一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

# GitHub 上の OSS を対象とする SBOM データセット構築の試み

岸本　理央† 神田　哲也†† 眞鍋　雄貴††† 井上　克郎†††† 仇　　実†††††
肥後　芳樹†

† 大阪大学大学院情報科学研究科 〒565–0871 大阪府吹田市山田丘 1–5
†† 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 〒700–8516 岡山県岡山市北区伊福町 2-16-9
††† 福知山公立大学情報学部情報学科 〒620–0886 京都府福知山市堀 3370
†††† 南山大学理工学部ソフトウェア工学科 〒466–8673 愛知県名古屋市昭和区山里町 18
††††† 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 〒212–8582 神奈川県川崎市幸区小向東芝町 1
E-mail: †{r-kisimt,higo}@ist.osaka-u.ac.jp, ††kanda@m.ndsu.ac.jp, †††manabe-yuki@fukuchiyama.ac.jp,
††††inoue599@nanzan-u.ac.jp, †††††shi1.qiu@toshiba.co.jp

**あらまし**　近年，ソフトウェア部品表（Software Bill of Materials, SBOM）を用いたソフトウェアの管理が推奨されている．しかし，ソフトウェアの SBOM が開発者によって提供されている例は少なく，SBOM の普及は進んでいない．その要因として，SBOM に関連するツールの機能の不足や不完全さが指摘されている．この課題を解決するために，GitHub で公開されている OSS の SBOM を作成し，それらを SBOM データセットとして提供することを目指す．本論文では，SBOM データセットの要件を整理し，SBOM データセットを作成するうえで既存の SBOM 生成ツールを利用する際の課題を調査した．調査の結果，複数の SBOM 生成ツールを組みわせることで，SBOM データセットの要件を満たす SBOM を作成できることが分かった．

**キーワード**　SBOM, SPDX, OSS, GitHub

## 1. ま え が き

ソフトウェア開発におけるライブラリの利用には開発期間の短縮や開発費用の削減が可能になるというメリットがある一方，ライブラリに含まれる不具合が取り込まれるというデメリットがある [1], [2]．ソフトウェアに含まれるセキュリティ上の欠陥は脆弱性と呼ばれ，脆弱性の発見時には脆弱性が含まれないバージョンへの更新対応などを迅速に行う必要がある．しかし，利用しているソフトウェアの管理は十分に行われておらず，ソフトウェア自身やソフトウェアが依存するライブラリの脆弱性への対応の遅れや対応漏れが問題となっている [3]．2021 年に深刻な脆弱性が発見された Java のログライブラリである Log4j では，2024 年 6 月時点においても，直近 1 か月間に日本から行われたダウンロードの 20% 程度が脆弱性を含むバージョンであることが確認されている [4]．

このようなライブラリを含むソフトウェアの管理に関する問題を解決するために，ソフトウェア部品表（Software Bill of Materials, 以降 SBOM）の利用が推奨されている [5]．SBOM は，特定のソフトウェアを構成するパッケージやドキュメントなどの要素，それらのライセンス，および要素間の関係が記述されているドキュメントである．開発者が SBOM を作成し，ソフトウェア本体とともに提供することで，ソフトウェアの利用者は SBOM に記述された情報に基づいて，ソフトウェアやそのソフトウェアが依存するライブラリに存在する既知の脆弱性やライセンス違反を確認できる．

SBOM は普及が十分に進んでいるとは言えず，ソフトウェアの開発者によって SBOM が提供されている例は少ない [6]．SBOM の普及を阻害する要因として，SBOM の生成ツールや，SBOM を用いたソフトウェアの管理を支援するツールが提供する機能の不足や不完全さが指摘されている [7]〜[9]．

