

## 【서지사항】

【서류명】	특허출원서
【참조번호】	W25P0293KR
【출원구분】	특허출원
【출원인】	
【명칭】	고려대학교 산학협력단
【특허고객번호】	2-2004-017068-0
【대리인】	
【명칭】	특허법인 위솔
【대리인번호】	9-2022-100001-9
【지정된변리사】	나강은, 김경용, 강현모, 편진호, 이영규
【발명의 국문명칭】	재사용 오픈소스 소프트웨어의 컴포넌트에 속한 버전 분포 를 포괄하는 적응형 버전 생성 방법 및 장치
【발명의 영문명칭】	METHOD AND APPARATUS FOR GENERATING ADAPTIVE VERSION COVERING VERSION DISTRIBUTION OF REUSED OPEN SOURCE SOFTWARE COMPONENT
【발명자】	
【성명】	우승훈
【성명의 영문표기】	Woo, Seung-Hoon
【국적】	KR
【주민등록번호】	
【우편번호】	
【주소】	

**【거주국】** KR

**【발명자】**

**【성명】** 최영재

**【성명의 영문표기】** Choi Young-Jae

**【국적】** KR

**【주민등록번호】**

**【우편번호】**

**【주소】**

**【거주국】** KR

**【출원언어】** 국어

**【심사청구】** 청구

**【공지예외적용대상증명서류의 내용】**

**【공개형태】** 학회 및 논문 (2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE))  
<https://ieeexplore.ieee.org/document/11029893/metrics#full-text-header>)

**【공개일자】** 2025.04.26

**【이 발명을 지원한 국가연구개발사업】**

**【과제고유번호】** 2710081298

**【과제번호】** 00517788

**【부처명】** 과학기술정보통신부

**【과제관리(전문)기관명】** 한국연구재단

<b>【연구사업명】</b>	개인기초연구(과기정통부)(R&D)
<b>【연구과제명】</b>	멀티레벨 코드 분석을 통한 지능형 SBOM 생성 및 취약점 분석 자동화 연구
<b>【과제수행기관명】</b>	고려대학교
<b>【연구기간】</b>	2025.03.01 ~ 2026.02.28

**【이 발명을 지원한 국가연구개발사업】**

<b>【과제고유번호】</b>	2710069045
<b>【과제번호】</b>	00440780
<b>【부처명】</b>	과학기술정보통신부
<b>【과제관리(전문)기관명】</b>	정보통신기획평가원
<b>【연구사업명】</b>	정보보호핵심원천기술개발(R&D, 정보화)
<b>【연구과제명】</b>	SW공급망 전주기 보안 내재화를 위한 SBOM과 VEX 연계 기반 취약점 통합 관리 플랫폼 기술
<b>【과제수행기관명】</b>	고려대학교산학협력단
<b>【연구기간】</b>	2025.01.01 ~ 2025.12.31

**【이 발명을 지원한 국가연구개발사업】**

<b>【과제고유번호】</b>	2710068997
<b>【과제번호】</b>	II220277
<b>【부처명】</b>	과학기술정보통신부
<b>【과제관리(전문)기관명】</b>	정보통신기획평가원
<b>【연구사업명】</b>	정보보호핵심원천기술개발(R&D, 정보화)

# 【연구과제명】 SW공급망 보안을 위한 SBOM 자동생성 및 무결성 검증기술 개발

【과제수행기관명】 고려대학교 산학협력단

**【연구기간】** 2025.01.01 ~ 2025.12.31

【이 발명을 지원한 국가연구개발사업】

【관제고유번호】 2370000334

【관제번호】 00345025

【부처명】 문화체육관광부

【과제관리(전문)기관명】 한국콘텐츠진흥원

글로벌 저작권 협약 신속 대응(R&D)

# 【연구과제명】 생성형 AI 저작권 관리 및 보호 기술 개발을 위한 국제공동 연구 및 글로벌 인재 양성

## 【과제수행기관명】 고려대학교 사학 협력단

【연구기간】 2025.01.01 ~ 2025.12.31

**【최지】** 위와 같이 틀현청장에게 제출합니다.

대리인 허락 법인 위 솔 (서명 또는 인)

## 【수수료】

### 【출원료】

[가산출원료]

## 【우선권 주장료】

【심사청구료】

【합계】

【감면사유】

【감면후 수수료】

【첨부서류】

1 : 공지예외적용대상(신규성상실의예외,\_출원시의특례)규정을\_적용받기\_위한\_증명서류

[PDF 파일 첨부](#)

2 : 기타첨부서류

[PDF 파일 첨부](#)

## 【발명의 설명】

### 【발명의 명칭】

재사용 오픈소스 소프트웨어의 컴포넌트에 속한 버전 분포를 포괄하는 적응형 버전 생성 방법 및 장치{METHOD AND APPARATUS FOR GENERATING ADAPTIVE VERSION COVERING VERSION DISTRIBUTION OF REUSED OPEN SOURCE SOFTWARE COMPONENT}

### 【기술분야】

**【0001】** 본 발명은 오픈소스 소프트웨어 관리 기술에 관한 것으로, 보다 구체적으로는 재사용된 오픈소스 소프트웨어 컴포넌트의 함수 단위 버전 분포를 분석하여 적응형 버전 정보를 생성하고, 이를 SBOM(Software Bill of Materials)에 반영함으로써 오픈소스 소프트웨어의 재사용 현황을 정밀하게 관리할 수 있도록 하는 기술에 관한 것이다.

### 【발명의 배경이 되는 기술】

**【0002】** 소프트웨어 개발 과정에서 오픈소스 소프트웨어 컴포넌트는 재사용성이 높고 개발 비용을 절감할 수 있는 장점으로 인해 널리 활용되고 있다. 이러한 재사용 환경에서는 소프트웨어 공급망의 투명성을 확보하기 위하여 SBOM(Software Bill of Materials)이 중요한 역할을 담당한다. SBOM은 소프트웨어에 포함된 구성 요소와 그 버전 정보를 기록하여, 보안 취약점 관리와 라이선스 검증 등에 활용되는 핵심 도구이다.

【0003】 현재 사용되고 있는 소프트웨어 구성요소 분석 기술과 SBOM 표준들은 대부분 단일 버전을 하나의 OSS 컴포넌트에 매핑하는 방식을 채택하고 있다. 예컨대 CycloneDX와 같은 표준은 각 컴포넌트에 대해 하나의 대표 버전만을 기록하도록 정의되어 있으며, 기존 연구들 또한 컴포넌트 내부에서 가장 지배적인 버전을 선택하여 전체를 대표시키는 접근을 사용하고 있다. 이러한 방식은 구현이 단순하다는 장점이 있어 여러 SCA 솔루션과 SBOM 생성 툴에서 채택되어 왔다.

【0004】 그러나 오픈소스 소프트웨어 컴포넌트 내부에는 서로 다른 시점에서 재사용된 다양한 버전이 공존하는 경우가 빈번히 발생한다. 단일 버전만을 기록하는 기존 방식은 이러한 현실을 반영하지 못하므로, 다수의 버전을 포함하는 복합적 상황에서는 취약점 탐지 과정에서 오탐 및 미탐이 다수 발생한다. 또한, 여러 오픈 소스 소프트웨어 프로젝트에 공통적으로 포함되는 알고리즘 코드와 같은 노이즈 요소가 존재하여 정확한 버전 추적을 방해하며, 동일 오픈소스 소프트웨어의 서로 다른 버전이 중복 재사용된 경우를 구분하기 어렵다는 한계가 있다. 결과적으로 기존 소프트웨어 구성요소 분석 기술과 기존의 SBOM 표준은 공급망 보안 강화라는 본래 목적을 충분히 달성하지 못하는 문제가 있다.

【0005】 따라서 오픈소스 소프트웨어 컴포넌트의 내부 버전 다양성을 정확히 식별하고, 중복된 재사용이나 노이즈 코드로 인한 오류를 줄이면서 신뢰성 있는 SBOM을 생성할 수 있는 기술이 필요하다. 이러한 개선은 취약점 관리의 정밀도를 높이고, 소프트웨어 공급망 보안을 실질적으로 강화하는 기반이 될 수 있다.

## 【선행기술문헌】

### 【특허문헌】

【0006】(특허문헌 0001) 대한민국 등록특허공보 제10-2006242 호

### 【발명의 내용】

#### 【해결하고자 하는 과제】

【0007】본 발명이 해결하고자 하는 과제는 오픈소스 소프트웨어 컴포넌트 내에 존재하는 다양한 버전을 정밀하게 식별하고 이를 기반으로 신뢰성 있는 SBOM 을 생성할 수 있는 방법을 제공하는 것이다. 이를 위해 함수 단위의 세밀한 버전 구분을 가능하게 하고, 코드 클러스터링 기법을 통해 중복된 재사용과 노이즈 요소를 효과적으로 배제함으로써, 각 오픈소스 소프트웨어 컴포넌트의 실제 버전 분포를 적응형으로 표현할 수 있도록 한다. 이러한 접근은 단순히 대표 버전을 기록하는 방식이 아니라, 동일 컴포넌트 내에서 발견되는 버전의 다양성과 중복성을 정확히 반영하여 실질적인 취약점 관리와 공급망 보안을 지원하는 것을 목표로 한다.

【0008】또한, 본 발명은 재사용된 코드가 어떤 범위의 버전에서 기원했는지를 체계적으로 표현할 수 있도록 함으로써, 개발자나 보안 관리자가 전파된 취약점을 보다 신속하고 정확하게 파악할 수 있도록 돋는다. 결과적으로, 소프트웨어 공급망 전반에서 오픈소스 소프트웨어 관리의 효율성을 높이고, 보안 위협에 대한 대응 능력을 강화하는 것이 본 발명의 핵심 과제이다.

