(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2022년12월08일
(11) 등록번호 10-2476358
(24) 등록일자 2022년12월06일

(51) 국제특허분류(Int. Cl.)
G06F 8/75 (2018.01) *G06F 16/215* (2019.01)
G06F 16/22 (2019.01) *G06F 8/36* (2018.01)
(52) CPC특허분류
G06F 8/75 (2013.01)
G06F 16/215 (2019.01)
(21) 출원번호 10-2021-0010585
(22) 출원일자 2021년01월26일
심사청구일자 2021년01월26일
(65) 공개번호 10-2022-0107677
(43) 공개일자 2022년08월02일
(56) 선행기술조사문헌
KR101568224 B1*
KR101780233 B1*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
이희조
우승훈
(74) 대리인
이대호, 박건홍

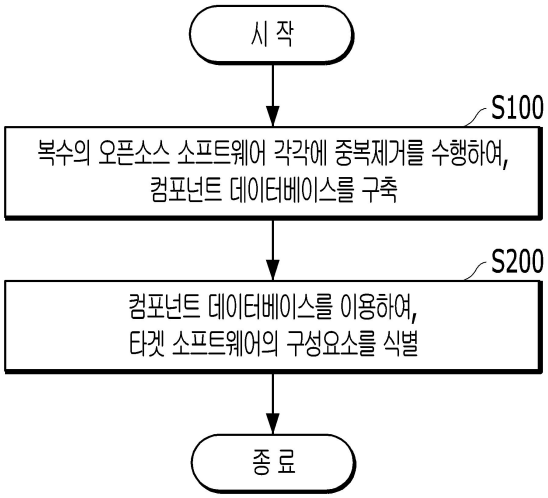
전체 청구항 수 : 총 12 항 심사관 : 지정훈

(54) 발명의 명칭 **소스코드 레벨에서의 오픈소스 소프트웨어 (OSS) 구성요소 식별을 위한 방법**

(57) 요 약

본 개시의 몇몇 실시예에 따른, 컴퓨팅 장치의 프로세서를 이용한 오픈소스 소프트웨어(Open Source Software: OSS) 구성요소 식별을 위한 방법이 개시된다. 상기 오픈소스 소프트웨어 구성요소 식별을 위한 방법은, 복수의 오픈소스 소프트웨어 각각에 중복제거를 수행하여, 컴포넌트 데이터베이스를 구축하는 단계; 및 상기 컴포넌트 데이터베이스를 이용하여, 타겟 소프트웨어의 구성요소를 식별하는 단계를 포함할 수 있다.

대 표 도 - 도2



(52) CPC특허분류
G06F 16/2255 (2019.01)
G06F 8/36 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711116238
과제번호	2019-0-01697-002
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	유망 신기술 및 글로벌선도기술 확보
연구과제명	블록체인 플랫폼 보안취약점 자동분석 기술 개발
기 여 율	1/1
과제수행기관명	고려대학교 산학협력단
연구기간	2019.06.01 ~ 2022.12.31

명 세 서

청구범위

청구항 1

컴퓨팅 장치의 프로세서를 이용한 오픈소스 소프트웨어(Open Source Software: OSS) 구성요소 식별을 위한 방법에 있어서,

복수의 오픈소스 소프트웨어 각각에 중복제거를 수행하여, 컴포넌트 데이터베이스를 구축하는 단계; 및

상기 컴포넌트 데이터베이스를 이용하여, 타겟 소프트웨어의 구성요소를 식별하는 단계;

를 포함하고,

상기 복수의 오픈소스 소프트웨어 각각에 중복제거를 수행하여, 컴포넌트 데이터베이스를 구축하는 단계는,

상기 복수의 오픈소스 소프트웨어 중 제 1 오픈소스 소프트웨어에 대한 중복 제거를 수행하는 경우, 상기 제 1 오픈소스 소프트웨어의 적어도 하나의 버전 각각에 등장하는 적어도 하나의 함수를 인식하는 단계; 및