SBOM に関連するツールを改善し，SBOM を普及させるためにはツールの評価に使用できる SBOM データセットがあることが望ましい．SBOM の元になるプロジェクトは多数公開されており，GitHub で公開されているオープンソースプロジェクトを用いることができる．一方，それらのプロジェクトから手作業で多数の SBOM を作成することは容易ではない．そこで，本研究では，SBOM データセットを作成するうえで既存の SBOM 生成ツールを利用する際，どのような課題があるかを調査した．調査では，ビルド済みのバイナリファイルを提供しているプロジェクトからの SBOM 作成，ビルド成果物が無いプロジェクトからの SBOM 作成，GitHub が提供する SBOM 生成機能について調査した．調査の結果，複数の SBOM 生成ツールを組みわせることで，SBOM データセットの要件を満たす SBOM を作成できることが分かった．

## 2. 背　　景

本章では，SBOM および SBOM の生成を支援するツールに関する課題を説明する．また，その解決を目的として構築を目指す SBOM のデータセットについて述べる．

### 2.1 SBOM

SBOM は，ソフトウェアを構成するライブラリやファイルについて，名前やバージョン，ライセンス，それらの間の依存関係などの情報を開発者が記述したドキュメントである．SBOM を利用して，ソフトウェアが依存するライブラリに脆弱性が発見されていないかや，使用しているライブラリのライセンス違反が発生していないかを確認できる．近年，ソフトウェアが依存するライブラリに含まれる脆弱性を利用した攻撃が増加していることから，適切なソフトウェアの管理のために SBOM を利用することが推奨されており，日本でも経済産業省によって企業による SBOM 導入の手引きが公開されている [10].

SBOM の主要なフォーマットとして，Software Package Data Exchange（SPDX）と CycloneDX がある．SPDX は Linux Foundation が策定しているフォーマットであり，バージョン 2.2.1 は ISO/IEC 5962:2021 として国際標準化されている．CycloneDX はソフトウェアのセキュリティ分野の研究やガイドラインの作成活動を行っている OWASP が策定しているフォーマットであり，SBOM のみでなく，機械学習モデルに関する BOM である ML-BOM や，IoT デバイスなどのハードウェアに関する BOM である HBOM なども記述できるように設計されている．両フォーマットともにソフトウェアに関連するより多くの情報を記述できるようにするために仕様の改定が継続的に行われている．バージョン 2 系列までの SPDX は SBOM の記述のみをサポートしていたが，2024 年 4 月に，CycloneDX と同様に ML-BOM なども記述できるように仕様が拡張されたバージョン 3.0.0 が公開された．また，CycloneDX も同月にバージョン 1.6 が公開され，仕様の継続的な改善が行われている．

### 2.2 SBOM 生成ツールと課題

SBOM はソフトウェアの構成要素や，依存関係にある要素から情報を収集して作成するため，手作業で作成するのは容易ではない．これらの負荷を軽減するため，様々な SBOM 生成ツールが存在している [10].

一方で，SBOM 生成ツールが生成する SBOM は正確でない可能性がある．Balliu ら [11] は Java で記述されたソフトウェアプロジェクトを対象に，CycloneDX 形式の SBOM を生成するツールの比較研究を行った．その結果，ツールによってソフトウェアからの情報収集方法が異なっており，ソフトウェアが依存するライブラリに対する完全な情報を含む SBOM を出力できるツールがないことを報告している．

SBOM 生成ツールが未成熟であることは，SBOM 普及にも影響を及ぼしている．Xia ら [8] は 65 名のソフトウェア開発者を対象にオンライン調査を行い，調査対象者の多くが既存の SBOM に関するツールの使いにくさや，相互運用性や標準化の欠如を課題として挙げていたことを報告している．また，Stalnaker ら [9] は，SBOM のステークホルダーが SBOM の作成・使用時に直面した課題を調査し，SBOM の生成や SBOM を用いたソフトウェアの管理を支援するツールの機能不足や不完全さが課題であると報告している．

### 2.3 SBOM のデータセット

SBOM 生成ツールの改善を行うためには，ツールを評価するためのベンチマークとなるデータセットがあることが望ましい．例として，コードクローン検出やソフトウェアテストの分野において，ツールやアルゴリズムの評価に用いることができるデータセットが公開されている [12], [13].