【0009】 한편, 본 발명의 기술적 과제들은 이상에서 언급한 기술적 과제들로 제한되지 않으며, 언급되지 않은 또 다른 기술적 과제들은 아래의 기재로부터 통상 의 기술자에게 명확하게 이해될 수 있을 것이다.

### 【과제의 해결 수단】

【0010】 일 실시예에 따른 프로세서에 의해 동작하는 적응형 버전 생성 장치 가 수행하는 방법에 있어서, 분석 대상 소프트웨어의 소스코드를 획득하는 동작; 상기 소스코드를 기초로 상기 소프트웨어가 포함하는 재사용된 오픈소스 소프트웨어 컴포넌트, 상기 컴포넌트가 위치하는 디렉토리 및 상기 디렉토리에 포함된 재사용 파일을 포함하는 디렉토리 계층 구조를 생성하는 동작; 각 컴포넌트에 대한 원본 오픈소스 소프트웨어의 실제 중복 파일 개수 보다 상기 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 많은 경우, 상기 디렉토리 계층 구조에 포함된 각 대상 중복 파일의 최소 공통 조상 노드를 기준으로 분기되는 디렉토리 경로를 기준으로 클러스터를 생성하는 동작; 및 각 클러스터에 포함된 함수의 오픈소스 소프트웨어 버전을 기초로, 각 클러스터 내에서 공존하는 복수의 버전 분포를 포괄적으로 표현하는 적응형 버전 정보를 생성하고, 상기 적응형 버전 정보를 각 클러스터에 매핑하여 상기 소프트웨어의 SBOM를 생성하는 동작을 포함할 수 있다.

【0011】 또한, 상기 디렉토리 계층 구조를 생성하는 동작은 상기 소스코드에 포함된 재사용된 오픈소스 소프트웨어 컴포넌트의 목록을 식별하는 동작; 각 컴포넌트 별로 재사용된 파일 리스트 및 상기 파일의 위치 경로를 추출하는 동작; 및

상기 파일 리스트 및 위치 경로를 기초로 루트 디렉토리, 상위 디렉토리 및 하위 디렉토리를 포함하는 트리 형태의 디렉토리 계층 구조를 구성하는 동작을 포함할 수 있다.

【0012】 또한, 상기 클러스터를 생성하는 동작은 각 클러스터에 포함된 함수의 개수가 원본 오픈소스 소프트웨어의 전체 함수 개수에서 차지하는 비율을 산출하고, 상기 비율이 미리 설정된 임계값 미만인 클러스터를 노이즈로 판단하여 제거하는 동작을 포함할 수 있다.

【0013】 또한, 상기 클러스터를 생성하는 동작은 상기 실제 중복 파일 개수 보다 상기 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 적거나 같은 경우, 상기 대상 중복 파일의 최소 공통 조상 노드를 기준으로 단일 클러스터를 생성하는 동작을 포함할 수 있다.

【0014】 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 모든 함수가 동일한 제1 버전에 속하는 경우, 상기 적응형 버전 정보를 상기 제1 버전으로 생성하는 동작을 포함할 수 있다.

【0015】 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 모든 함수가 동일한 제1 버전에 속하나, 일부 함수의 버전이 비유효한 버전에 해당하는 경우, 상기 적응형 버전 정보를 상기 제1 버전에 '+' 기호를 부가하여 생성하는 동작을 포함할 수 있다.

【0016】 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 함수들이 서로 다른 복수의 major 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 버전 기준에 '\*' 기호를 부가하여 생성하는 동작을 포함할 수 있다.

【0017】 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 함수들이 동일한 major 버전에 속하나 서로 다른 minor 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 minor 버전 기준에 '^' 기호를 부가하여 생성하는 동작을 포함할 수 있다.

【0018】 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 함수들이 동일한 major 버전과 동일한 minor 버전에 속하나 서로 다른 patch 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 patch 버전 기준에 '~' 기호를 부가하여 생성하는 동작을 포함할 수 있다.

【0019】 일 실시예에 따른 적응형 버전 생성 장치는 명령어를 포함하는 메모리; 및 상기 명령어를 기초로 소정의 동작을 수행하는 프로세서를 포함하고, 상기 프로세서의 동작은 분석 대상 소프트웨어의 소스코드를 획득하는 동작; 상기 소스코드를 기초로 상기 소프트웨어가 포함하는 재사용된 오픈소스 소프트웨어 컴포넌트, 상기 컴포넌트가 위치하는 디렉토리 및 상기 디렉토리에 포함된 재사용 파일을 포함하는 디렉토리 계층 구조를 생성하는 동작; 각 컴포넌트에 대한 원본 오픈소스 소프트웨어의 실제 중복 파일 개수 보다 상기 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 많은 경우, 상기 디렉토리 계층 구조에 포함된 각 대상 중복

파일의 최소 공통 조상 노드를 기준으로 분기되는 디렉토리 경로를 기준으로 클러스터를 생성하는 동작; 및 각 클러스터에 포함된 함수의 오픈소스 소프트웨어 버전을 기초로, 각 클러스터 내에서 공존하는 복수의 버전 분포를 포괄적으로 표현하는 적응형 버전 정보를 생성하고, 상기 적응형 버전 정보를 각 클러스터에 매핑하여 상기 소프트웨어의 SBOM를 생성하는 동작을 포함할 수 있다.

【0020】 또한, 상기 디렉토리 계층 구조를 생성하는 동작은 상기 소스코드에 포함된 재사용된 오픈소스 소프트웨어 컴포넌트의 목록을 식별하는 동작; 각 컴포넌트 별로 재사용된 파일 리스트 및 상기 파일의 위치 경로를 추출하는 동작; 및 상기 파일 리스트 및 위치 경로를 기초로 루트 디렉토리, 상위 디렉토리 및 하위 디렉토리를 포함하는 트리 형태의 디렉토리 계층 구조를 구성하는 동작을 포함할 수 있다.

【0021】 또한, 상기 클러스터를 생성하는 동작은 각 클러스터에 포함된 함수의 개수가 원본 오픈소스 소프트웨어의 전체 함수 개수에서 차지하는 비율을 산출하고, 상기 비율이 미리 설정된 임계값 미만인 클러스터를 노이즈로 판단하여 제거하는 동작을 포함할 수 있다.

【0022】 또한, 상기 클러스터를 생성하는 동작은 상기 실제 중복 파일 개수보다 상기 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 적거나 같은 경우, 상기 대상 중복 파일의 최소 공통 조상 노드를 기준으로 단일 클러스터를 생성하는 동작을 포함할 수 있다.

【0023】 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에

포함된 모든 함수가 동일한 제1 버전에 속하는 경우, 상기 적응형 버전 정보를 상기 제1 버전으로 생성하는 동작을 포함할 수 있다.

**【0024】** 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 모든 함수가 동일한 제1 버전에 속하나, 일부 함수의 버전이 비유효한 버전에 해당하는 경우, 상기 적응형 버전 정보를 상기 제1 버전에 '+' 기호를 부가하여 생성하는 동작을 포함할 수 있다.

**【0025】** 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 함수들이 서로 다른 복수의 major 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 버전 기준에 '\*' 기호를 부가하여 생성하는 동작을 포함할 수 있다.

**【0026】** 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 함수들이 동일한 major 버전에 속하나 서로 다른 minor 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 minor 버전 기준에 '^' 기호를 부가하여 생성하는 동작을 포함할 수 있다.

**【0027】** 또한, 상기 소프트웨어의 SBOM를 생성하는 동작은 제1 클러스터에 포함된 함수들이 동일한 major 버전과 동일한 minor 버전에 속하나 서로 다른 patch 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 patch 버전 기준에 '~' 기호를 부가하여 생성하는 동작을 포함할 수 있다.

### 【발명의 효과】

【0028】 본 발명에 따르면 오픈소스 소프트웨어 컴포넌트 내부의 다양한 버전을 정밀하게 식별할 수 있으므로, 기존 방식에서 발생하던 불필요한 오탐을 크게 줄이고 발견되지 못했던 취약점까지 탐지할 수 있는 효과가 있다. 특히, 함수 수준의 세밀한 버전 식별과 코드 클러스터링을 통한 노이즈 제거 및 중복 구분은 각 컴포넌트의 실제 버전 분포를 보다 정확히 반영할 수 있도록 하여, SBOM의 신뢰성을 현저히 향상시킨다. 이와 같은 개선은 단순한 컴포넌트 관리 차원을 넘어, 공급망 전반의 보안 체계를 강화하는 실질적 기반이 된다.

【0029】 더하여, 본 발명은 적응형 버전 개념을 도입함에 따라, 재사용된 코드가 포함할 수 있는 버전의 범위를 명확히 정의할 수 있어 개발자와 보안 담당자가 효율적으로 대응할 수 있도록 지원한다. 이를 통해 SBOM 솔루션과 취약점 관리 시스템의 활용성이 확대되고, 대규모 소프트웨어 생태계에서 오픈소스 소프트웨어 재사용에 따른 위험을 정량적으로 관리할 수 있는 새로운 패러다임을 제시한다.

【0030】 한편, 본 발명에 의한 효과는 이상에서 언급한 것들로 제한되지 않으며, 언급되지 않은 또 다른 기술적 효과들은 아래의 기재로부터 통상의 기술자에게 명확하게 이해될 수 있을 것이다.

### 【도면의 간단한 설명】

【0031】 도 1은 일 실시예에 따른 적응형 버전 생성 장치의 구성도이다. 도 2는 일 실시예에 따른 적응형 버전 생성 장치가 수행하는 동작의 단계를 나타낸 흐름도이다.