상기 적어도 하나의 함수 각각의 해시 값을 키(key)로 설정하고, 상기 적어도 하나의 함수 각각이 등장하는 적어도 하나의 버전을 상기 키에 대한 값(value)으로 설정하여 제 1 오픈소스 소프트웨어에 대한 제 1 딕셔너리(dictionary) 자료구조에 저장하는 단계;

를 포함하고,

상기 제 1 딕셔너리(dictionary) 자료구조는,

함수가 등장하는 버전의 개수에 따라 서로 다른 그룹으로 구분되는,

오픈소스 소프트웨어 구성요소 식별을 위한 방법.

청구항 2

삭제

청구항 3

삭제

청구항 4

제 1 항에 있어서,

상기 컴포넌트 데이터베이스를 이용하여, 타겟 소프트웨어의 구성요소를 식별하는 단계는,

상기 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를 고유한 코드 부분 및 빌려온 코드 부분으로 세분화하는 단계; 및

상기 타겟 소프트웨어에 포함된 타겟 소스 코드 및 상기 복수의 오픈소스 소프트웨어 각각에 포함된 상기 고유한 코드 부분을 비교하여, 상기 복수의 오픈소스 소프트웨어 중 상기 타겟 소프트웨어 작성 시 사용된 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 단계;

를 포함하는,

오픈소스 소프트웨어 구성요소 식별을 위한 방법.

청구항 5

제 4 항에 있어서,

상기 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를 고유한 코드 부분 및 빌려온 코드 부분으로 세분화하는 단계는,

상기 복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수를 인식하는 단계;

상기 복수의 오픈소스 소프트웨어 중 상기 제 3 오픈소스 소프트웨어와 비교 분석할 제 4 오픈소스 소프트웨어를 선정하는 단계;

상기 제 4 오픈소스 소프트웨어에 포함된 제 2 소스 코드에 등장하는 하나 이상의 제 2 함수를 인식하는 단계;

상기 하나 이상의 제 1 함수와 상기 하나 이상의 제 2 함수를 비교하여 동일하다고 인식되는 하나 이상의 공통 함수를 추출하는 단계;

상기 하나 이상의 공통 함수 중 상기 제 4 오픈소스 소프트웨어에 먼저 등장한 하나 이상의 제 3 함수를 인식하는 단계; 및

상기 컴포넌트 데이터베이스 내의 상기 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에서 상기 하나 이상의 제 3 함수를 제거하는 단계;

를 포함하는,

오픈소스 소프트웨어 구성요소 식별을 위한 방법.

청구항 6

제 5 항에 있어서,

상기 컴포넌트 데이터베이스 내의 상기 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리(dictionary) 자료구조에서 상기 하나 이상의 제 3 함수를 제거하는 단계는,

상기 하나 이상의 제 2 함수의 제 1 개수와 상기 하나 이상의 제 3 함수의 제 2 개수에 기초하여 산출된 값이 기 설정된 값 이상인 경우, 상기 제 2 딕셔너리 자료구조에 포함된 상기 하나 이상의 제 1 함수에서 상기 하나 이상의 제 3 함수를 제거하는 단계;

를 포함하는,

오픈소스 소프트웨어 구성요소 식별을 위한 방법.

청구항 7

제 4 항에 있어서,

상기 타겟 소프트웨어에 포함된 타겟 소스 코드 및 상기 복수의 오픈소스 소프트웨어 각각에 포함된 상기 고유한 코드 부분을 비교하여, 상기 복수의 오픈소스 소프트웨어 중 상기 타겟 소프트웨어 작성 시 사용된 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 단계는,

상기 타겟 소스 코드에 포함된 함수와 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 상기 고유한 코드 부분의 함수에 공통적으로 포함되는 함수의 제 3 개수 및 상기 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 상기 고유한 코드 부분의 함수의 제 4 개수에 기초하여 상기 복수의 오픈소스 소프트웨어 각각과 상기 타겟 소프트웨어 사이의 코드 유사도 값 각각을 인식하는 단계; 및

상기 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 중 상기 코드 유사도 값이 기 설정된 값 이상인 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 단계;

를 포함하는,

오픈소스 소프트웨어 구성요소 식별을 위한 방법.

