージを使用しないこと, システムにインストールされたバイナリに対して正確に個別のソースファイルと対応付けするのは困難であることから扱わず, 全てパッケージ全体への記述として処理する.

Stand-alone License stanza からは License と Comment を読み込み, Header stanza と Files stanza で発行された LicenseRef に対応するものなら, 対応する `hasExtractedLicensingInfos` の項目に License field の本文と Comment field の補足情報を登録し, 対応しないものは無視する. SPDX License List には Public Domain にあたるものが登録されていないため SPDX ではライセンス本文が必要となるが, copyright ファイルでは public-domain のライセンス本文が省略されている場合があるので, その場合はフォーマット仕様 [11] に従い `No license required for any purpose; the work is not subject to copyright in any jurisdiction.` を代わりに使用する.

5.6.3.1 License field の構文解析

5.6.2 節の通り Licence field はそのまま SPDX に記載できる形式でないため, 別途解析を要する. Header stanza と Files stanza の License field については, まず synopsis であるライセンス式を解析する. 空白文字とカンマを区切り文字としてトークン分解し, 同時に `or` と `and/or` を `OR` に, `and` を `AND` に変換しておく. これにより, Listing 8 の 2 つのライセンス式は Listing 9 に処理される.

このトークン列に対して構文解析を行い, 構文の正しさを検証するとともに, フォーマット仕様 [11] に従い Perl ライセンスを `GPL-1.0-or-later OR Artistic-1.0` に変換する処理, 例外規定を含むライセンス表記を単一のライセンス名として 1 トークン化する処理, カンマを解決して括弧や演算子に変換する処理も同時に行う. 構文誤りが見つかった場合は, ライセンス式全体を単一のライセンス名(1 トークン)として処理する. これにより, Listing 9 は Listing 10 に処理される.

次に, 各ライセンス名についてヒューリスティックで SPDX License List と照合し, 見つかった場合は SPDX 短縮名に置き換える. このヒューリスティックでは, まずライセンス名を小文字化し, バージョン番号末尾の `.0` を削除(末端から順に, あとに . が続かない `.0` を全て削除する)した上で, 既知の対応表に従ってライセンス名を変換する. そこで合致しなかったものについては, SPDX License List 上の全ての短縮名について, 入力トークンと同様の正規化処理を施した状態で文字列比較し, 一致すれば対応する SPDX 短縮名で置き換える. 見つからなかった場合は LicenseRef を発行するが, これにあたってはトークン側の短縮名を小文字化した文字列をキーとする対応表を保持し, 後に同一のライセンスに同じ LicenseRef を発行したり, ライセンス本文などの情報を追加したりできるようにする. これにより, Listing 10 は Listing 11 に処理される[*30]. 2 つ目の例は `libldap2:amd64` パッケージで見られたものであるため, LicenseRef にはその旨が含まれている.

[*30] copyright ファイルにおける Expat ライセンスは MIT ライセンスのことである. MIT ライセンスには当初無数の亜種が存在し, SPDX License List で標準形が示されるまでは同定する術がなかったため, 「Expat ライブラリで使用された MIT ライセンス」という意味で Expat の名称が使われている [15].

Listing 8　ライセンス式の例

```
1 GPL-1+ or Artistic, and Expat
2 GPL-2+ with Libtool exception and GPL-3+ with Libtool exception and GPL-3+
```

Listing 9　トークン分割後

```
1 [ "GPL-1+", "OR", "Artistic", ",", "AND", "Expat" ]
2 [ "GPL-2+", "with", "Libtool", "exception", "AND", "GPL-3+", "with", "Libtool", "
      exception", "AND", "GPL-3+" ]
```

Listing 10　構文解析後

```
1 [ "(", "GPL-1+", "OR", "Artistic", ")", "AND", "Expat" ]
2 [ "GPL-2+ with Libtool exception", "AND", "GPL-3+ with Libtool exception", "AND", "GPL
      -3+" ]
```

Listing 11　ライセンス照合後

```
1 [ "(", "GPL-1.0-or-later", "OR", "Artistic-1.0", ")", "AND", "MIT" ]
2 [ "LicenseRef-libldap2-amd64--gpl-2.0-or-later-with-libtool-exception", "AND", "
      LicenseRef-libldap2-amd64--gpl-3.0-or-later-with-libtool-exception", "AND", "GPL
      -3.0-or-later" ]
```

5.7　SBOM の生成

以上で抽出した情報を用いて, 以下に示す項目を JSON 形式に変換してファイル出力し, 最終的な SBOM を生成する. なお，SBOM 生成対象とするソフトウェア自体のソフトウェア名, バージョン, 開発者, ライセンス, 著作権などの情報は, ユーザ入力から受け取る.