도 3은 일 실시예에 따른 디렉토리 계층 구조의 예시도이다.

도 4는 일 실시예에 따른 디렉토리 계층 구조의 클러스터링 과정에 대한 예시도이다.

도 5는 일 실시예에 따른 클러스터별 함수 비율을 나타낸 예시도이다.

도 6은 일 실시예에 따른 적응형 버전 정보를 생성하는 동작의 예시도이다.

도 7은 기존 기법을 통해 생성된 SBOM의 예시도이다.

도 8은 본 발명의 기법을 통해 생성된 SBOM의 예시도이다.

도 9는 본 발명의 기법과 기존 기법을 비교한 실험 결과를 나타낸 표이다.

도 10은 본 발명의 기법의 실행 시간을 분석한 결과를 나타낸 그래프이다.

### **【발명을 실시하기 위한 구체적인 내용】**

**【0032】** 본 발명의 목적과 기술적 구성 및 그에 따른 작용 효과에 관한 자세한 사항은 본 발명의 명세서에 첨부된 도면에 의거한 이하의 상세한 설명에 의해 보다 명확하게 이해될 것이다. 첨부된 도면을 참조하여 본 발명에 따른 실시예를 상세하게 설명한다.

**【0033】** 본 명세서에서 개시되는 실시예들은 본 발명의 범위를 한정하는 것으로 해석되거나 이용되지 않아야 할 것이다. 이 분야의 통상의 기술자에게 본 명세서의 실시예를 포함한 설명은 다양한 응용을 갖는다는 것이 당연하다. 따라서, 본 발명의 상세한 설명에 기재된 임의의 실시예들은 본 발명을 보다 잘 설명하기 위한 예시적인 것이며 본 발명의 범위가 실시예들로 한정되는 것을 의도하지 않는

다.

**【0034】** 도면에 표시되고 아래에 설명되는 기능 블록들은 가능한 구현의 예들일 뿐이다. 다른 구현들에서는 상세한 설명의 사상 및 범위를 벗어나지 않는 범위에서 다른 기능 블록들이 사용될 수 있다. 또한, 본 발명의 하나 이상의 기능 블록이 개별 블록들로 표시되지만, 본 발명의 기능 블록들 중 하나 이상은 동일 기능을 실행하는 다양한 하드웨어 및 소프트웨어 구성들의 조합일 수 있다.

**【0035】** 또한, 어떤 구성요소들을 포함한다는 표현은 “개방형”의 표현으로서 해당 구성요소들이 존재하는 것을 단순히 지칭할 뿐이며, 추가적인 구성요소들을 배제하는 것으로 이해되어서는 안 된다.

**【0036】** 나아가 어떤 구성요소가 다른 구성요소에 “연결되어” 있다거나 “접속되어” 있다고 언급될 때에는, 그 다른 구성요소에 직접적으로 연결 또는 접속되어 있을 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 한다.

**【0037】** 이하, 본 발명의 다양한 실시예가 첨부된 도면을 참조하여 기재된다. 그러나, 이는 본 발명을 특정한 실시 형태에 대해 한정하려는 것이 아니며, 본 발명의 실시예의 다양한 변경(modification), 균등물(equivalent), 및/또는 대체물(alternative)을 포함하는 것으로 이해되어야 한다.

**【0038】** 본 발명은 오픈소스 소프트웨어 관리 기술에 관한 것으로, 보다 구체적으로는 재사용된 오픈소스 소프트웨어 컴포넌트의 함수 단위 버전 분포를 분석

하여 적응형 버전 정보를 생성하고, 이를 SBOM(Software Bill of Materials)에 반영함으로써 오픈소스 소프트웨어의 재사용 현황을 정밀하게 관리할 수 있도록 하는 기술을 구현하는 적응형 버전 생성 장치(100)를 제안한다.

【0039】 이하, 본 발명의 적응형 버전 생성 장치(100)에 대한 구성과, 각 구성의 동작을 살펴보도록 한다.

【0040】 도 1은 일 실시예에 따른 적응형 버전 생성 장치(100)(이하, '장치(100)'로 지칭)의 구성도이다.

【0041】 도 1을 참조하면, 일 실시예에 따른 장치(100)는 각각 메모리(110), 프로세서(120), 입출력 인터페이스(130) 및 통신 인터페이스(140)를 포함할 수 있다.

【0042】 메모리(110)는 외부 장치로부터 획득한 데이터 또는 스스로 생성한 데이터를 저장할 수 있다. 메모리(110)는 프로세서(120)의 동작을 수행시킬 수 있는 명령어들을 저장할 수 있다. 예를 들어, 메모리(110)는 분석 대상 소프트웨어, 소스코드, 디렉토리 계층 구조, 오픈소스 소프트웨어에 존재하는 실제 중복 파일 개수의 정보를 저장하도록 사전에 구축된 데이터베이스 등을 저장할 수 있다.

【0043】 프로세서(120)는 전반적인 동작을 제어하는 연산 장치이다. 프로세서(120)는 메모리(110)에 저장된 명령어들을 실행할 수 있다. 본 문서의 실시예에 따른 장치(100)의 동작은 프로세서(120)에 의해 수행되는 동작으로 이해될 수 있다.

【0044】 입출력 인터페이스(130)는 정보를 입력하거나 출력하는 하드웨어 인터페이스 또는 소프트웨어 인터페이스를 포함할 수 있다.

【0045】 통신 인터페이스(140)는 통신망을 통해 정보를 송수신 할 수 있게 한다. 이를 위해, 통신 인터페이스(140)는 무선 통신모듈 또는 유선 통신모듈을 포함할 수 있다.

【0046】 장치(100)는 프로세서(120)를 통해 연산을 수행하고 네트워크를 통해 정보를 송수신할 수 있는 다양한 형태의 장치로 구현될 수 있다. 예를 들면, 서버, 컴퓨터 장치, 휴대용 통신 장치, 스마트 폰, 휴대용 멀티미디어 장치, 노트북, 태블릿 PC 등의 형태로 구현될 수 있으나, 이러한 예시에 한정되는 것은 아니다.

【0047】 도 2는 일 실시예에 따른 장치(100)가 수행하는 동작의 단계를 나타낸 흐름도이다. 도 2의 실시예에 따른 장치(100)의 동작은 프로세서(120)에 의해 수행되는 동작으로 이해될 수 있다.

【0048】 한편, 도 2에 개시된 각 단계는 본 발명의 목적을 달성함에 있어서 바람직한 실시예일 뿐이며, 필요에 따라 일부 단계가 추가 또는 삭제될 수 있음은 물론이고, 어느 한 단계가 다른 단계에 포함되어 수행될 수도 있다. 도 2에 개시된 각 동작의 순서는 이해의 편의를 위해 배치된 순서일 뿐, 이러한 순서가 시계열적인 순서로 한정되는 것이 아니며, 설계자의 선택에 따라 순서가 다르게 변경되어 동작될 수 있다.

【0049】 도 2를 참조하면, S1010 단계에서, 장치(100)는 분석 대상 소프트웨어의 소스코드를 획득할 수 있다.

【0050】 분석 대상 소프트웨어는 오픈소스 소프트웨어 컴포넌트가 재사용된 형태로 포함된 응용 소프트웨어 또는 시스템 소프트웨어를 의미한다. 이러한 분석 대상 소프트웨어로부터 추출된 소스코드는 이후 재사용된 오픈소스 소프트웨어 컴포넌트의 식별과 디렉토리 계층 구조의 생성을 위한 기초 데이터로 활용된다.

【0051】 한편, 본 발명은 이러한 분석 대상 소프트웨어의 소스코드를 대상으로, 그 내부에 재사용된 오픈소스 소프트웨어 컴포넌트를 함수 단위까지 정밀하게 식별하고, 식별된 결과를 기초로 다양한 버전 분포를 포괄하는 적응형 버전을 생성하는 것을 목표로 한다. 이를 통해 기존의 단일 버전 기반 접근 방식에서 발생하는 한계를 극복하고, 보다 신뢰성 있는 SBOM(Software Bill of Materials)를 생성하고자 한다.

【0052】 S1020 단계에서, 장치(100)는 소스코드를 기초로 소프트웨어가 포함하는 재사용된 오픈소스 소프트웨어 컴포넌트, 컴포넌트가 위치하는 디렉토리 및 디렉토리에 포함된 재사용 파일을 포함하는 디렉토리 계층 구조를 생성할 수 있다.

【0053】 여기서 컴포넌트란, 분석 대상 소프트웨어 내부에 포함된 특정 오픈 소스 소프트웨어 단위를 의미하며, 해당 오픈소스 소프트웨어의 이름과 버전을 기준으로 구분될 수 있다. 디렉토리란, 컴포넌트가 배치된 소스코드 경로상의 계층적 구조를 의미하며, 상위 디렉토리와 하위 디렉토리의 포함 관계를 통해 파일의 위치

와 종속성을 파악할 수 있다. 재사용 파일이란, 컴포넌트로부터 실제로 복사되거나 변형되어 분석 대상 소프트웨어 내에서 활용되는 소스코드 파일을 의미하며, 원본 오픈소스 소프트웨어의 동일한 파일명 또는 유사한 파일명을 유지할 수 있다.

**【0054】** 도 3은 일 실시예에 따른 디렉토리 계층 구조의 예시도이다.

**【0055】** 도 3을 참조하면, 장치(100)는 소스코드를 기초로 재사용된 오픈소스 소프트웨어 컴포넌트의 목록을 식별할 수 있다. 또한 각 컴포넌트별로 재사용된 파일 리스트와 파일의 위치 경로를 추출할 수 있으며, 추출된 정보에 기초하여 루트 디렉토리, 상위 디렉토리 및 하위 디렉토리를 포함하는 트리 형태의 디렉토리 계층 구조를 구성할 수 있다. 이와 같이 구성된 디렉토리 계층 구조는 소프트웨어 내에서 재사용된 오픈소스 소프트웨어의 포함 관계와 파일 배치를 직관적으로 표현 할 수 있다.