청구항 8

제 7 항에 있어서,

상기 코드 유사도 값은 제 1 수학식에 기초하여 결정되고,

상기 제 1 수학식은,

$\Phi = \frac{|T \cap S|}{|S|}$ 이고,

상기 Φ는 코드유사도 값이고, 상기 T는 타겟 소프트웨어의 함수이고, 상기 S는 코드 세분화를 수행한 복수의 오픈소스 소프트웨어의 고유한 코드 부분의 함수인,

오픈소스 소프트웨어 구성요소 식별을 위한 방법.

청구항 9

복수의 오픈소스 소프트웨어 각각에 중복제거를 수행하는 프로세서; 및

상기 중복제거를 수행한 결과를 저장하는 컴포넌트 데이터베이스부;

를 포함하고,

상기 프로세서는,

상기 컴포넌트 데이터베이스부를 이용하여, 타겟 소프트웨어의 구성요소를 식별하고,

상기 복수의 오픈소스 소프트웨어 중 제 1 오픈소스 소프트웨어에 대한 중복 제거를 수행하는 경우, 상기 제 1 오픈소스 소프트웨어의 적어도 하나의 버전 각각에 등장하는 적어도 하나의 함수를 인식하고, 그리고

상기 적어도 하나의 함수 각각의 해시 값을 키(key)로 설정하고, 상기 적어도 하나의 함수 각각이 등장하는 적어도 하나의 버전을 상기 키에 대한 값(value)으로 설정하여 제 1 오픈소스 소프트웨어에 대한 제 1 딕셔너리(dictionary) 자료구조로 상기 컴포넌트 데이터베이스부에 저장하고,

상기 제 1 딕셔너리(dictionary) 자료구조는,

함수가 등장하는 버전의 개수에 따라 서로 다른 그룹으로 구분되는,

오픈소스 소프트웨어 구성요소 식별을 위한 장치.

청구항 10

삭제

청구항 11

삭제

청구항 12

제 9 항에 있어서,

상기 프로세서는,

상기 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를 고유한 코드 부분 및 빌려온 코드 부분으로 세분화하고,

상기 타겟 소프트웨어에 포함된 타겟 소스 코드 및 상기 복수의 오픈소스 소프트웨어 각각에 포함된 상기 고유

한 코드 부분을 비교하여, 상기 복수의 오픈소스 소프트웨어 중 상기 타겟 소프트웨어 작성 시 사용된 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는,

오픈소스 소프트웨어 구성요소 식별을 위한 장치.

청구항 13

제 12 항에 있어서,

상기 프로세서는,

상기 복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수를 인식하고,

상기 복수의 오픈소스 소프트웨어 중 상기 제 3 오픈소스 소프트웨어와 비교 분석할 제 4 오픈소스 소프트웨어를 선정하고,

상기 제 4 오픈소스 소프트웨어에 포함된 제 2 소스 코드에 등장하는 하나 이상의 제 2 함수를 인식하고,

상기 하나 이상의 제 1 함수와 상기 하나 이상의 제 2 함수를 비교하여 동일하다고 인식되는 하나 이상의 공통 함수를 추출하고,

상기 하나 이상의 공통 함수 중 상기 제 4 오픈소스 소프트웨어에 먼저 등장한 하나 이상의 제 3 함수를 인식하고,

상기 컴포넌트 데이터베이스부 내의 상기 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에서 상기 하나 이상의 제 3 함수를 제거하는 오픈소스 소프트웨어 구성요소 식별을 위한 장치.

청구항 14

제 13 항에 있어서,

상기 프로세서는,

상기 하나 이상의 제 2 함수의 제 1 개수와 상기 하나 이상의 제 3 함수의 제 2 개수에 기초하여 산출된 값이 기 설정된 값 이상인 경우, 상기 제 2 딕셔너리 자료구조에 포함된 상기 하나 이상의 제 1 함수에서 상기 하나 이상의 제 3 함수를 제거하는,

오픈소스 소프트웨어 구성요소 식별을 위한 장치.