- **spdxVersion:** 固定値で `SPDX-2.3` を使用.
- **dataLicense:** 固定値で `CC0-1.0` を使用.
- **SPDXID:** 固定値で `SPDXRef-DOCUMENT` を使用.
- **name:** ユーザ入力の対象ソフトウェア名を使用.
- **documentNamespace:** ユーザ入力の対象ソフトウェア名を含めて自動生成.
- **creationInfo**
  - **created:** 生成日時を自動入力.
  - **creators:** ツール名を自動入力. 任意でユーザ入力も受け付ける.
- **documentDescribes:** 固定値で `SPDXRef-RootPackage` を使用.
- **packages:** 5.3 節以降で抽出. 対象ソフトウェア自体を指す親パッケージは, ユーザ入力の情報を用いて name，SPDXID（固定値で `SPDXRef-RootPackage`），versionInfo, downloadLocation（固定値で `NOASSERTION`），filesAnalyzed（固定値で `false`），supplier，licenseConcluded，licenseDeclared，copyrightText のみを出力する.
  - **name**
  - **SPDXID:** パッケージアーカイブの SHA1 値, 抽出できなければ UUID (Universally Unique Identifier) 値を使って生成した文字列を自動設定.

  - **versionInfo**
  - **packageFileName**
  - **downloadLocation**
  - **filesAnalyzed:** 固定値で `false` を使用.
  - **homepage**
  - **supplier**
  - **originator**
  - **summary**
  - **description**
  - **checksums**
  - **externalRefs**
  - **copyrightText**
  - **licenseConcluded**
  - **licenseDeclared**
  - **licenseComments**
  - **comment**
- **hasExtractedLicensingInfos:** 5.6 節で抽出.
  - **licenseId**
  - **name**
  - **extractedText**
  - **comment**
- **relationships:** 固定値で `SPDXRef-DOCUMENT` から `SPDXRef-RootPackage` への `DESCRIBES` を記述し, それ以外は全パッケージについて `DEPENDS_ON` で記述. 直接対象のソフトウェアがビルド時・実行時に使用する依存関係を抽出しているから, この表記が実態に即している.

6 開発した手法の評価

本節では，開発した手法の評価として，依存関係とメタデータを抽出する能力を調べるとともに，生成される SBOM の SPDX 規格や NTIA 要件への準拠度を調べる．そのために，まず C/C++ のソフトウェアに対して実際に SBOM を生成することで，依存ファイルを抽出し，それを提供するパッケージを特定し，パッケージメタデータを抽出する一連の手法の性能を評価する．しかし，これでは数十個のパッケージメタデータしか評価対象とならないため，システム上にインストールされた全パッケージに対する SBOM も生成し，本手法におけるパッケージメタデータを抽出する部分の汎用性をより詳しく評価する．そして，SPDX 公式が提供する，SPDX フォーマットとしての正しさを検証する SPDX Validator[*31]と，NTIA 要件の準拠度を検証する NTIA Conformance Checker[*32]を，以上で生成された各 SBOM に適用する．

本評価において，当該項目に対して有意な値を設定する上で必要な情報を，前節で説明した手法により取得できた場合に抽出成功とする．`checksums` については，1 つ以上のアルゴリズムによるチェックサム値が取得できていれば抽出成功とする．

6.1 C/C++ のソフトウェアに対する SBOM 生成

著名なネットワークツール curl[*33]と Web サーバソフトウェア Apache HTTP Server[*34]は C 言語で開発されている．これらは豊富な機能の提供にあたり外部ライブラリを活用しており，特に HTTPS 通信のために OpenSSL を使用している．OpenSSL は過去に重大な脆弱性が繰り返し報告されており [48]，今後発見された際には早急な対応が必要となる．ここで，本手法が curl や Apache HTTP Server の OpenSSL 依存を検出して SBOM 内に出力すれば，他の C/C++ のソフトウェアについても同様に OpenSSL 依存を確認し，新たな脆弱性が発見された際の迅速な対応に役立てられることを確認できる．そこで，本評価における SBOM 生成対象として curl 8.10.1[*35]と Apache HTTP Server 2.4.63[*36]を選定した．バージョンは共に実験時点での最新版である．curl はビルド時に有効化する機能を選択できるが，OpenSSL ライブラリを使用する `--with-openssl` オプションを与えた．

6.1.1 ビルド時に組み込まれる依存関係

まず，ビルド時に組み込まれる依存関係について SBOM 生成を行った．その結果，curl では 316 件の依存ファイルが抽出され，それら全ての出自が特定されて `libssl-dev`（OpenSSL の開発ライブラリで，ヘッダファイルを提供する）と `libssl3t64`（OpenSSL のバイナリライブラリ）を含む 9 件の

[*31] `https://tools.spdx.org/app/validate/`
[*32] `https://tools.spdx.org/app/ntia_checker/`
[*33] `https://curl.se/`
[*34] `https://httpd.apache.org/`
[*35] `https://github.com/curl/curl/releases/download/curl-8_10_1/curl-8.10.1.tar.xz`
[*36] `https://dlcdn.apache.org/httpd/httpd-2.4.63.tar.gz`