公開されている SBOM はあるものの，SBOM 生成ツールを評価することを意図したデータセットは存在していない．Linux Foundation [14] は 13 個のソフトウェアプロジェクトに対して作成した SPDX 形式のサンプルファイルを公開している．また，OWASP [15] は 7 個のソフトウェアプロジェクトに対して作成した CycloneDX 形式のサンプルファイルを公開している．Chainguard [16] は OSS の開発者により公開されている 53 個の SBOM と，人気のある 1,000 個の Docker イメージから複数の SBOM 生成ツールを用いて作成した 3,397 個の SBOM を公開している．

## 3. SBOM のデータセットが満たすべき要件

本章では，SBOM のデータセットが満たすべき要件を検討し，整理する．

SBOM に関連するツールの開発や検証のためのデータセットとして満たすべき要件を考えたとき，最も重要な記述は依存関係に関する部分である．依存関係は，SBOM を用いた脆弱性検査など多くの重要なタスクにも用いられる部分であり，そのため既存研究での SBOM 生成ツールの課題としても重点的に調査されている [11].

データセット内の SBOM に記載されているライブラリ間の依存関係が正確であることは必須である．また，SBOM の理念として，あるソフトウェアの実行に必要なライブラリだけでなく，ビルドやテスト時にのみ用いたライブラリなど，開発環境やデプロイ環境に用いられた依存関係も全て記載することが理想的とされている [11]．そのため，正確性に加え網羅性もデータセットに備わっているべき要素となる．

SBOM に記述可能なライブラリに関する情報としては，ライブラリの一意な識別子，名前，バージョン，ライセンス，ライブラリに含まれるファイルのチェックサムなどがある．一意な識別子が記述されていれば，その他の情報はパッケージ管理システムなどから取得可能であると考えられるが，SBOM に関連するツールの検証に利用されるデータセットでは，識別子以外の情報も正確に含む必要がある．

また，依存関係はソフトウェア毎に異なるため，これらの要件を満たすデータセットが十分な規模である必要がある．各言語のエコシステムによっても依存関係の情報取得の仕組みは異なるため，データセットには多様なプログラミング言語で開発されているソフトウェアの SBOM が含まれるべきである．

## 4. 調査目的と方法

このような要件が考えられる一方で，多くのソフトウェアプロジェクトの SBOM からなる大規模な SBOM データセットを構築するためには，SBOM の作成作業を可能な限り自動化する必要がある．そこで，既存の SBOM 生成ツールが生成した SBOM と要件とを照らし合わせ，SBOM データセット構築に向けた課題の調査を行う．

SBOM は，生成対象のソフトウェアに関する情報を，いつ，どのように収集するかによって，表 1 に示す 6 種類に分類される [17]．ソースファイルを解析して生成される Source タイプの SBOM や，ソフトウェアのバイナリなどの成果物を解析して生成される Analyzed タイプの SBOM の作成には，ソフトウェアのビルドが不要であるため，作成作業の自動化が容易であると予想される．そこで，本調査では，Source タイプや Analyzed タイプの SBOM を生成する SBOM 生成ツールを用いて GitHub 上の OSS を対象に SBOM を作成し，SBOM データセットの構築に向けた課題を調査する．

また，複数言語を対象とすることで，言語による現状の違いを明らかにする．具体的には，Java に対して Source タイプと Analyzed タイプの生成ツールそれぞれ 1 種類を，C#に対して Source タイプの生成ツール 2 種類を適用し，出力された SBOM を分析する．

### 4.1 Java を対象とする SBOM 生成ツールの適用可能性

GitHub 上で公開されている Java のソフトウェアプロジェクトを対象に，Source タイプの SBOM と Analyzed タイプの SBOM を作成した（図 1）．2 種類の SBOM を組み合わせることで，SBOM データセットを構成する SBOM として必要な情報を含む SBOM を作成できるかを調査する．

#### 4.1.1 SBOM の生成対象

この調査では，GitHub 上で公開されているリポジトリから，SBOM の生成対象のリポジトリを以下の条件で選択した．

（1） 使用されているプログラミング言語が Java
（2） 2023 年以降にコミットがある
（3） アーカイブされていない
（4） スター数が 1,000 以上