**【0056】** 예를 들어, 도 3에서 Filament 소프트웨어는 루트 오픈소스 소프트웨어 컴포넌트로 표현되며, 그 하위에 third\_party 디렉토리와 libs 디렉토리가 존재한다. third\_party 디렉토리에는 libgtest 디렉토리와 오픈소스 소프트웨어 컴포넌트 Libassimp가 포함된다. libgtest 디렉토리 하위에는 오픈소스 소프트웨어 컴포넌트 googletest가 존재하며, 이 컴포넌트에는 재사용된 파일 gtest.cc가 포함된다. 또한 오픈소스 소프트웨어 컴포넌트 Libassimp에는 contrib 디렉토리가 존재하고, 그 내부에 위치한 오픈소스 소프트웨어 컴포넌트 gtest에는 역시 gtest.cc 파일이 포함된다. 한편, libs 디렉토리에는 오픈소스 소프트웨어 컴포넌트 geometry 가 포함되고, 그 하위의 tests 디렉토리에는 test\_transcoder.cpp 파일이

위치한다.

**【0057】** 이하의 설명에서는 본 발명에 대한 이해의 편의를 위해, 계속하여 도 3에 도시된 디렉토리 계층 구조를 예시로 들어 설명하기로 한다

**【0058】** S1030 단계에서, 장치(100)는 각 컴포넌트에 대한 원본 오픈소스 소프트웨어의 실제 중복 파일 개수 보다, 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 많은 경우, 디렉토리 계층 구조에 포함된 각 대상 중복 파일의 최소 공통 조상 노드를 기준으로 각 대상 중복 파일이 속한 하위 계층을 분리하여 클러스터를 생성할 수 있다.

**【0059】** 여기서 원본 오픈소스 소프트웨어의 실제 중복 파일 개수보다 대상 중복 파일의 개수가 많다는 의미는, 원본 오픈소스 소프트웨어에서 알려진 중복 파일 수보다 분석 대상 소프트웨어의 디렉토리 계층 구조 상에서 탐지된 동일 파일명의 수가 더 크게 산출된 상태를 의미한다. 즉, 이는 원본 소프트웨어에 존재하지 않는 불필요한 복제, 변형, 또는 중복 배치가 분석 대상 소프트웨어 내에 존재함을 나타내며, 이러한 경우 클러스터 분리를 통해 중복 파일이 속한 경로들을 세분화하여 처리할 필요가 있음을 의미한다.

**【0060】** 따라서, 장치(100)는 오픈소스 소프트웨어에 존재하는 실제 중복 파일 개수의 정보를 저장하도록 사전에 구축된 데이터베이스를 참조하여, 각 컴포넌트에 대한 원본 오픈소스 소프트웨어의 실제 중복 파일 개수를 획득할 수 있다. 이 때 데이터베이스는 오픈소스 소프트웨어 별로 파일명, 디렉토리 경로 및 특정 파일의 중복 사용 횟수에 대한 정보를 포함하며, 이를 통해 특정 컴포넌트 내에서 정상

적으로 존재하는 실제 중복 파일 개수의 기준 값을 제공한다. 따라서 장치(100)는 디렉토리 계층 구조에서 탐지된 대상 중복 파일의 개수와 실제 중복 파일 개수를 비교함으로써, 불필요하게 확장된 중복이나 변형된 파일 배치를 효과적으로 식별할 수 있다.

**【0061】** 도 4는 일 실시예에 따른 디렉토리 계층 구조의 클러스터링 과정에 대한 예시도이다.

**【0062】** 도 4를 참조하면, 장치(100)는 디렉토리 계층 구조에서 탐지된 중복 파일의 개수가 원본 오픈소스 소프트웨어에 존재하는 실제 중복 파일 개수보다 많은 경우, 각 중복 파일의 최소 공통 조상 노드를 기준으로 하여 클러스터를 분리할 수 있다. 예컨대, 원본 오픈소스 소프트웨어에서는 gtest.cc 파일의 중복 개수가 0 개로 알려져 있으나, 분석 대상 소프트웨어에서는 동일한 파일명이 서로 다른 두 위치에서 탐지되므로, 장치(100)는 gtest.cc 파일의 최소 공통 조상 노드인 third\_party를 기준으로 분기되는 디렉토리 경로를 기준으로 Cluster I, Cluster II 및 Cluster III로 분리하여 클러스터링할 수 있다. 이와 같이 클러스터링된 결과를 통해 분석 대상 소프트웨어 내에서 원본 오픈소스 소프트웨어에는 존재하지 않는 불필요한 중복이나 변형된 파일 배치를 효과적으로 식별할 수 있다.

**【0063】** 이때, 장치(100)는 각 클러스터에 포함된 함수의 개수가 원본 오픈 소스 소프트웨어의 전체 함수 개수에서 차지하는 비율을 산출하고, 비율이 미리 설정된 임계값 미만인 클러스터를 노이즈로 판단하여 제거할 수 있다. 즉, 비율이 미리 설정된 임계값 미만이면 해당 클러스터는 오픈소스 소프트웨어와 무관한 코드로

서 정확한 버전 예측을 방해하는 잡음으로 간주하여 제거할 수 있다.

【0064】 도 5는 일 실시예에 따른 클러스터별 함수 비율을 나타낸 예시도이다.

【0065】 도 5를 참조하면, Cluster I과 Cluster II는 오픈소스 소프트웨어의 재사용 함수 비율이 각각 52.8% 및 45.9%를 차지하므로 클러스터가 유지되는 반면, Cluster III는 0.9%로 임계값(예: 3%)에 미달하여 노이즈로 판정되고 제거된다. 이와 같이 임계값 기반의 프루닝 과정을 통해 분석 대상 소프트웨어의 각 클러스터 내에서 의미 없는 경로나 불필요한 중복이 제거되어, 결과적으로 신뢰성 높은 클러스터링 결과를 확보할 수 있다.

【0066】 또한, S1030 단계에서, 원본 오픈소스 소프트웨어의 실제 중복 파일 개수 보다 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 적거나 같은 경우, 장치(100)는 대상 중복 파일의 최소 공통 조상 노드를 기준으로 연관된 모든 하위 노드를 단일 클러스터로 생성할 수도 있다.

【0067】 한편, S1030 단계의 클러스터 생성 과정에서, 장치(100)는 각 클러스터에 포함된 함수가 본래 속한 원본 오픈소스 소프트웨어의 버전 정보를 참조하여, 각 함수에 원본 오픈소스 소프트웨어의 버전 정보를 매핑할 수 있다. 예를 들어, 장치(100)는 사전에 구축된 오픈소스 소프트웨어 메타데이터를 검색하여, 함수의 심볼명, 파일명 및 코드 스니펫을 기준으로 원본 오픈소스 소프트웨어의 특정 버전과 대응시킨다. 이때 검색된 버전 문자열은 시맨틱 버전 규칙(예: major.minor.patch 형식)에 따라 정규화되며, 정규화된 버전 정보가 각 함수에 매

평된다. 이를 통해 동일 함수가 복수의 버전에 걸쳐 존재하는 경우에도 함수 단위로 명확한 버전 추적이 가능해진다

【0068】 S1040 단계에서, 장치(100)는 각 클러스터에 포함된 함수의 오픈소스 소프트웨어 버전을 기초로, 각 클러스터 내에서 공존하는 복수의 버전 분포를 포괄적으로 표현하는 적응형 버전 정보를 생성하고, 적응형 버전 정보를 각 클러스터에 매핑하여 분석 대상 소프트웨어의 SBOM(Software Bill of Materials)를 생성 할 수 있다.

【0069】 즉, 적응형 버전 정보란 단일 버전에 국한되지 않고, 클러스터 내에서 발견된 복수의 함수 버전을 기준으로 대표 버전을 산출하면서, 서로 다른 버전이 공존하는 상황을 기호 또는 확장 규칙을 통해 표현하는 메타 버전을 의미한다.

【0070】 도 6은 일 실시예에 따른 적응형 버전 정보를 생성하는 동작의 예시 도이다.

【0071】 도 6의 { } 내부는 특정 클러스터에 포함된 함수들이 속한 오픈소스 소프트웨어의 버전 집합을 의미한다. 장치가 정규화한 버전 정보는 세 자리로 구성되며, 각 자리는 차례대로 major, minor, patch 버전을 나타낸다. major 버전은 하위 호환성이 깨질 수 있는 큰 변경을 의미하고, minor 버전은 하위 호환성을 유지하면서 기능이 확장된 경우를 의미하며, patch 버전은 기능 추가 없이 버그 수정이나 보안 업데이트와 같이 세부적인 수정을 반영한다.

【0072】 첫 번째 줄의 {1.2.0}은 클러스터에 포함된 모든 함수가 동일한 단일 버전(1.2.0)에 속하는 경우를 나타내므로, 장치(100)는 해당 버전을 그대로 적응형 버전 정보로 결정할 수 있다.