依存パッケージが抽出された．Apache HTTP Server では 390 件の依存ファイルが抽出され，それら全ての出自が特定されて同様に `libssl-dev` と `libssl3t64` を含む 17 件の依存パッケージが抽出された．

抽出されたメタデータは表 4 の通りである．生成時間は curl では 27.9 秒，Apache では 37.2 秒であった．NTIA 要件に含まれる項目を太字で示す．固定値やツール側で自ら生成する値など手法の性能に関わらず 100% となる項目，上流のメンテナが存在する場合のみ記載する `originator` 項目，何か補足情報がある場合に使用するコメント項目は省略する．

表 4　ビルド時に組み込まれる依存関係のメタデータ抽出率

項目	curl	Apache HTTP Server
name	**100% (9)**	**100% (17)**
versionInfo	**100% (9)**	**100% (17)**
supplier	**100% (9)**	**100% (17)**
externalRefs (PURL)	**100% (9)**	**100% (17)**
packageFileName	100% (9)	100% (17)
downloadLocation	100% (9)	100% (17)
homepage	88.9% (8)	82.4% (14)
summary	100% (9)	100% (17)
description	100% (9)	100% (17)
checksums	100% (9)	100% (17)
copyrightText	33.3% (3)	29.4% (5)
licenseConcluded/licenseDeclared	88.9% (8)	94.1% (16)

`homepage` は `apt-cache` に当該情報がなかったため，ライセンス情報は copyright ファイルが所定の場所に存在しなかったため，一部のパッケージについて抽出に失敗した．`copyrightText` は，curl では copyright ファイルが存在しなかったパッケージ 1 件と存在したが機械可読フォーマットでなかったパッケージ 5 件，Apache HTTP Server ではそれぞれ 1 件と 11 件で抽出に失敗した．しかし，これらは NTIA 要件外の項目であり，太字で示した NTIA 要件内の項目は全て 100% と，ビルド時に組み込まれる依存関係について NTIA 要件を満たす SBOM を生成できていることがわかる．実際に，SPDX Validator と NTIA Conformance Checker をいずれも通過した．

ライセンス情報は，curl では 14 件中 9 件，Apache HTTP Server では 47 件中 32 件が SPDX License List 外の特殊ライセンスとして判定され，これら特殊ライセンスについて表 5 の通り，情報が抽出された．copyright ファイルが機械可読フォーマットでなかったために全文をライセンスとみなして処理したパッケージでライセンス名を抽出できなかった点を除き，全ての情報を抽出できた．特殊ライセンスとして判定されたものは gnulib，chromium，例外条項付きライセンス，MIT ライセンスや BSD ライセンスの亜種，機械可読でない copyright ファイルなどで，この判定は正しいものであった．

表 5　ビルド時に組み込まれる依存関係のライセンス情報抽出率

項目	curl	Apache HTTP Server
name	44.4% (4)	65.6% (21)
extractedText	100% (9)	100% (32)

また，ライセンス名を同定するヒューリスティックの性能を調べるため，ヒューリスティック無効と有効でライセンス同定能力がどの程度変化するかを調査した．その結果，ヒューリスティック無効では `Artistic`，`GPL-1+`，`BSD-3-clause`（c が小文字）などが同定できず特殊ライセンスと判定され，同定されたライセンスが curl では 5 件から 3 件に，Apache HTTP Server では 15 件から 5 件に減少したことから，ヒューリスティックが有益に機能していることを確認できた．

6.1.2　実行時の依存関係

次に，実行時の依存関係について SBOM 生成を行った．その結果，curl では 31 件の依存ファイルが抽出され，それら全ての出自が特定されて `libssl3t64` を含む 27 件のパッケージが抽出された．Apache HTTP Server では 7 件の依存ファイルが抽出され，それら全ての出自が特定されて 7 件のパッケージが抽出された．Apache HTTP Server では OpenSSL が静的リンクされていたためか，実行時の依存関係ではなかった．

抽出されたメタデータを表 6 に，ビルド時に組み込まれる依存関係と同様の形式で示す．生成時間は curl では 14.0 秒，Apache HTTP Server では 4.3 秒であった．

表 6　実行時の依存関係のメタデータ抽出率

項目	curl	Apache HTTP Server
name	**100% (27)**	**100% (7)**
versionInfo	**100% (27)**	**100% (7)**
supplier	**100% (27)**	**100% (7)**
externalRefs (PURL)	**100% (27)**	**100% (7)**
packageFileName	100% (27)	100% (7)
downloadLocation	100% (27)	100% (7)
homepage	100% (27)	85.7% (6)
summary	100% (27)	100% (7)
description	100% (27)	100% (7)
checksums	100% (27)	100% (7)
copyrightText	70.4% (19)	42.9% (3)
licenseConcluded/licenseDeclared	100% (27)	100% (7)