現在もアクティブに開発されているプロジェクトを生成対象とするために，2023 年以降にコミットが行われており，かつアーカイブされていないプロジェクトを対象とした．

これらの条件を満たすリポジトリを，GitHub が提供するリポジトリの検索 API を用いて得た．得られたリポジトリには，サンプルプログラムやドキュメントなどの通常のソフトウェアプロジェクト以外のリポジトリも存在する可能性があるが，その場合はバイナリとしての成果物が含まれていることは少ないと考えられるため，SBOM の生成対象からは除かれると考えられる．

#### 4.1.2 SBOM の作成

Analyzed タイプの SBOM は Syft [18] を用いて作成した．Syft は，Java のアーカイブファイルを入力としてサポートしており，生成対象のソフトウェアのアーカイブファイルを指定することで，SPDX 形式や CycloneDX 形式の SBOM を生成することができる．生成対象のリポジトリのビルド済みのバイナリをダウンロードし，Syft に入力として与えて SPDX 形式の SBOM を作成した．

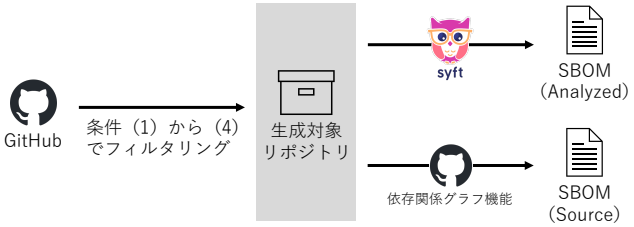


図 1 Java を対象とする SBOM の作成手順

一方，Source タイプの SBOM は GitHub の機能を用いて作成した．GitHub は，リポジトリ内のマニフェストファイルやロックファイルを静的に解析してプロジェクトが依存するライブラリの情報を取得し，プロジェクトの依存関係を確認可能にする依存関係グラフ機能を提供しており，プロジェクトの依存関係の情報は SPDX 形式の SBOM としてダウンロードできる．この機能によって得られる SBOM は，マニフェストファイルやロックファイルを静的に解析して生成されたものであるため，Source タイプの SBOM に該当する．

### 4.2 C#を対象とする SBOM 生成ツールの適用可能性

C#は Java に次いで多くのリポジトリが GitHub 上で公開されている言語であり，Java に似た構文を持つプログラミング言語である．C#を対象とした SBOM 生成ツールの調査は行われていないが，C#では Windows 環境でのみ利用可能なライブラリが存在するなど，Java とはソフトウェアエコシステム間の差異があるため，C#を対象とした SBOM 生成に特有の課題が存在する可能性がある．そこで，C#を対象とした SBOM 生成ツールを用いて SBOM を作成し，C#を対象とする SBOM 生成の課題を調査する．作成手順の概要を図 2 に示す．なお，C#のビルド済みバイナリから SBOM を生成する OSS のツールは著者の調査した限り存在しなかったため，Source タイプの SBOM 作成のみを調査する．

#### 4.2.1 SBOM の生成対象

この調査では，GitHub 上で公開されているリポジトリから，生成対象のリポジトリを以下の条件で選択した．

（1） 使用されているプログラミング言語が C#
（2） 2023 年以降にコミットがある
（3） アーカイブされていない
（4） 通常のソフトウェアプロジェクトである

現在もアクティブに開発されているプロジェクトを生成対象とするために，2023 年以降にコミットが行われており，かつアーカイブされていないプロジェクトを対象とした．また，C#に対する調査では，Analyzed タイプの SBOM を生成するツールを利用せず，サンプルプログラムやドキュメントなどバイナリとしての成果物が含まれないリポジトリが多く生成対象となる恐れがある．そこで，目視で通常のソフトウェアプロジェクトをフィルタリングする．