【0073】 두 번째 줄의 {1.2.0, invalid\_ver}은 클러스터에 포함된 모든 함수가 동일한 버전(1.2.0)으로 확인되었으나 일부 함수의 버전 정보가 비유효(invalid)한 버전으로 감지된 경우를 의미한다. 이 경우 장치(100)는 기준 버전인 1.2.0에 '+' 기호를 부가하여 적응형 버전 정보를 생성할 수 있다. 여기서 비유효한 버전이란, 원본 오픈소스 소프트웨어의 메타데이터에서 정상적으로 정의되지 않았거나, 버전 문자열이 누락, 손상, 비정상 형식(예: "unknown", "dev", "-1.0" 등)으로 기록된 경우를 포함한다. 또한, 소스코드 내에서 특정 함수가 어느 버전에 속하는지 추적 불가능하거나, 데이터베이스 참조 시 매칭되지 않아 판별이 곤란한 경우도 비유효한 버전으로 간주될 수 있다.

【0074】 세 번째 줄의 {3.2.0, 2.2.5, 1.2.0}은 동일 클러스터 내 함수들이 서로 다른 major 버전에 속하는 경우를 의미한다. 장치(100)는 가장 낮은 major 버전(1.2.0)을 기준으로 선택하고, 여기에 '\*' 기호를 부가하여 적응형 버전 정보를 산출할 수 있다.

【0075】 네 번째 줄의 {1.2.0, 1.2.5, 1.3.2}는 동일 major 버전에 속하지만 minor 버전이 상이한 경우를 나타낸다. 이 경우 장치(100)는 가장 낮은 minor 버전(1.2.0)을 기준으로 선택하고, 여기에 '^' 기호를 부가하여 적응형 버전 정보를 결정할 수 있다.

【0076】 다섯 번째 줄의 {1.2.0, 1.2.5, 1.2.7}은 동일 major 버전과 minor 버전에 속하나 patch 버전이 상이한 경우를 의미한다. 장치(100)는 가장 낮은 patch 버전(1.2.0)을 기준으로 선택하고, 여기에 '~' 기호를 부가하여 적응형 버전 정보를 생성할 수 있다.

【0077】 이와 같이 생성된 적응형 버전 정보를 이용하여, 장치(100)는 각 클러스터별로 산출된 적응형 버전 정보를 매핑하여, 분석 대상 소프트웨어의 SBOM(Software Bill of Materials)을 생성할 수 있다.

【0078】 도 7은 기준 기법을 통해 생성된 SBOM의 예시도이고, 도 8은 본 발명의 기법을 통해 생성된 SBOM의 예시도이다.

【0079】 도 7의 기준 기법에서는 컴포넌트의 버전을 단일 버전 값으로만 식별하기 때문에, 실제 분석 대상 소프트웨어 내에 서로 다른 하위 버전들이 공존하더라도 SBOM에는 특정 버전(v2.10.0)만 기재된다. 이로 인해, SBOM이 실제 사용 환경의 복수 버전 분포를 정확히 반영하지 못하고, 잠재적인 보안 취약점이나 라이선스 충돌 가능성을 놓칠 수 있다.

【0080】 반면, 도 8의 본 발명의 기법에 따르면, 장치(100)는 각 클러스터 내에서 확인된 다수의 버전 정보를 수집하고 이를 기반으로 적응형 버전 정보를 생성한다. 예컨대, 도 8에서는 v2.9.10, v2.9.12, v2.10.0, v2.10.3과 같은 복수의 버전이 혼재되어 있음에도 불구하고, 장치(100)는 가장 낮은 minor 버전(v2.9.10)을 기준으로 하여 '^' 기호가 부가된 적응형 버전(v^2.9.10)으로 표현한다. 따라서

생성된 SBOM은 실제 소프트웨어 내에 존재하는 다양한 버전 분포를 포괄적으로 반영함으로써, 더욱 정밀하고 신뢰성 있는 구성 정보를 제공할 수 있다. 따라서, 기존의 단일 버전 기반 SBOM에 비해, 함수 단위까지 추적 가능한 세밀한 버전 정보를 제공할 수 있으며, 소프트웨어 구성 요소에 존재하는 잠재적 취약점이나 보안 위협을 보다 정확하게 탐지하고 관리할 수 있다.

【0081】 도 9는 본 발명의 기법과 기존 기법을 비교한 실험 결과를 나타낸 표이다.

【0082】 도 9를 참조하면, 본 발명의 기법(TIVER)은 중복 컴포넌트 식별에서 TP 46, FN 6으로 재현율(Recall) 0.88을 달성한 반면, 기존 기법(CNEPS)은 TP 20, FN 28로 재현율 0.42에 그쳤다. 이는 본 발명이 함수 단위 세분화 버전 식별과 코드 클러스터링 기반의 노이즈 제거 및 중복 탐지 과정을 통해 기존 접근법보다 훨씬 더 정확하게 오픈소스 소프트웨어의 중복 컴포넌트를 판별했음을 나타낸다.

【0083】 도 10은 본 발명의 기법의 실행 시간을 분석한 결과를 나타낸 그래프이다.

【0084】 도 10을 참조하면, x축은 소프트웨어의 코드 라인 수를, y축은 해당 코드를 분석하는 데 소요된 실행 시간을 나타낸다. 그래프에서 확인할 수 있듯이, 수백만 라인 이상의 대규모 코드베이스를 분석하는 경우에도 대부분의 실행 시간이 수 초 이내로 유지되며, 평균 실행 시간은 약 1.67초로 나타났다. 이는 본 발명이 대규모 소프트웨어에서도 실용적으로 적용 가능한 수준의 효율성을 확보하고 있음을 나타낸다.

【0085】 도 11은 일 실시예에 따라 클러스터 별 재사용 함수의 비율을 계산하여 임계값에 따른 클러스터 제거에 따른 탐지 결과의 변화를 나타낸 그래프이다.

【0086】 도 11을 참조하면, x축은 미리 설정한 임계값의 수치를, y축은 탐지 결과의 개수를 나타내며, 빨간색 곡선은 False Positive(FP), 초록색 곡선은 False Negative(FN)를 의미한다. 도 11의 그래프에 따르면 0.03(3%)일 때 FN과 FP가 각각 42, 43으로 교차점을 형성하며 균형을 이루는 것을 확인할 수 있다. 이는 임계값을 0.03으로 설정함으로써 FP와 FN 간의 trade-off를 최적화할 수 있음을 의미한다. 이처럼, 본 발명은 소프트웨어의 특성 및 분석 목적에 따라 임계값을 조정하여, 중복 탐지의 민감도와 정밀도를 상황에 맞게 제어할 수 있다.

【0087】 상술한 실시예에 따르면, 본 발명은 오픈소스 소프트웨어 컴포넌트 내부의 다양한 버전을 정밀하게 식별할 수 있으므로, 기존 방식에서 발생하던 불필요한 오탐을 크게 줄이고 발견되지 못했던 취약점까지 탐지할 수 있는 효과가 있다. 특히, 함수 수준의 세밀한 버전 식별과 코드 클러스터링을 통한 노이즈 제거 및 중복 구분은 각 컴포넌트의 실제 버전 분포를 보다 정확히 반영할 수 있도록 하여, SBOM의 신뢰성을 현저히 향상시킨다. 이와 같은 개선은 단순한 컴포넌트 관리 차원을 넘어, 공급망 전반의 보안 체계를 강화하는 실질적 기반이 된다.

【0088】 더하여, 본 발명은 적응형 버전 개념을 도입함에 따라, 재사용된 코드가 포함할 수 있는 버전의 범위를 명확히 정의할 수 있어 개발자와 보안 담당자가 효율적으로 대응할 수 있도록 지원한다. 이를 통해 SBOM 솔루션과 취약점 관리 시스템의 활용성이 확대되고, 대규모 소프트웨어 생태계에서 오픈소스 소프트웨어

재사용에 따른 위험을 정량적으로 관리할 수 있는 새로운 패러다임을 제시한다.

**【0089】** 본 문서의 다양한 실시예들 및 이에 사용된 용어들은 본 문서에 기재된 기술적 특징들을 특정한 실시예들로 한정하려는 것이 아니며, 해당 실시예의 다양한 변경, 균등물, 또는 대체물을 포함하는 것으로 이해되어야 한다. 도면의 설명과 관련하여, 유사한 또는 관련된 구성요소에 대해서는 유사한 참조 부호가 사용될 수 있다. 아이템에 대응하는 명사의 단수 형은 관련된 문맥상 명백하게 다르게 지시하지 않는 한, 아이템 한 개 또는 복수 개를 포함할 수 있다.

**【0090】** 본 문서에서, "A 또는 B", "A 및 B 중 적어도 하나", "A 또는 B 중 적어도 하나,""A, B 또는 C," "A, B 및 C 중 적어도 하나,"및 "A, B, 또는 C 중 적어도 하나"와 같은 문구들 각각은 그 문구들 중 해당하는 문구에 함께 나열된 항목들의 모든 가능한 조합을 포함할 수 있다. " 1", "제2", 또는 "첫째" 또는 "둘째"와 같은 용어들은 단순히 해당 구성요소를 다른 해당 구성요소와 구분하기 위해 사용될 수 있으며, 해당 구성요소들을 다른 측면(예: 중요성 또는 순서)에서 한정하지 않는다. 어떤(예: 제1) 구성요소가 다른(예: 제2) 구성요소에, "기능적으로" 또는 "통신적으로"라는 용어와 함께 또는 이런 용어 없이, "커플드" 또는 "커넥티드"라고 언급된 경우, 그것은 어떤 구성요소가 다른 구성요소에 직접적으로(예: 유선으로), 무선으로, 또는 제3 구성요소를 통하여 연결될 수 있다는 것을 의미한다.