청구항 15

제 12 항에 있어서,

상기 프로세서는,

상기 타겟 소스 코드에 포함된 함수와 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 상기 고유한 코드 부분의 함수에 공통적으로 포함되는 함수의 제 3 개수 및 상기 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 상기 고유한 코드 부분의 함수의 제 4 개수에 기초하여 상기 복수의 오픈소스 소프트웨어 각각과 상기 타겟 소프트웨어 사이의 코드 유사도 값 각각을 인식하고,

상기 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 중 상기 코드 유사도 값이 기 설정된 값 이상인 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는,

오픈소스 소프트웨어 구성요소 식별을 위한 장치.

청구항 16

제 15 항에 있어서,

상기 코드 유사도 값은 제 1 수학식에 기초하여 결정되고,

$\Phi = \frac{|T \cap S|}{|S|}$ 이고,

상기 Φ는 코드유사도 값이고, 상기 T는 타겟 소프트웨어의 함수이고, 상기 S는 코드 세분화를 수행한 복수의 오픈소스 소프트웨어의 고유한 코드 부분의 함수인,

오픈소스 소프트웨어 구성요소 식별을 위한 장치.

발명의 설명

기 술 분 야

[0001] 본 개시는 특정 소프트웨어가 재사용하고 있는 오픈소스 소프트웨어 구성요소 탐지에 관한 것으로, 구체적으로 재사용 중인 오픈소스 소프트웨어 리스트를 비롯하여 재사용 패턴까지도 확장성있고 정확하게 탐지하기 위한 방법에 관한 것이다.

배 경 기 술

[0003] 오픈소스 소프트웨어(Open Source Software: OSS)란 소스코드가 공개되어 있으면서, 라이선스를 준수하는 한 누구나 특별한 제한 없이 재사용, 수정 및 재배포가 가능한 소프트웨어를 의미할 수 있다. 개발자들은 소프트웨어 개발 시 필요한 세부 기능들을 일일이 구현하지 않고, 기 구현되어 있는 오픈소스 소프트웨어 코드의 재사용을 통해 소프트웨어 개발 시간 및 비용을 단축할 수 있다.

[0004] 이러한 이점에도 불구하고, 관리되지 않은 무분별한 오픈소스 소프트웨어의 재사용은 여러 가지 문제를 야기할 수 있다. 대표적으로 취약한 오픈소스 소프트웨어가 재사용되면서 취약점을 전파시키는 문제와 라이선스 정책을 따르지 않는 오픈소스 소프트웨어의 재사용으로 인한 라이선스 위반 문제가 발생될 수 있다.

[0005] 재사용 중인 오픈소스 소프트웨어 구성요소를 명확하게 탐지할 수 있으면 이러한 문제를 예방할 수 있다. 하지만 수정된 오픈소스 소프트웨어의 재사용은 정확한 구성요소 탐지를 도전적인 문제로 만들었다. 개발자들은 오픈소스 소프트웨어를 재사용하는 과정에서 소스코드의 일부분만 재사용하거나, 코드 및 구조를 수정하여 재사용하는 경우가 잦다. 만일 타겟 소프트웨어와 특정 오픈소스 소프트웨어사이에 공통된 소스 코드가 일부 존재한다는 사실을 확인했을 때, 실제로 오픈소스 소프트웨어의 일부분을 재사용하고 있는 것인지(정탐), 혹은 오픈소스 소프트웨어내의 서브-구성요소만 재사용하고 있는 것인지(이 경우, 전체 오픈소스 소프트웨어를 재사용하고 있다고 판단하는 것은 오탐)를 구분하는 것은 쉽지 않은 문제이다.