表 1　SBOM の分類

| SBOM の種類 | 定義 |
|---|---|
| Design | ソフトウェアの設計段階に作成され，利用が想定される要素の情報を含む SBOM. |
| Source | 開発環境，ソースファイル，ビルド時の依存情報から直接作成される SBOM. |
| Build | ソフトウェアのリリース可能な成果物をビルドする過程で生成される SBOM. |
| Analyzed | ソフトウェアの成果物を解析することで生成される SBOM. |
| Deployed | 設定ファイルの内容やデプロイ環境での動作の分析結果などに基づいて生成される SBOM. |
| Runtime | 実行中に実際に使用されているコンポーネントの情報を収集することで生成される SBOM. |

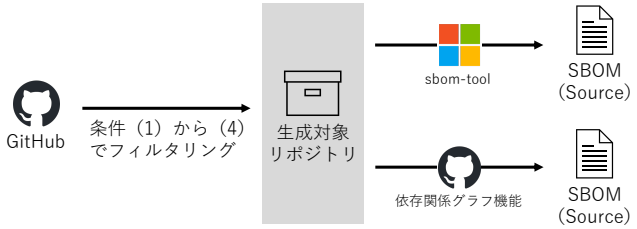


図 2　C#を対象とする SBOM の作成手順

条件（1）から（3）について，使用されているプログラミング言語，コミット時期，アーカイブについて GitHub の API を用いてフィルタリングを行った．その中から，スター数上位のリポジトリについて README ファイルや説明文を確認し，条件（4）に当てはまるものを 30 件選択した．

#### 4.2.2　SBOM の作成

SBOM の作成には，Microsoft が公開している sbom-tool [19] を用いた．sbom-tool は，生成対象のソフトウェアのソースコードが存在するディレクトリを入力として与えることで，ディレクトリ内に存在するソフトウェアのマニフェストファイルやロックファイルを分析し，SPDX 形式の SBOM を出力するツールである．また，Java を対象にした調査と同様に，GitHub の依存関係グラフ機能を用いて SBOM を作成した．

## 5.　調査結果と考察

本章では，SBOM データセットの構築における SBOM 生成ツールの適用可能性について調査した結果と考察を述べる．

### 5.1　Java を対象とする SBOM 生成ツールの適用可能性

ビルド済みのバイナリが提供されていたリポジトリの割合と，SBOM 生成ツールによって生成された SBOM の内容について説明し，Java を対象として SBOM 生成ツールをデータセットの構築に利用する際の課題について考察する．

#### 5.1.1　ビルド済みのバイナリが GitHub の Release 機能で提供されているリポジトリの割合

**4.1** で述べた条件（1）から（3）を満たすリポジトリは 3,198 リポジトリであり，そのうち 220 リポジトリ（6.88%）で GitHub の Release 機能を用いてビルド済みのバイナリが提供されていた（表 2）．jar ファイルなどの Java のアーカイブファイルが提供されていたものが 108 リポジトリ（3.38%），zip ファイルに圧縮されて Java のアーカイブファイルが提供されていたものが 121 リポジトリ（3.78%）であった．両方の形式で Java のアーカイブファイルが提供されていたリポジトリが 9 個存在したため，ビルド済みのバイナリが提供されていたリポジトリは合計 220 個である．

表 2　GitHub の Release 機能でバイナリが提供されていたリポジトリ

| ファイルの種類 | 件数 | 割合 |
|---|---|---|
| Java のアーカイブファイル | 108 | 3.38% |
| Java のアーカイブファイルを含む zip ファイル | 121 | 3.78% |
| 合計 | 220 | 6.88% |

#### 5.1.2　Syft で作成した SBOM

これらの Java のアーカイブファイルを対象として，SBOM 生成ツールの Syft を実行すると，すべてのファイルに対して SBOM の生成に成功した．生成された SBOM には，Java のアーカイブファイルに含まれる依存ライブラリの情報が含まれており，直接的または推移的に依存するライブラリの情報が含まれていた．一方で，単体テストのライブラリのように実行時に使用されないライブラリの情報は含まれなかった．