`homepage` は `apt-cache` に当該情報がなかったため，Apache HTTP Server において 1 件のパッケージで抽出に失敗した．`copyrightText` は，共に copyright ファイルが機械可読フォーマットでなかったパッケージで抽出に失敗した．しかし，それ以外は全て 100% と，実行時の依存関係について NTIA 要件を満たす SBOM を生成できていることがわかる．実際に，SPDX Validator と NTIA Conformance Checker をいずれも通過した．

ライセンス情報は，curl では 140 件中 58 件，Apache HTTP Server では 22 件中 8 件が SPDX License List 外の特殊ライセンスとして判定され，これら特殊ライセンスについて表 7 の通り，情報が抽出された．copyright ファイルが機械可読フォーマットでなかったために全文をライセンスとみなして処理したパッケージでライセンス名を抽出できなかった点を除き，全ての情報を抽出できた．特殊ライセンスとして判定されたものは MIT ライセンスや BSD ライセンスの亜種，例外条項付きライセンス，パブリックドメイン，独自ライセンス，機械可読でない copyright ファイルなどで，この判定は正しいものであった．また，ビルド時に組み込まれる依存関係と同様にライセンス名のヒューリスティックの性能を調べたところ，ヒューリスティック無効では `Artistic`，`GPL-1+`，`zlib`，`Expat`，`BSD-3-clause`（`c` が小文字）などが同定できず特殊ライセンスと判定され，同定されたライセンスが curl では 82 件から 31 件に，Apache HTTP Server では 14 件から 4 件に減少したことから，ヒューリスティックが有益に機能していることを確認できた．

表 7　実行時の依存関係のライセンス情報抽出率

項目	curl	Apache HTTP Server
name	86.2% (50)	50.0% (4)
extractedText	100% (58)	100% (8)

6.2　パッケージメタデータの抽出能力

最後に，実際に Web サーバなどの役割を担い本番稼働している Ubuntu 24.04 サーバと，仮想環境上に構築した Debian 12 と Linux Mint 22.1 に対し，インストールされた全てのパッケージの SBOM を生成し，メタデータ抽出能力を評価した．パッケージ数はそれぞれ 984 件，1677 件，2523 件であった．その結果を前節と同様の形式で表 8 に示す．

いくつかの項目に漏れがあるが高い水準で抽出できており，太字で示した NTIA 要件内の項目は全て 100% と，3 つ全ての環境で全パッケージについて NTIA 要件を満たす SBOM を生成できていることがわかる．実際に，SPDX Validator と NTIA Conformance Checker をいずれも通過した．

実稼働中の Ubuntu サーバについて，ライセンス情報は 3467 件のうち 1187 件が SPDX License List 外の特殊ライセンスとして判定され，その 1187 件について表 9 の通り，高い水準で情報が抽出された．また，前節と同様にライセンス名のヒューリスティックの性能を調べたところ，ヒューリスティック有効では 2280 件同定されていたものが 855 件まで減少したことから，ヒューリスティックが有益に機能していることを確認できた．

表 8　システム上の全パッケージのメタデータ抽出率

項目	Ubuntu	Debian	Linux Mint
name	**100% (984)**	**100% (1677)**	**100% (2523)**
versionInfo	**100% (984)**	**100% (1677)**	**100% (2523)**
supplier	**100% (984)**	**100% (1677)**	**100% (2523)**
externalRefs (PURL)	**100% (984)**	**100% (1677)**	**100% (2523)**
packageFileName	95.2% (937)	99.9% (1676)	98.7% (2491)
downloadLocation	95.2% (937)	99.9% (1676)	98.7% (2491)
homepage	81.2% (799)	87.8% (1472)	86.0% (2170)
summary	94.5% (930)	99.9% (1676)	91.0% (2295)
description	94.5% (930)	99.9% (1676)	90.9% (2294)
checksums	95.2% (937)	99.9% (1676)	98.7% (2491)
copyrightText	70.4% (693)	72.8% (1220)	76.4% (1927)
licenseConcluded/licenseDeclared	91.2% (897)	100% (1677)	98.7% (2489)

表 9　システム上の全パッケージのライセンス情報抽出率

項目	Ubuntu
name	82.0% (973)
extractedText	98.3% (1167)

6.3　考察

以上の実験結果から，本手法はビルド時に組み込まれる依存関係と実行時の依存関係について，全ての依存ファイルに対して出自パッケージを特定し，SPDX 規格と NTIA 要件に準拠した SBOM の生成に必要なメタデータを抽出した．抽出された依存関係に OpenSSL が含まれることから，その正確性も確認した．実行時間も共に 1 分以内と，開発者の作業を妨げずに SBOM 生成できる．