[0006] 이와 더불어, 꾸준히 증가하는 오픈소스 소프트웨어의 개수 및 각 오픈소스 소프트웨어의 코드 사이즈는 합리적인 시간 내의 오픈소스 소프트웨어 구성요소의 탐지를 더욱 어렵게 만들 수 있다.

[0007] 오픈소스 소프트웨어 구성요소를 바이너리 레벨에서 탐지하고자 했던 기술 “Identifying Open-Source License Violation and 1-day Security Risk at Large Scale” 이 2017년 ACM Conference on Computer and Communications Security (CCS)에 발표가 되었으나, 해당 기술은 수정된 오픈소스 소프트웨어 구성요소를 충분히 고려하지 않았고, 큰 사이즈의 소프트웨어에서 오픈소스 소프트웨어 구성요소를 탐지하기에는 확장성도 부족할 수 있다.

[0008] 따라서, 수정된 오픈소스 소프트웨어의 재사용을 고려하면서 정확하고 확장성있게 오픈소스 소프트웨어 구성요소를 탐지할 방법이 필요하다.

선행기술문헌

특허문헌

[0010] (특허문헌 0001) 대한민국 등록특허 10-1780233

발명의 내용

해결하려는 과제

[0011] 본 개시는 전술한 배경기술에 대응하여 안출된 것으로, 대규모의 오픈소스 소프트웨어 집합으로부터 특정 소프트웨어가 재사용 중인 오픈소스 소프트웨어 구성요소 리스트 및 그들의 재사용 패턴을 정확하게 탐지하는 방법을 제공하고자 한다.

[0012] 본 개시의 기술적 과제들은 이상에서 언급한 기술적 과제로 제한되지 않으며, 언급되지 않은 또 다른 기술적 과제들은 아래의 기재로부터 당업자에게 명확하게 이해될 수 있을 것이다.

과제의 해결 수단

[0014] 전술한 바와 같은 과제를 해결하기 위한 본 개시의 일 실시예에 따라, 컴퓨팅 장치의 프로세서를 이용한 오픈소스 소프트웨어 구성요소 식별을 위한 방법이 개시된다. 상기 오픈소스 소프트웨어 구성요소 식별을 위한 방법은, 복수의 오픈소스 소프트웨어 각각에 중복제거를 수행하여, 컴포넌트 데이터베이스를 구축하는 단계; 및 상기 컴포넌트 데이터베이스를 이용하여, 타겟 소프트웨어의 구성요소를 식별하는 단계; 를 포함할 수 있다.

[0015] 또한, 상기 복수의 오픈소스 소프트웨어 각각에 중복제거를 수행하여, 컴포넌트 데이터베이스를 구축하는 단계는, 상기 복수의 오픈소스 소프트웨어 중 제 1 오픈소스 소프트웨어에 대한 중복 제거를 수행하는 경우, 상기 제 1 오픈소스 소프트웨어의 적어도 하나의 버전 각각에 등장하는 적어도 하나의 함수를 인식하는 단계; 및 상기 적어도 하나의 함수 각각의 해시 값을 키(key)로 설정하고, 상기 적어도 하나의 함수 각각이 등장하는 하나 이상의 버전을 상기 키에 대한 값(value)으로 설정하여 제 1 오픈소스 소프트웨어에 대한 제 1 딕셔너리(dictionary) 자료구조에 저장하는 단계; 를 포함할 수 있다.

[0016] 또한, 상기 제 1 딕셔너리(dictionary) 자료구조는, 함수가 등장하는 버전의 개수에 따라 서로 다른 그룹으로 구분될 수 있다.

[0017] 또한, 상기 컴포넌트 데이터베이스를 이용하여, 타겟 소프트웨어의 구성요소를 식별하는 단계는, 상기 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를 고유한 코드 부분 및 빌려온 코드 부분으로 세분화하는 단계; 및 상기 타겟 소프트웨어에 포함된 타겟 소스 코드 및 상기 복수의 오픈소스 소프트웨어 각각에 포함된 상기 고유한 코드 부분을 비교하여, 상기 복수의 오픈소스 소프트웨어 중 상기 타겟 소프트웨어 작성 시 사용된 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 단계; 를 포함할 수 있다.