ライブラリに関する情報の記述例をコード 1 に示す．この例では caffeine というライブラリのバージョン 2.9.3 に関する情報が記述されている．5 行目から 8 行目には，ライブラリのファイルのチェックサム情報が記述されている．SPDX 形式では，異なるアルゴリズムを用いて計算された複数のチェックサムの値を記述することができるが，生成された SBOM には SHA1 で計算された値のみが記述されていた．9 行目にはライブラリのライセンス情報が，ライブラリのライセンスファイルへの URL を記述した要素への参照として記述されている．11 行目から 13 行目にはライブラリの識別子である purl の情報が記述されている．これらの情報は，ソースコードに記述された情報や，生成元のアーカイブファイルの中身を展開して確認できる情報と一致しており，Syft が依存するライブラリの情報を正確に取得できていることが確認できた．ただし，ライセンスの情報はライセンスファイルへの URL を用いて記述されており，SPDX で定められた SPDX-License-Identifier と呼ばれる正規化されたライセンス識別子は使用されていない．

#### 5.1.3　GitHub の依存関係グラフ機能で作成した SBOM

GitHub の依存関係グラフ機能で作成した SBOM には，直接的に依存する Java のライブラリなどの情報が含まれていた．一方で，推移的に依存するライブラリの情報は含まれなかった．

依存する Java のライブラリは，Maven の pom.xml ファイルに記述されたプロジェクトの依存関係から特定されている．pom.xml では，ライブラリのバージョンを省略したり範囲指定したりすることが可能なため，pom.xml の静的な解析では

コード 1　Syft で生成した SBOM でのライブラリ情報の例

```
{
  "name": "caffeine",
  "SPDXID": "SPDXRef-Package-java-archive-caffe...",
  "versionInfo": "2.9.3",
  "checksums": [{
    "algorithm": "SHA1",
    "checksumValue": "b162491..."
  }],
  "licenseDeclared": "LicenseRef-https---www.apache.org-licenses-LICENSE-2.0",
  "externalRefs": [{
     "referenceCategory": "PACKAGE-MANAGER",
     "referenceType": "purl",
     "referenceLocator": "pkg:maven/com.github.ben-manes.caffeine/caffeine@2.9.3"
  } ...], ...
}
```

コード 2　GitHub で生成した SBOM でのライブラリ情報の例

```
{
  "SPDXID": "SPDXRef-maven-com.github.ben-ma...",
  "name": "maven:com.github.ben-manes.caffe...",
  "versionInfo": "", ...
}
```

依存するライブラリのバージョンを特定できないことがある．そのため，生成された SBOM にはバージョン情報を含まないライブラリ情報が多く見られた．生成された SBOM に含まれるライブラリ情報の例をコード 2 に示す．この例では，コード 1 と同様に caffeine というライブラリに関する情報が記述されているが，バージョンの特定に失敗しているためバージョン情報は空文字列であり，ライセンスに関する記述も含まれていない．一方で，バージョンの特定に成功したライブラリの場合はバージョン情報が記述され，ライセンスの情報も SPDX-License-Identifier を用いて記述されていた．

#### 5.1.4　考　　察

GitHub の Release 機能を用いてビルド済みのバイナリが提供されている Java のリポジトリの割合は 7% 程度であった．スター数が比較的少ないリポジトリでもビルド済みのバイナリが提供されていたことから，リポジトリの選択条件であるスター数の下限をより小さくすることで，十分な数の SBOM 生成対象を得られると思われる．

Syft と GitHub の依存関係グラフ機能では生成される SBOM が含む情報に違いがあり，一方のツールのみでは SBOM データセットの要件を満たす SBOM は作成できない．SBOM データセットの要件を満たす SBOM を作成するためには，複数の SBOM 生成ツールを用いて作成した SBOM を組み合わせて，単一のツールでは不足する情報を補うことが有効であると考えられる．また，ライブラリのチェックサムについては，今回調査した両ツールともに情報が不足していた．SBOM 生成ツールのみでは補えない情報については，SBOM 生成ツール以外のツールも用いて情報を取得し，不足する情報を SBOM に追加することで対応が可能であると考えられる．