システムにインストールされた全パッケージに対する実験でも，3 つ全ての環境で SPDX 規格と NTIA 要件に準拠した SBOM を生成した．さらに，NTIA 要件外のメタデータも高い水準で抽出したことから，本手法のパッケージメタデータ抽出処理は高い汎用性を持ち，NTIA 要件を超える有用性の高い SBOM を生成する能力があるといえる．

NTIA 要件外であるライセンス情報の抽出にかかるヒューリスティックに関しても，高い性能を確認した．ライセンスコンプライアンス管理においてライセンスの互換性を判断する際，SPDX License List に掲載された既知のライセンスに対応付けられていれば，その手間が大きく削減される．本手法のヒューリスティックは，これに貢献できる性能を持つといえる．

7　今後の課題

本手法では，ビルド時に組み込まれる依存関係を抽出する際に，ビルド時に使用されるシステム上のヘッダファイルとバイナリライブラリの情報を使用した．しかし，ビルド時の依存関係を網羅的に把握するためには，システム上の外部ライブラリだけでなく，コードベース中に取り込まれた外部ライブラリの情報も重要である．C/C++ のコードベースに含まれる依存関係を抽出する手法 [29] の発展など，これらの依存関係を抽出する手法の開発が必要であろう．

本手法では，Ubuntu を含む Debian 系の Linux ディストリビューションにおいて，GCC でビルドする C/C++ のソフトウェアを対象とした．しかし，エンタープライズ用途では堅牢な RedHat 系の Linux ディストリビューションも広く用いられるほか，Windows を用いる組織も多い．また，コンパイラも Clang や MSVC など，GCC 以外のコンパイラで開発される C/C++ のソフトウェアも数多い．本手法を真に実用的なものとするには，これらの環境で依存関係を抽出する手法を開発し，より手法の汎用性を高める必要があろう．

本手法では，ライセンス情報の抽出にあたり，機械可読フォーマットを採用しない copyright ファイルは特に解析せず，ファイル内容を丸ごとライセンス文として処理した．しかし，得られたメタデータをライセンスコンプライアンス管理で活用する上では，こうした copyright ファイルについても解析し，その中に記述されたライセンス情報や著作権情報を識別するような改良が望ましい．

本手法の評価にあたっては curl と Apache HTTP Server を対象としたが，これだけで全ての C/C++ のソフトウェアに対する適用性を示せたとはいえない．今後，評価対象のソフトウェアを増やすことで，手法の汎用性や有用性をさらに深く検証する必要があろう．

8 おわりに

本研究では，多くの外部ライブラリを活用してソフトウェア開発を行う昨今，サプライチェーンリスクの迅速な把握と対応に貢献すると期待されている SBOM について，その普及を妨げる原因を調査し，その結果明らかとなった課題を解決するための手法を開発した．

課題抽出では主要な開発者向け Q&A サイトである Stack Overflow において，SBOM 利活用に関する質問を調査した．その結果，一般のソフトウェア開発者が直面している，SBOM の普及に際して解決すべき技術課題は「SBOM ツールのユースケースの網羅性に不足がある」，「各種要件を満たす SBOM を生成できない」，「SBOM ツールに不具合がある，または利用方法が不明瞭である」の 3 つであることがわかった．

この調査結果を受けて，SBOM 普及に資するため，既存ツールではサポートされないユースケースである C 言語及び C++ 言語で開発されたソフトウェアに対し，米国政府機関 NTIA が定める要件を満たす SBOM を自動生成する手法を開発した．パッケージマネージャが存在しない C/C++ のソフトウェアに対して SBOM を生成するにあたり，ビルド処理から得られる情報と，その結果生成されるバイナリに含まれるメタデータに着目した．本手法を著名なネットワークツール curl と Web サーバソフトウェア Apache HTTP Server に適用したところ，curl ではビルド時に組み込まれる依存関係 9 件と実行時の依存関係 27 件を特定し，Apache HTTP Server ではそれぞれ 17 件と 7 件を特定し，全てについて NTIA 要件を満たす SBOM を実用的な実行時間で生成できた．また，本手法のメタデータ抽出処理の汎用性を検証するため，Ubuntu サーバ，Debian 仮想マシン，Linux Mint 仮想マシンにインストールされた全パッケージ 984 件，1677 件，2523 件についてメタデータ抽出処理を適用したところ，全パッケージについて NTIA 要件を満たす SBOM を生成でき，NTIA 要件外のメタデータについても高い水準で抽出できたことから，本手法のメタデータ抽出処理は高い汎用性を持ち，NTIA 要件を超える有用性の高い SBOM を生成できることを示した．

謝辞

本研究を行うにあたり，方向性や発表方法などご多忙の中懇切なるご指導，ご助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後 芳樹 教授に，心から感謝の意を表します．

本研究の全過程を通して，方向性の検討やアイデアの立案，論文の添削など，手厚くご指導，ご助言を賜りました南山大学理工学部ソフトウェア工学科 井上 克郎 教授に心から感謝の意を表します．