[0018] 또한, 상기 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를 고유한 코드 부분 및 빌려온 코드 부분으로 세분화하는 단계는, 상기 복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수를 인식하는 단계; 상기 복수의 오픈소스 소프트웨어 중 상기 제 3 오픈소스 소프트웨어와 비교 분석할 제 4 오픈소스 소프트웨어를 선정하는 단계; 상기 제 4 오픈소스 소프트웨어에 포함된 제 2 소스 코드에 등장하는 하나 이상의 제 2 함수를 인식하는 단계; 상기 하나 이상의 제 1 함수와 상기 하나 이상의 제 2 함수를 비교하여 동일하다고 인식되는 하나 이상의 공통 함수를 추출하는 단계; 상기 하나 이상의 공통 함수 중 상기 제 4 오픈소스 소프트웨어에 먼저 등장한 하나 이상의 제 3 함수를 인식하는 단계; 및 상기 컴포넌트 데이터베이스 내의 상기 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에서 상기 하나 이상의 제 3 함수를 제거하는 단계; 를 포함할 수 있다.

[0019] 또한, 상기 컴포넌트 데이터베이스 내의 상기 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리(dictionary) 자료구조에서 상기 하나 이상의 제 3 함수를 제거하는 단계는, 상기 하나 이상의 제 2 함수의 제 1 개수와 상기 하나 이상의 제 3 함수의 제 2 개수에 기초하여 산출된 값이 기 설정된 값 이상인 경우, 상기 제 2 딕셔너리 자료구조에 포함된 상기 하나 이상의 제 1 함수에서 상기 하나 이상의 제 3 함수를 제거하는 단계; 를 포함할 수 있다.

[0020] 또한, 상기 타겟 소프트웨어에 포함된 타겟 소스 코드 및 상기 복수의 오픈소스 소프트웨어 각각에 포함된 상기 고유한 코드 부분을 비교하여, 상기 복수의 오픈소스 소프트웨어 중 상기 타겟 소프트웨어 작성 시 사용된 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 단계는, 상기 타겟 소스 코드에 포함된 함수와 코드 세분화를

수행한 복수의 오픈소스 소프트웨어 각각의 상기 고유한 코드 부분의 함수에 공통적으로 포함되는 함수의 제 3 개수 및 상기 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 상기 고유한 코드 부분의 함수의 제 4 개수에 기초하여 상기 복수의 오픈소스 소프트웨어 각각과 상기 타겟 소프트웨어 사이의 코드 유사도 값 각각을 인식하는 단계; 및 상기 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 중 상기 코드 유사도 값이 기 설정된 값 이상인 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 단계; 를 포함할 수 있다.

[0021] 또한, 상기 코드 유사도 값은 제 1 수학식에 기초하여 결정되고, 상기 제 1 수학식은, $\Phi = \frac{|T \cap S|}{|S|}$ 이고, 상기 Φ는 코드유사도 값이고, 상기 T는 타겟 소프트웨어의 함수이고, 상기 S는 코드 세분화를 수행한 복수의 오픈소스 소프트웨어의 고유한 코드 부분의 함수일 수 있다.

[0022] 또한, 복수의 오픈소스 소프트웨어 각각에 중복제거를 수행하는 프로세서; 및 상기 중복제거를 수행한 결과를 저장하는 컴포넌트 데이터베이스부; 를 포함하고, 상기 프로세서는, 상기 컴포넌트 데이터베이스부를 이용하여, 타겟 소프트웨어의 구성요소를 식별할 수 있다.