コード 3　sbom-tool で生成した SBOM でのライブラリ情報の例

```
{
  "name": "ReactiveUI.WPF",
  "SPDXID": "SPDXRef-Package-A0E054779263F...",
  "licenseDeclared": "NOASSERTION",
  "versionInfo": "19.6.1",
  "externalRefs": [{
     "referenceCategory": "PACKAGE-MANAGER",
     "referenceType": "purl",
     "referenceLocator": "pkg:nuget/ReactiveUI.WPF@19.6.1"
  }], ...
}
```

### 5.2　C#を対象とする SBOM 生成ツールの適用可能性

SBOM 生成ツールによって生成された SBOM の内容について説明し，C#を対象として SBOM 生成ツールをデータセットの構築に利用する際の課題について考察する．

#### 5.2.1　sbom-tool で作成した SBOM

sbom-tool を用いて，依存するライブラリの情報を含む SBOM を作成するためには，C#のパッケージ管理システムを用いて依存関係の復元処理を行う必要があった．依存関係の復元処理を行うことで，ビルド時に使用されるライブラリのバージョンが決定され，依存するライブラリのバージョン情報を正確に取得できるようになる．依存関係の復元処理は，生成対象として選択された 30 個のリポジトリのうち，19 個のリポジトリ（63%）で成功した．これら 19 個のリポジトリについて sbom-tool を実行すると，すべてのリポジトリで SBOM が生成された．

sbom-tool で作成した SBOM には，直接的または推移的に依存するライブラリの情報が含まれていた．生成された SBOM に含まれる依存ライブラリの情報の例をコード 3 に示す．この例では，ReactiveUI.WPF というライブラリのバージョン 19.6.1 に関する情報が記述されており，ライブラリのバージョン情報や識別子（7 行目から 11 行目）が含まれていることが確認できる．一方で，ライセンスに関する情報や，ライブラリのファイルのチェックサム情報は含まれていなかった．

#### 5.2.2　GitHub の依存関係グラフ機能で作成した SBOM

GitHub の依存関係グラフ機能で作成した SBOM には，直接的に依存するライブラリなどの情報が含まれていた．一方で，推移的に依存するライブラリの情報は含まれなかった．

生成された SBOM に含まれるライブラリ情報の例をコード 4 に示す．この例では，コード 3 と同じく ReactiveUI.WPF というライブラリに関する情報が記述されている．ライブラリのバージョン情報や識別子（5 行目から 9 行目）は含まれているが，ライセンスに関する情報や，ライブラリのファイルのチェックサム情報は含まれていない．C#においても Java（Maven）と同様に，依存するライブラリのバージョンの指定

コード 4　GitHub で生成した SBOM でのライブラリ情報の例

```
{
  "SPDXID": "SPDXRef-nuget-ReactiveUI.WPF-20.1.1",
  "name": "nuget:ReactiveUI.WPF",
  "versionInfo": "20.1.1",
  "externalRefs": [{
     "referenceCategory": "PACKAGE-MANAGER",
     "referenceLocator": "pkg:nuget/ReactiveUI.WPF
          @20.1.1",
     "referenceType": "purl"
  }], ...
}
```

を省略したり，範囲指定したりすることができるため，バージョン情報が含まれていないライブラリ情報が存在した．

#### 5.2.3 考　　察

sbom-tool は，ビルド時に使用されるライブラリのバージョン情報を正確に取得するために，依存関係の復元処理を行う必要があった．依存関係の復元処理に成功する割合は 63% であり，十分な数の SBOM 生成対象を確保する上で大きな問題にはならないと考えられる．

sbom-tool と GitHub の依存関係グラフ機能は，ともに Source タイプの SBOM を生成するが，生成された SBOM に含まれる情報には違いがあった．C#においても，複数の SBOM 生成ツールや，不足する情報を取得できるその他のツールを組み合わせることで SBOM データセットの要件を満たす SBOM が作成可能であると考えられる．

## 6. ま と め

本研究では，GitHub 上で公開されている OSS を対象に既存のツールを用いた SBOM の作成を行い，SBOM データセットの構築における課題を調査した．Java と C#のいずれの場合も，単一の生成ツールのみでは SBOM データセットの要件を満たす SBOM は作成できず，複数のツールを用いて不足する情報を補う必要があることが分かった．