**【0091】** 본 문서에서 사용된 용어 "모듈"은 하드웨어, 소프트웨어 또는 펌웨어로 구현된 유닛을 포함할 수 있으며, 예를 들면, 로직, 논리 블록, 부품, 또는 회로 등의 용어와 상호 호환적으로 사용될 수 있다. 모듈은, 일체로 구성된 부품

또는 하나 또는 그 이상의 기능을 수행하는, 부품의 최소 단위 또는 그 일부가 될 수 있다. 예를 들면, 일 실시예에 따르면, 모듈은 ASIC(application-specific integrated circuit)의 형태로 구현될 수 있다.

**【0092】** 본 문서의 다양한 실시예들은 기기(예: 전자 장치)에 의해 읽을 수 있는 저장 매체(예: 메모리)에 저장된 하나 이상의 명령어들을 포함하는 소프트웨어(예: 프로그램)로서 구현될 수 있다. 저장 매체는 RAM(random access memory), 메모리 버퍼, 하드 드라이브, 데이터베이스, EPROM(erasable programmable read-only memory), EEPROM(electrically erasable read-only memory), ROM(read-only memory) 및/또는 등을 포함할 수 있다.

**【0093】** 또한, 본 문서의 실시예들의 프로세서는, 저장 매체로부터 저장된 하나 이상의 명령어들 중 적어도 하나의 명령을 호출하고, 그것을 실행할 수 있다. 이것은 기기가 호출된 적어도 하나의 명령어에 따라 적어도 하나의 기능을 수행하도록 운영되는 것을 가능하게 한다. 이러한 하나 이상의 명령어들은 컴파일러에 의해 생성된 코드 또는 인터프리터에 의해 실행될 수 있는 코드를 포함할 수 있다. 프로세서는 범용 프로세서, FPGA(Field Programmable Gate Array), ASIC(Application Specific Integrated Circuit), DSP(Digital Signal Processor) 및/또는 등을 일 수 있다.

**【0094】** 기기로 읽을 수 있는 저장매체는, 비일시적(non-transitory) 저장매체의 형태로 제공될 수 있다. 여기서, '비일시적'은 저장매체가 실재(tangible)하는 장치이고, 신호(예: 전자기파)를 포함하지 않는다는 것을 의미할 뿐이며, 이 용

어는 데이터가 저장매체에 반영구적으로 저장되는 경우와 임시적으로 저장되는 경우를 구분하지 않는다.

【0095】 본 문서에 개시된 다양한 실시예들에 따른 방법은 컴퓨터 프로그램 제품(computer program product)에 포함되어 제공될 수 있다. 컴퓨터 프로그램 제품은 상품으로서 판매자 및 구매자 간에 거래될 수 있다. 컴퓨터 프로그램 제품은 기기로 읽을 수 있는 저장 매체(예: compact disc read only memory (CD-ROM))의 형태로 배포되거나, 또는 어플리케이션 스토어(예: 플레이 스토어)를 통해 또는 두 개의 사용자 장치들(예: 스마트폰들) 간에 직접, 온라인으로 배포(예: 다운로드 또는 업로드)될 수 있다. 온라인 배포의 경우에, 컴퓨터 프로그램 제품의 적어도 일부는 제조사의 서버, 어플리케이션 스토어의 서버, 또는 서버의 메모리와 같은 기기로 읽을 수 있는 저장 매체에 적어도 일시 저장되거나, 임시적으로 생성될 수 있다.

【0096】 다양한 실시예들에 따르면, 기술한 구성요소들의 각각의 구성요소(예: 모듈 또는 프로그램)는 단수 또는 복수의 개체를 포함할 수 있다. 다양한 실시예들에 따르면, 전술한 해당 구성요소들 중 하나 이상의 구성요소들 또는 동작들이 생략되거나, 또는 하나 이상의 다른 구성요소들 또는 동작들이 추가될 수 있다. 대체적으로 또는 추가적으로, 복수의 구성요소들(예: 모듈 또는 프로그램)은 하나의 구성요소로 통합될 수 있다. 이런 경우, 통합된 구성요소는 복수의 구성요소들 각각의 구성요소의 하나 이상의 기능들을 통합 이전에 복수의 구성요소들 중 해당 구성요소에 의해 수행되는 것과 동일 또는 유사하게 수행할 수 있다. 다양한 실시

예들에 따르면, 모듈, 프로그램 또는 다른 구성요소에 의해 수행되는 동작들은 순차적으로, 병렬적으로, 반복적으로, 또는 휴리스틱하게 실행되거나, 동작들 중 하나 이상이 다른 순서로 실행되거나, 생략되거나, 또는 하나 이상의 다른 동작들이 추가될 수 있다.

### 【부호의 설명】

【0097】 100: 장치

110: 메모리

120: 프로세서

130: 입출력 인터페이스

140: 통신 인터페이스

## 【청구범위】

### 【청구항 1】

프로세서에 의해 동작하는 적응형 버전 생성 장치가 수행하는 방법에 있어서,

분석 대상 소프트웨어의 소스코드를 획득하는 동작;

상기 소스코드를 기초로 상기 소프트웨어가 포함하는 재사용된 오픈소스 소프트웨어 컴포넌트, 상기 컴포넌트가 위치하는 디렉토리 및 상기 디렉토리에 포함된 재사용 파일을 포함하는 디렉토리 계층 구조를 생성하는 동작;

각 컴포넌트에 대한 원본 오픈소스 소프트웨어의 실제 중복 파일 개수 보다 상기 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 많은 경우, 상기 디렉토리 계층 구조에 포함된 각 대상 중복 파일의 최소 공통 조상 노드를 기준으로 분기되는 디렉토리 경로를 기준으로 클러스터를 생성하는 동작; 및

각 클러스터에 포함된 함수의 오픈소스 소프트웨어 버전을 기초로, 각 클러스터 내에서 공존하는 복수의 버전 분포를 포괄적으로 표현하는 적응형 버전 정보를 생성하고, 상기 적응형 버전 정보를 각 클러스터에 매핑하여 상기 소프트웨어의 SBOM를 생성하는 동작을 포함하는,

방법.

### 【청구항 2】

제1항에 있어서,

상기 디렉토리 계층 구조를 생성하는 동작은

상기 소스코드에 포함된 재사용된 오픈소스 소프트웨어 컴포넌트의 목록을

식별하는 동작;

각 컴포넌트 별로 재사용된 파일 리스트 및 상기 파일의 위치 경로를 추출하는 동작; 및

상기 파일 리스트 및 위치 경로를 기초로 루트 디렉토리, 상위 디렉토리 및 하위 디렉토리를 포함하는 트리 형태의 디렉토리 계층 구조를 구성하는 동작을 포함하는,

방법.

### 【청구항 3】

제1항에 있어서,

상기 클러스터를 생성하는 동작은

각 클러스터에 포함된 함수의 개수가 원본 오픈소스 소프트웨어의 전체 함수 개수에서 차지하는 비율을 산출하고, 상기 비율이 미리 설정된 임계값 미만인 클러스터를 노이즈로 판단하여 제거하는 동작을 포함하는,

방법.

### 【청구항 4】

제1항에 있어서,

상기 클러스터를 생성하는 동작은

상기 실제 중복 파일 개수 보다 상기 디렉토리 계층 구조에 포함된 대상 중  
복 파일의 개수가 적거나 같은 경우, 상기 대상 중복 파일의 최소 공통 조상 노드  
를 기준으로 단일 클러스터를 생성하는 동작을 포함하는,  
방법.

#### 【청구항 5】

제1항에 있어서,  
상기 소프트웨어의 SBOM를 생성하는 동작은  
제1 클러스터에 포함된 모든 함수가 동일한 제1 버전에 속하는 경우, 상기  
적응형 버전 정보를 상기 제1 버전으로 생성하는 동작을 포함하는,  
방법.

#### 【청구항 6】

제5항에 있어서,  
상기 소프트웨어의 SBOM를 생성하는 동작은  
제1 클러스터에 포함된 모든 함수가 동일한 제1 버전에 속하나, 일부 함수의  
버전이 비유효한 버전에 해당하는 경우, 상기 적응형 버전 정보를 상기 제1 버전에  
'+' 기호를 부가하여 생성하는 동작을 포함하는,  
방법.

#### 【청구항 7】

제6항에 있어서,

상기 소프트웨어의 SBOM를 생성하는 동작은

제1 클러스터에 포함된 함수들이 서로 다른 복수의 major 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 버전 기준에 '\*' 기호를 부가하여 생성하는 동작을 포함하는,  
방법.

### 【청구항 8】

제7항에 있어서,

상기 소프트웨어의 SBOM를 생성하는 동작은  
제1 클러스터에 포함된 함수들이 동일한 major 버전에 속하나 서로 다른 minor 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 minor 버전 기준에 '^' 기호를 부가하여 생성하는 동작을 포함하는,  
방법.

### 【청구항 9】

제8항에 있어서,

상기 소프트웨어의 SBOM를 생성하는 동작은  
제1 클러스터에 포함된 함수들이 동일한 major 버전과 동일한 minor 버전에 속하나 서로 다른 patch 버전에 속하는 경우, 상기 적응형 버전 정보를 가장 낮은 patch 버전 기준에 '~' 기호를 부가하여 생성하는 동작을 포함하는,  
방법.