[0023] 또한, 상기 프로세서는, 상기 복수의 오픈소스 소프트웨어 중 제 1 오픈소스 소프트웨어에 대한 중복 제거를 수행하는 경우, 상기 제 1 오픈소스 소프트웨어의 적어도 하나의 버전 각각에 등장하는 적어도 하나의 함수를 인식하고, 상기 적어도 하나의 함수 각각의 해시 값을 키(key)로 설정하고, 상기 적어도 하나의 함수 각각이 등장하는 하나 이상의 버전을 상기 키에 대한 값(value)으로 설정하여 제 1 오픈소스 소프트웨어에 대한 제 1 딕셔너리(dictionary) 자료구조로 상기 컴포넌트 데이터베이스부에 저장할 수 있다.

[0024] 또한, 상기 제 1 딕셔너리(dictionary) 자료구조는, 함수가 등장하는 버전의 개수에 따라 서로 다른 그룹으로 구분될 수 있다.

[0025] 또한, 상기 프로세서는, 상기 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를 고유한 코드 부분 및 빌려온 코드 부분으로 세분화하고, 상기 타겟 소프트웨어에 포함된 타겟 소스 코드 및 상기 복수의 오픈소스 소프트웨어 각각에 포함된 상기 고유한 코드 부분을 비교하여, 상기 복수의 오픈소스 소프트웨어 중 상기 타겟 소프트웨어 작성 시 사용된 적어도 하나의 제 2 오픈소스 소프트웨어를 추출할 수 있다.

[0026] 또한, 상기 프로세서는, 상기 복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수를 인식하고, 상기 복수의 오픈소스 소프트웨어 중 상기 제 3 오픈소스 소프트웨어와 비교 분석할 제 4 오픈소스 소프트웨어를 선정하고, 상기 제 4 오픈소스 소프트웨어에 포함된 제 2 소스 코드에 등장하는 하나 이상의 제 2 함수를 인식하고, 상기 하나 이상의 제 1 함수와 상기 하나 이상의 제 2 함수를 비교하여 동일하다고 인식되는 하나 이상의 공통 함수를 추출하고, 상기 하나 이상의 공통 함수 중 상기 제 4 오픈소스 소프트웨어에 먼저 등장한 하나 이상의 제 3 함수를 인식하고, 상기 컴포넌트 데이터베이스부 내의 상기 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에서 상기 하나 이상의 제 3 함수를 제거할 수 있다.

[0027] 또한, 상기 프로세서는, 상기 하나 이상의 제 2 함수의 제 1 개수와 상기 하나 이상의 제 3 함수의 제 2 개수에 기초하여 산출된 값이 기 설정된 값 이상인 경우, 상기 제 2 딕셔너리 자료구조에 포함된 상기 하나 이상의 제 1 함수에서 상기 하나 이상의 제 3 함수를 제거할 수 있다.

[0028] 또한, 상기 프로세서는, 상기 타겟 소스 코드에 포함된 함수와 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 상기 고유한 코드 부분의 함수에 공통적으로 포함되는 함수의 제 3 개수 및 상기 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 상기 고유한 코드 부분의 함수의 제 4 개수에 기초하여 상기 복수의 오픈소스 소프트웨어 각각과 상기 타겟 소프트웨어 사이의 코드 유사도 값 각각을 인식하고, 상기 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 중 상기 코드 유사도 값이 기 설정된 값 이상인 적어도 하나의 제 2 오픈소스 소프트웨어를 추출할 수 있다.

[0029] 또한, 상기 코드 유사도 값은 제 1 수학식에 기초하여 결정되고, 상기 제 1 수학식은, $\Phi = \frac{|T \cap S|}{|S|}$ 이고, 상기 Φ는 코드유사도 값이고, 상기 T는 타겟 소프트웨어의 함수이고, 상기 S는 코드 세분화를 수행한 복수의 오픈소스 소프트웨어의 고유한 코드 부분의 함수일 수 있다.

[0030] 본 개시에서 얻을 수 있는 기술적 해결 수단은 이상에서 언급한 해결 수단들로 제한되지 않으며, 언급하지 않은 또 다른 해결 수단들은 아래의 기재로부터 본 개시가 속하는 기술분야에서 통상의 지식을 가진 자에게 명확하게 이해될 수 있을 것이다.