本研究の全過程を通して，方向性の検討や懸念点の指摘，論文の添削など手厚くご指導，ご助言を賜りました福知山公立大学情報学部情報学科 眞鍋 雄貴 講師に心から感謝の意を表します．

本研究の全過程を通して，研究の進捗管理，方向性の整理，論文や発表内容の添削など手厚くご指導，ご助言を賜り，加えて研究生活におけるご相談にも乗ってくださったノートルダム清心女子大学情報デザイン学部情報デザイン学科 神田 哲也 准教授に心から感謝の意を表します．

実社会の課題を共有して研究テーマのシーズを提供するなど，本研究を行うにあたり不可欠なご助言を賜りました株式会社東芝デジタルイノベーションテクノロジーセンター 仇 実 氏に心から感謝の意を表します．

本研究を行うにあたり，アイデアの提案や方向性の整理，加えて計算機資源の管理上のアドバイスなどご指導，ご助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授に心から感謝の意を表します．

本研究を行うにあたり，研究成果のまとめ方などご指導，ご助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 Raula Gaikovina Kula 教授に心から感謝の意を表します．

また，大阪大学大学院情報科学研究科コンピュータサイエンス専攻 岸本 理央 氏には，同じくSBOM 関連の研究を遂行する立場で SBOM に関する知見やその他の助言など，本研究を行うにあたり不可欠なお力添えをいただきました．

最後に，日々ご助言やご協力頂き，激励を賜りました肥後研究室の皆様ならびに事務職員 軽部 瑞穂 氏に心から感謝致します．

参考文献

[1] Anchor. Syft. https://github.com/anchore/syft. Accessed Jan 2025.

[2] Bill Bensing. History of the Software Bill of Material (SBOM). https://billbensing.com/software-supply-chain/history-software-bill-of-material-sbom/, 2022.

[3] Tingting Bi, Boming Xia, Zhenchang Xing, Qinghua Lu, and Liming Zhu. On the Way to SBOMs: Investigating Design Issues and Solutions in Practice. *ACM Trans. Softw. Eng. Methodol.*, Vol. 33, No. 6, June 2024.

[4] Joseph Robinette Biden, Jr. Executive Order on Improving the Nation's Cybersecurity. https://www.federalregister.gov/executive-order/14028, May 2021.

[5] Black Duck Software. 2024 Open Source Security and Risk Analysis Report. https://www.blackduck.com/resources/analyst-reports/open-source-security-risk-analysis.html, 2024.

[6] Jaewoo Cho. Understanding Open Source: Compliance and Enforcement. *The SciTech Lawyer*, Vol. 10, No. 4, 2014.

[7] Lasse Collin. XZ Utils backdoor. https://tukaani.org/xz-backdoor/, July 2024.

[8] Cybersecurity and Infrastructure Security Agency. Types of Software Bill of Material (SBOM) Documents. https://www.cisa.gov/sites/default/files/2023-04/sbom-types-document-508c.pdf, 2023.

[9] Debian. 12. Documentation – Debian Policy Manual 4.7.0.2. https://www.debian.org/doc/debian-policy/ch-docs, January 2025.

[10] Debian. 5. Control files and their fields – Debian Policy Manual 4.7.0.2. https://www.debian.org/doc/debian-policy/ch-controlfields, January 2025.

[11] Debian. Machine-readable debian/copyright file. https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/, January 2025.

[12] Debian Manpages. apt(8) – apt – Debian bookworm. https://manpages.debian.org/bookworm/apt/apt.8, April 2020.

[13] Debian Manpages. ldd(1) – manpages – Debian bookworm. https://manpages.debian.org/bookworm/manpages/ldd.1, February 2023.

[14] European Commission. Cyber Resilience Act. https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act, September 2022.

[15] Free Software Foundation. Various Licenses and Comments about Them - GNU Project. https://www.gnu.org/licenses/license-list, November 2024.

[16] GitHub. Exporting a software bill of materials for your repository. https://docs.github.com/en/code-security/supply-chain-security/understanding-your-softw

are-supply-chain/exporting-a-software-bill-of-materials-for-your-repository. Accessed Jan 2025.

[17] GitHub Staff. Octoverse: AI leads Python to top language as the number of global developers surges. https://github.blog/news-insights/octoverse/octoverse-2024, October 2024.

[18] Stephen Hendrick and Jim Zemlin. The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness. Technical report, Linux Foundation, January 2022.

[19] 経済産業省 商務情報政策局 サイバーセキュリティ課. ソフトウェア管理に向けた SBOM（Software Bill of Materials）の導入に関する手引 Ver. 1.0. https://www.meti.go.jp/press/2023/07/20230728004/20230728004-1-2.pdf, July 2023.

[20] 経済産業省 商務情報政策局 サイバーセキュリティ課. ソフトウェア管理に向けた SBOM（Software Bill of Materials）の導入に関する手引 ver. 2.0. https://www.meti.go.jp/press/2024/08/20240829001/20240829001-1r.pdf, August 2024.