## 【청구항 10】

명령어를 포함하는 메모리; 및

상기 명령어를 기초로 소정의 동작을 수행하는 프로세서를 포함하고,

상기 프로세서의 동작은

분석 대상 소프트웨어의 소스코드를 획득하는 동작;

상기 소스코드를 기초로 상기 소프트웨어가 포함하는 재사용된 오픈소스 소프트웨어 컴포넌트, 상기 컴포넌트가 위치하는 디렉토리 및 상기 디렉토리에 포함된 재사용 파일을 포함하는 디렉토리 계층 구조를 생성하는 동작;

각 컴포넌트에 대한 원본 오픈소스 소프트웨어의 실제 중복 파일 개수 보다 상기 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 많은 경우, 상기 디렉토리 계층 구조에 포함된 각 대상 중복 파일의 최소 공통 조상 노드를 기준으로 분기되는 디렉토리 경로를 기준으로 클러스터를 생성하는 동작; 및

각 클러스터에 포함된 함수의 오픈소스 소프트웨어 버전을 기초로, 각 클러스터 내에서 공존하는 복수의 버전 분포를 포괄적으로 표현하는 적응형 버전 정보를 생성하고, 상기 적응형 버전 정보를 각 클러스터에 매핑하여 상기 소프트웨어의 SBOM를 생성하는 동작을 포함하는,

적응형 버전 생성 장치.

## 【요약서】

### 【요약】

일 실시예에 따른 적응형 버전 생성 장치는 분석 대상 소프트웨어의 소스코드를 획득하는 동작; 상기 소스코드를 기초로 상기 소프트웨어가 포함하는 재사용된 오픈소스 소프트웨어 컴포넌트, 상기 컴포넌트가 위치하는 디렉토리 및 상기 디렉토리에 포함된 재사용 파일을 포함하는 디렉토리 계층 구조를 생성하는 동작; 각 컴포넌트에 대한 원본 오픈소스 소프트웨어의 실제 중복 파일 개수 보다 상기 디렉토리 계층 구조에 포함된 대상 중복 파일의 개수가 많은 경우, 상기 디렉토리 계층 구조에 포함된 각 대상 중복 파일의 최소 공통 조상 노드를 기준으로 분기되는 디렉토리 경로를 기준으로 클러스터를 생성하는 동작; 및 각 클러스터에 포함된 함수의 오픈소스 소프트웨어 버전을 기초로, 각 클러스터 내에서 공존하는 복수의 버전 분포를 포괄적으로 표현하는 적응형 버전 정보를 생성하고, 상기 적응형 버전 정보를 각 클러스터에 매핑하여 상기 소프트웨어의 SBOM를 생성하는 동작을 수행할 수 있다.