ソフトウェアエコシステム毎に SBOM の作成における課題は異なる可能性があるため，他の言語に対しても同様の調査を行う必要がある．また，複数のツールから得た情報を組み合わせて要件を満たす SBOM を作成する方法の検討が必要である．

**謝辞**

本研究は, JSPS 科研費 JP24H00692, JP21K18302, JP21H04877, JP23K24823, JP22K11985, JP24K14895, JP21K02862, JP23K28065, 2024 年度南山大学パッヘ研究奨励金 I-A-2 の助成を得て行われた．

## 文　　献

[1] O.P.N. Slyngstad, A. Gupta, R. Conradi, P. Mohagheghi, H. Rønneberg, and E. Landre, “An Empirical Study of Developers Views on Software Reuse in Statoil ASA,” Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE), pp.242–251, 2006.

[2] Y. Wang, B. Chen, K. Huang, B. Shi, C. Xu, X. Peng, Y. Wu, and Y. Liu, “An Empirical Study of Usages, Updates and Risks of Third-Party Libraries in Java Projects,” Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp.35–45, 2020.

[3] M. Alfadel, D.E. Costa, and E. Shihab, “Empirical Analysis of Security Vulnerabilities in Python Packages,” Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp.446–457, 2021.

[4] Sonatype, “Log4j Updates and Vulnerability Resources”. 2024 年 6 月 25 日に閲覧. https://www.sonatype.com/resources/log4j-vulnerability-resource-center

[5] NTIA Multistakeholder Process on Software Component Transparency Framing Working Group, “Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM),” https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf, 2019.

[6] S. Nocera, S. Romano, M.D. Penta, R. Francese, and G. Scanniello, “Software Bill of Materials Adoption: A Mining Study from GitHub,” Proceedings of the 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp.39–49, 2023.

[7] The Linux Foundation, “The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness,” 2022. 2024 年 6 月 25 日に閲覧. https://www.linuxfoundation.org/research/the-state-of-software-bill-of-materials-sbom-and-cybersecurity-readiness

[8] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu, “An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead,” Proceedings of the IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp.2634–2646, 2023.

[9] T. Stalnaker, N. Wintersgill, O. Chaparro, M. Di Penta, D.M. German, and D. Poshyvanyk, “Boms away! inside the minds of stakeholders: A comprehensive study of bills of materials for software systems,” Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE), pp.513–525, 2024. https://doi.org/10.1145/3597503.3623347

[10] 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,” https://www.meti.go.jp/press/2023/07/20230728004/20230728004.html，2023.

[11] M. Balliu, B. Baudry, S. Bobadilla, M. Ekstedt, M. Monperrus, J. Ron, A. Sharma, G. Skoglund, C. Soto-Valero, and M. Wittlinger, “Challenges of Producing Software Bill of Materials for Java,” IEEE Security & Privacy, pp.2–13, 2023.

[12] F. Al-Omari, C.K. Roy, and T. Chen, “Semanticclonebench: A semantic code clone benchmark using crowd-source knowledge,” 2020 IEEE 14th International Workshop on Software Clones (IWSC), pp.57–63, 2020.

[13] 肥後芳樹，“自動テスト生成技術を利用した機能等価メソッドデータセットの構築,” ソフトウェアエンジニアリングシンポジウム 2023 論文集，vol.2023，pp.30–38，08 2023. https://cir.nii.ac.jp/crid/1050018218946104192

[14] SPDX Workgroup, “spdx-examples,” https://github.com/spdx/spdx-examples.

[15] CycloneDX, “bom-examples,” https://github.com/CycloneDX/bom-examples.

[16] Chainguard, “bom-shelter,” https://github.com/chainguard-dev/bom-shelter, 2022.

[17] CISA, “Types of software bill of materials (sbom) documents,” https://www.cisa.gov/sites/default/files/2023-04/sbom-types-document-508c.pdf, 2023.

[18] anchore, “syft,” https://github.com/anchore/syft, 2020.

[19] Microsoft, “sbom-tool”. 2024 年 6 月 25 日に閲覧. https://github.com/microsoft/sbom-tool