[21] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. Structure and Evolution of Package Dependency Networks. In *Proc. MSR2017*, pp. 102–112, 2017.

[22] Rio Kishimoto, Tetsuya Kanda, Yuki Manabe, Katsuro Inoue, and Yoshiki Higo. Osmy: A Tool for Periodic Software Vulnerability Assessment and File Integrity Verification using SPDX Documents. In *Proc. SANER2024*, pp. 445–449, March 2024.

[23] Berend Kloeg, Aaron Yi Ding, Sjoerd Pellegrom, and Yury Zhauniarovich. Charting the Path to SBOM Adoption: A Business Stakeholder-Centric Approach. In *Proc. ASIACCS2024*, p. 1770 − 1783, New York, NY, USA, 2024.

[24] Wayne C. Lim. *Managing software reuse: a comprehensive guide to strategically reengineering the organization for reusable components*. Prentice Hall, c1998.

[25] Linux Foundation. About SPDX. https://spdx.dev/about/overview/. Accessed Jan 2025.

[26] Linux Foundation and its Contributors. Annex E: Using SPDX short identifiers in Source Files – specification v2.3.0. https://spdx.github.io/spdx-spec/v2.3/using-SPDX-short-identifiers-in-source-files/, 2022.

[27] Linux Foundation and its Contributors. Annex G: SPDX Lite – specification v2.3.0. https://spdx.github.io/spdx-spec/v2.3/SPDX-Lite/, 2022.

[28] Microsoft. SBOM Tool. https://github.com/microsoft/sbom-tool. Accessed Jan 2025.

[29] Yoonjong Na, Seunghoon Woo, Joomyeong Lee, and Heejo Lee. CNEPS: A Precise Approach for Examining Dependencies among Third-Party C/C++ Open-Source Components. In *Proc. ICSE2024*, New York, NY, USA, 2024.

[30] Sabato Nocera, Massimiliano Di Penta, Rita Francese, Simone Romano, and Giuseppe Scanniello. If it's not SBOM, then what? How Italian Practitioners Manage the Software Supply Chain. In *Proc. ICSME2024*, pp. 730–740, October 2024.

[31] Sabato Nocera, Simone Romano, Massimiliano Di Penta, Rita Francese, and Giuseppe Scanniello. Software Bill of Materials Adoption: A Mining Study from GitHub. In *Proc. ICSME2023*, pp. 39–49, October 2023.

[32] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In *Proc. DIMVA2020*, pp. 23–43, 2020.

[33] OWASP Foundation. CycloneDX - software bill of materials (SBOM). `https://cyclonedx.org/capabilities/sbom/`. Accessed Jan 2025.

[34] purl authors. Package URL Type definitions. `https://github.com/package-url/purl-spec/blob/master/PURL-TYPES.rst#deb`. Accessed Jan 2025.

[35] Q-Success DI Gelbmann GmbH. Usage Statistics and Market Share of Linux for Websites, January 2025. `https://w3techs.com/technologies/overview/os-linux`, January 2025.

[36] Odd Petter N. Slyngstad, Anita Gupta, Reidar Conradi, Parastoo Mohagheghi, Harald Rønneberg, and Einar Landre. An Empirical Study of Developers Views on Software Reuse in Statoil ASA. In *Proc. ISESE2006*, pp. 242–251, 2006.

[37] Stack Exchange Inc. All Sites - Stack Exchange. `https://stackexchange.com/sites`. Accessed Feb 2024.

[38] Stack Overflow. Usage of /info [GET] - Stack Exchange API. `https://api.stackexchange.com/docs/info#filter=default&site=stackoverflow&run=true`. Accessed Feb 2024.

[39] Stack Overflow. 2024 Stack Overflow Developer Survey. `https://survey.stackoverflow.co/2024/`, 2024.

[40] Trevor Stalnaker, Nathan Wintersgill, Oscar Chaparro, Massimiliano Di Penta, Daniel M German, and Denys Poshyvanyk. BOMs Away! Inside the Minds of Stakeholders: A Comprehensive Study of Bills of Materials for Software Systems. In *Proc. ICSE2024*, 2024.

[41] The European Union Agency for Cybersecurity. Guidelines for Securing the Internet of Things. `https://www.enisa.europa.eu/publications/guidelines-for-securing-the-internet-of-things`, 2020.

[42] The United States Department of Commerce. The Minimum Elements For a Software Bill of Materials (SBOM). `https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom`, 2021.

[43] The White House. Initial Rescissions Of Harmful Executive Orders And Actions. https://www.whitehouse.gov/presidential-actions/2025/01/initial-rescissions-of-harmful-executive-orders-and-actions/, January 2025.