### 【대표도】

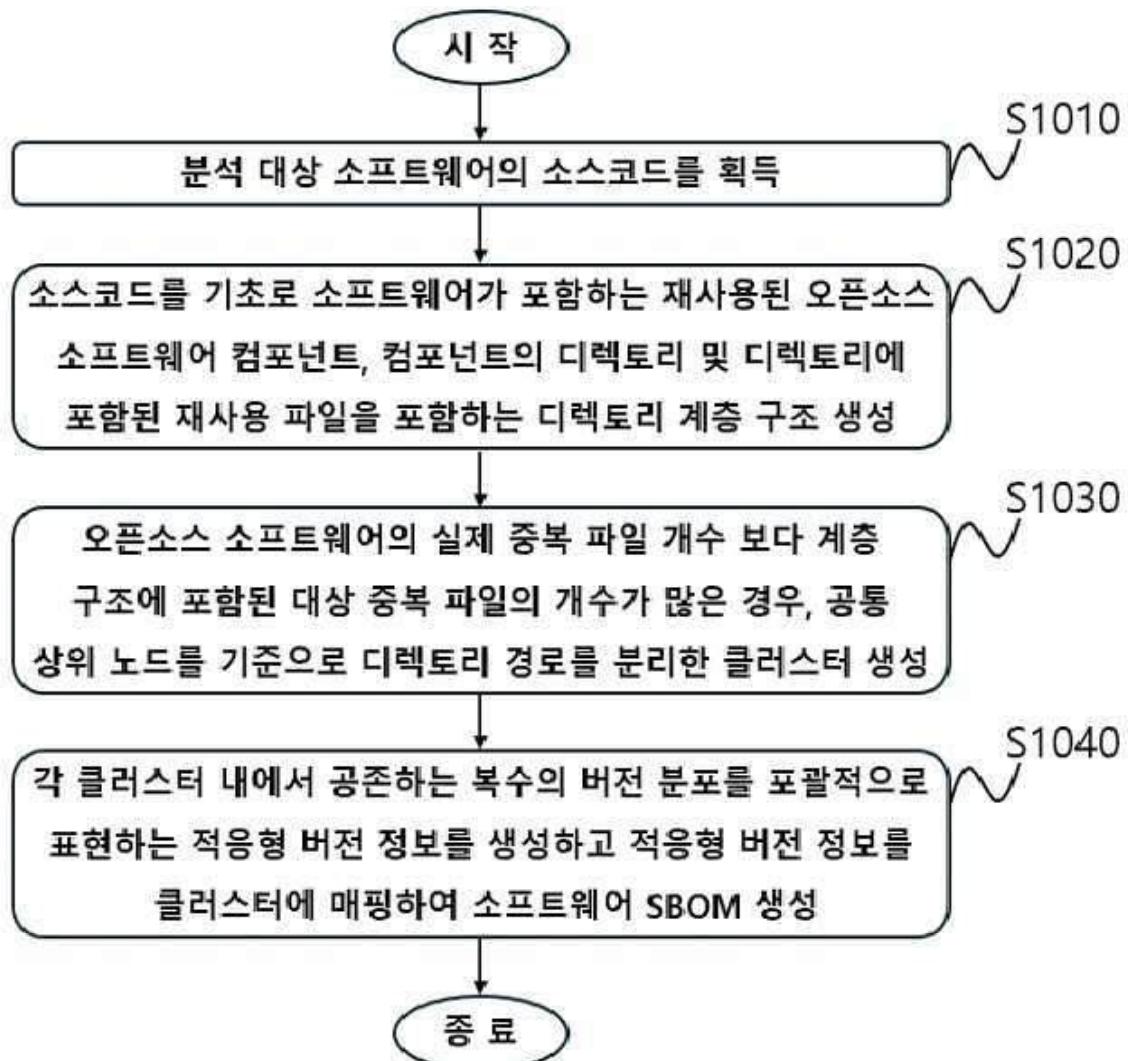
도 2

【도면】

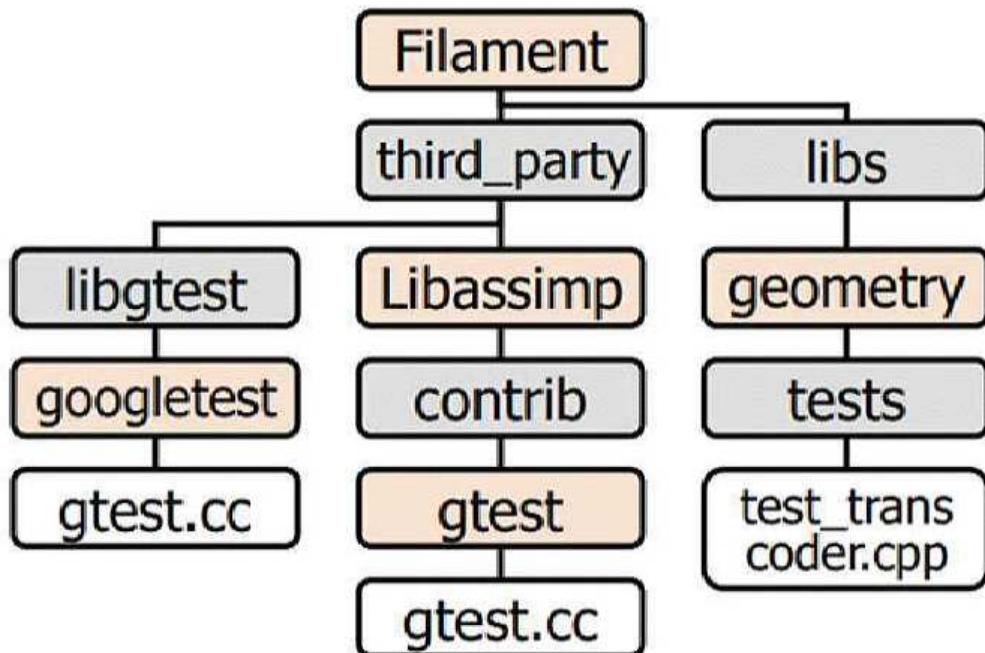
【도 1】



## 【도 2】



## 【도 3】

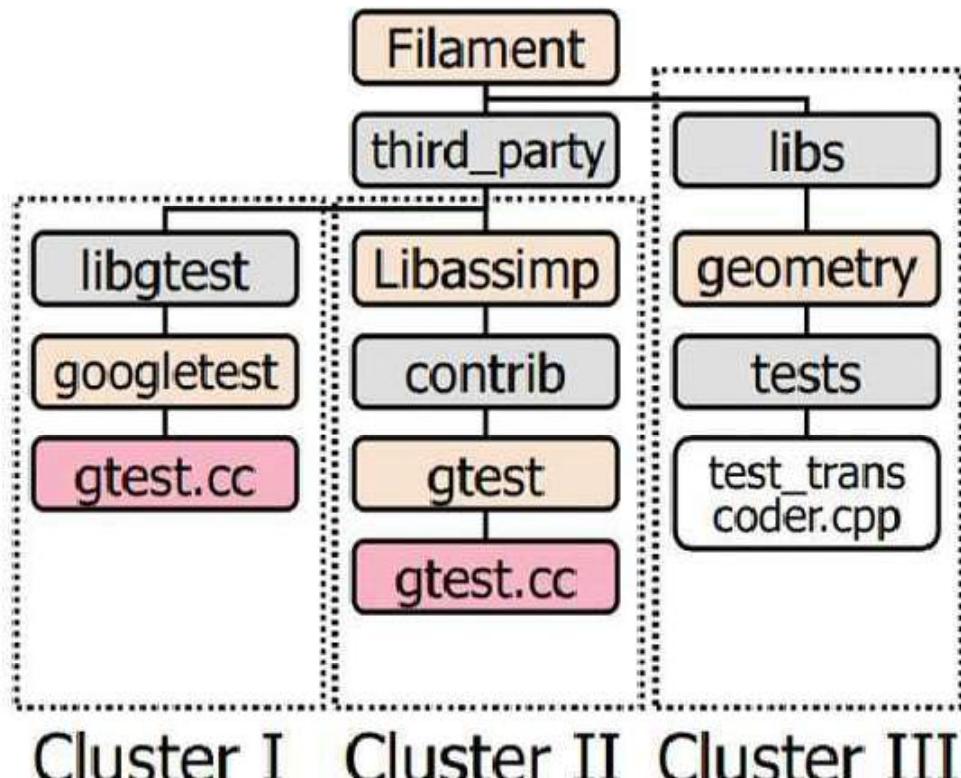


OSS

dir

file

## 【도 4】

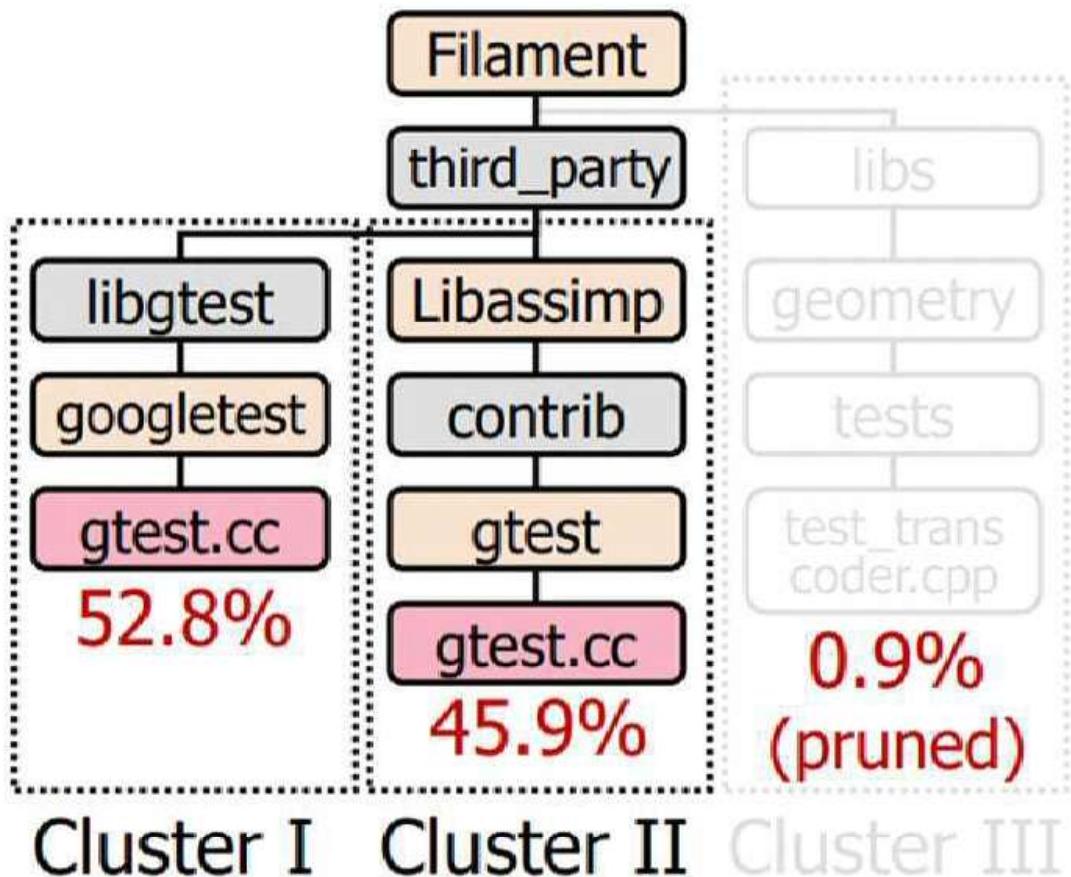


원본 OSS의  
중복 파일 개수 : 0



대상 SW의  
중복 파일 개수 : 2

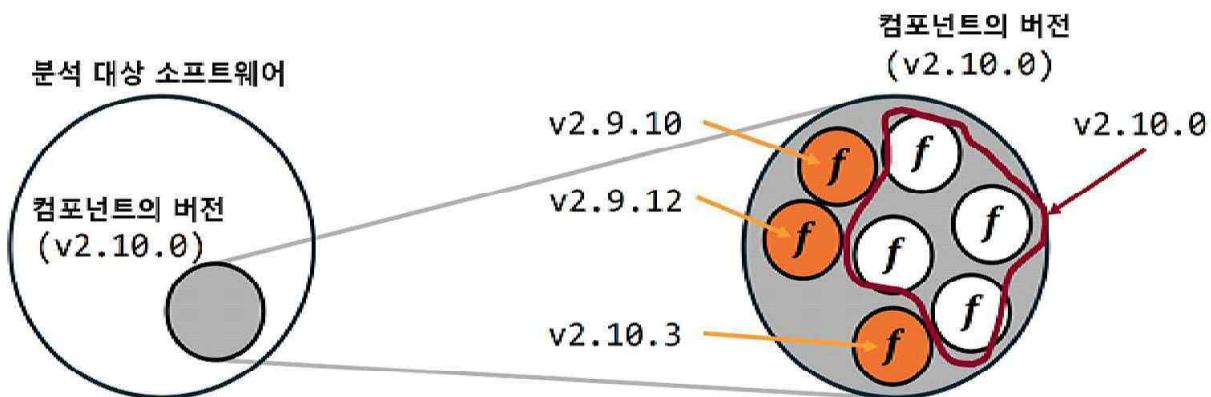
【도 5】



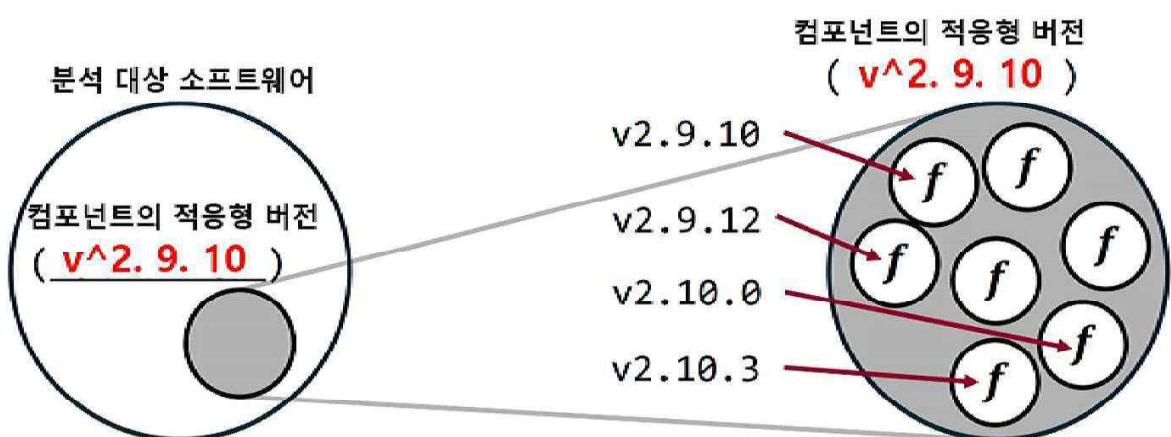
【도 6】

{1.2.0} ->	1.2.0
{1.2.0, invalid_ver <sup>†</sup> } ->	+1.2.0
{3.2.0, 2.2.5, 1.2.0} ->	*1.2.0
{1.2.0, 1.2.5, 1.3.2} ->	^1.2.0
{1.2.0, 1.2.5, 1.2.7} ->	~1.2.0

【도 7】



【도 8】

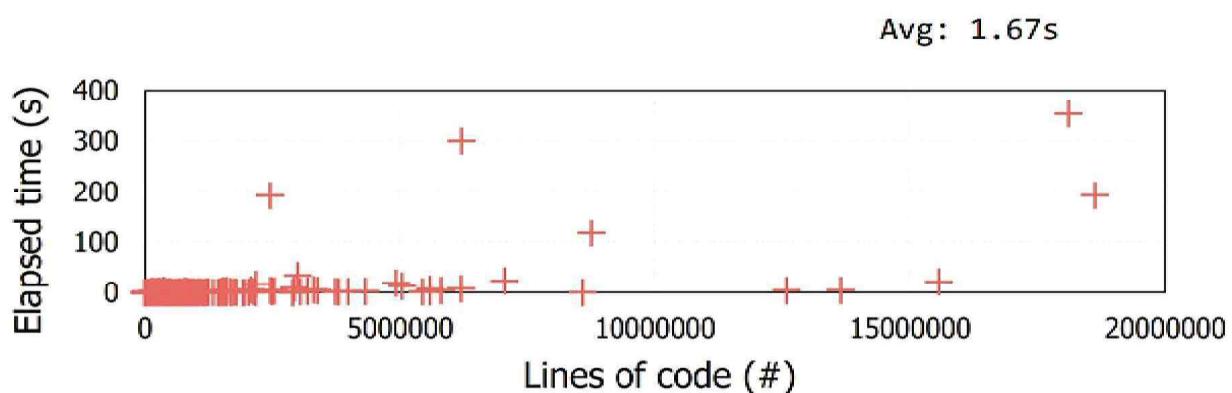


【도 9】

## VS. CNEPS (ICSE 2024)

	TIVER	CNEPS
TP	46	20
FN	6	28
Recall	0.88	0.42

【도 10】



## 【도 11】