[44] Ying Wang, Bihuan Chen, Kaifeng Huang, Bowen Shi, Congying Xu, Xin Peng, Yijian Wu, and Yang Liu. An Empirical Study of Usages, Updates and Risks of Third-Party Libraries in Java Projects. In *Proc. ICSME 2020*, pp. 35–45, 2020.

[45] Evan D. Wolff, Kate M. Growley, Maida O. Lerner, Matthew B. Welling, Michael G. Gruden, and Jacob Canter. Navigating the SolarWinds Supply Chain Attack. *The Procurement Lawyer*, Vol. 56, No. 2, pp. 3–10,28, 2021.

[46] Boming Xia, Tingting Bi, Zhenchang Xing, Qinghua Lu, and Liming Zhu. An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead. In *Proc. ICSE2023*, pp. 2630–2642, 2023.

[47] 田邉傑士, 神田哲也, 眞鍋雄貴, 井上克郎, 肥後芳樹. Debian パッケージ間の依存関係を表す SPDX ドキュメント自動生成ツールの開発. 電子情報通信学会論文誌 D, Vol. J106-D, No. 9, pp. 457–458, September 2023.

[48] 独立行政法人情報処理推進機構. OpenSSL の脆弱性対策について (CVE-2022-3602、CVE-2022-3786). https://www.ipa.go.jp/security/security-alert/2022/alert20221102.html, November 2022.

付録

Listing 12　libgmp10 の copyright ファイル

```
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: GMP
Source: http://gmplib.org/
Comment: This gmp package was built for Debian by
  Steve M. Robbins <smr@debian.org>
  Philipp Matthias Hahn <pmhahn@debian.org>
 .
 The documentation is released under the GNU Free Documentation License
 (GFDL) and it has cover texts.  As such, it has been determined not to
 meet the Debian Free Software Guidelines, and is not shipped in the
 debian packages.
Files-Excluded: doc

Files: *
Copyright: 1991, 1996, 1999, 2000, 2007 Free Software Foundation, Inc.
License: GPL-2+ or LGPL-3+
Comment:
 This file is part of the GNU MP Library.
 .
 The GNU MP Library is free software; you can redistribute it and/or modify
 it under the terms of either:
 .
  * the GNU Lesser General Public License as published by the Free
    Software Foundation; either version 3 of the License, or (at your
    option) any later version.
 .
 or
 .
  * the GNU General Public License as published by the Free Software
    Foundation; either version 2 of the License, or (at your option) any
    later version.
 .
 or both in parallel, as here.
 .
 The GNU MP Library is distributed in the hope that it will be useful, but
 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
 or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 for more details.
 .
 You should have received copies of the GNU General Public License and the
 GNU Lesser General Public License along with the GNU MP Library.  If not,
 see https://www.gnu.org/licenses/.

Files: demos/calc/*.[chyl]
 demos/factorize.c demos/isprime.c demos/primes.[ch] demos/qcn.c demos/pexpr*
Copyright: 1995, 1997-2003, 2005, 2006, 2008, 2009, 2012, 2015 Free Software
    Foundation, Inc.
License: GPL-3+

Files: demos/calc/calc.[ch]
Copyright: (C) 1984, 1989-1990, 2000-2015, 2018-2020 Free Software Foundation, Inc.
License: GPL-3+ with Bison exception
   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.
 .
```

```
57    This program is distributed in the hope that it will be useful,
58    but WITHOUT ANY WARRANTY; without even the implied warranty of
59    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
60    GNU General Public License for more details.
61  .
62    You should have received a copy of the GNU General Public License
63    along with this program.  If not, see <http://www.gnu.org/licenses/>.
64  .
65    As a special exception, you may create a larger work that contains
66    part or all of the Bison parser skeleton and distribute that work
67    under terms of your choice, so long as that work isn't itself a
68    parser generator using the skeleton or a modified version thereof
69    as a parser skeleton.  Alternatively, if you modify or redistribute
70    the parser skeleton itself, you may (at your option) remove this
71    special exception, which will cause the skeleton and the resulting
72    Bison output files to be licensed under the GNU General Public
73    License without this special exception.
74
75 License: GPL-2+
76  The GNU General Public License v2 text is contained in /usr/share/common-licenses/GPL
       -2.
77
78 License: GPL-3+
79  This program is free software; you can redistribute it and/or modify it under
80  the terms of the GNU General Public License as published by the Free Software
81  Foundation; either version 3 of the License, or (at your option) any later
82  version.
83  .
84  This program is distributed in the hope that it will be useful, but WITHOUT ANY
85  WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
86  PARTICULAR PURPOSE.  See the GNU General Public License for more details.
87  .
88  You should have received a copy of the GNU General Public License along with
89  this program.  If not, see <https://www.gnu.org/licenses/>.
90  .
91  The GNU General Public License v3 text is contained in /usr/share/common-licenses/GPL
       -3.
92
93 License: LGPL-3+
94  The GNU Lesser General Public License v3 text is contained in /usr/share/common-
       licenses/LGPL-3.
```