발명의 효과

[0032] 본 개시의 몇몇 실시예에 따르면, 취약점 존재 여부, 라이선스 위반 여부를 포함한 소프트웨어의 보안성 향상 및 소프트웨어 코드 관리를 효과적으로 수행할 수 있다.

[0033] 본 개시에서 얻을 수 있는 효과는 이상에서 언급한 효과로 제한되지 않으며, 언급하지 않은 또 다른 효과들은 아래의 기재로부터 본 개시가 속하는 기술분야에서 통상의 지식을 가진 자에게 명확하게 이해될 수 있을 것이다.

도면의 간단한 설명

[0035] 다양한 양상들이 이제 도면들을 참조로 기재되며, 여기서 유사한 참조 번호들은 총괄적으로 유사한 구성요소들을 지칭하는데 이용된다. 이하의 실시예에서, 설명 목적을 위해, 다수의 특정 세부사항들이 하나 이상의 양상들의 총체적 이해를 제공하기 위해 제시된다. 그러나, 그러한 양상(들)이 이러한 특정 세부사항들 없이 실시될 수 있음은 명백할 것이다. 다른 예시들에서, 공지의 구조들 및 장치들이 하나 이상의 양상들의 기재를 용이하게 하기 위해 블록도 형태로 도시된다.

도 1은 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치의 일례를 설명하기 위한 블록 구성도이다.

도 2는 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 오픈소스 소프트웨어의 구성요소를 식별하기 위한 방법의 일례를 설명하기 위한 흐름도이다.

도 3은 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 컴포넌트 데이터베이스를 구축하는 방법의 일례를 설명하기 위한 흐름도이다.

도 4는 본 개시의 몇몇 실시예에 따른 딕셔너리 자료구조를 설명하기 위한 도면이다.

도 5는 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 타겟 소프트웨어의 구성요소를 식별하는 방법의 일례를 설명하기 위한 흐름도이다.

도 6은 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 코드 세분화를 수행하는 방법의 일례를 설명하기 위한 흐름도이다.

도 7은 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 코드 유사도 값을 이용하여, 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 방법의 일례를 설명하기 위한 흐름도이다.

도 8은 본 개시의 몇몇 실시예에 따른 본 발명과 종래의 기술과의 비교를 위한 도면이다.

도 9는 본 개시내용의 실시예들이 구현될 수 있는 예시적인 컴퓨팅 환경에 대한 일반적인 개략도를 도시한다.

발명을 실시하기 위한 구체적인 내용

[0036] 다양한 실시예들 및/또는 양상들이 이제 도면들을 참조하여 개시된다. 하기 설명에서는 설명을 목적으로, 하나 이상의 양상들의 전반적 이해를 돕기 위해 다수의 구체적인 세부사항들이 개시된다. 그러나, 이러한 양상(들)은 이러한 구체적인 세부사항들 없이도 실행될 수 있다는 점 또한 본 개시의 기술 분야에서 통상의 지식을 가진 자에게 감지될 수 있을 것이다. 이후의 기재 및 첨부된 도면들은 하나 이상의 양상들의 특정한 예시적인 양상들을 상세하게 기술한다. 하지만, 이러한 양상들은 예시적인 것이고 다양한 양상들의 원리들에서의 다양한 방법들 중 일부가 이용될 수 있으며, 기술되는 설명들은 그러한 양상들 및 그들의 균등물들을 모두 포함하고자 하는 의도이다. 구체적으로, 본 명세서에서 사용되는 "실시예", "예", "양상", "예시" 등은 기술되는 임의의 양상 또는 설계가 다른 양상 또는 설계들보다 양호하다거나, 이점이 있는 것으로 해석되지 않을 수도 있다.

[0037] 이하, 도면 부호에 관계없이 동일하거나 유사한 구성요소는 동일한 참조 번호를 부여하고 이에 대한 중복되는 설명은 생략한다. 또한, 본 명세서에 개시된 실시예를 설명함에 있어서 관련된 공지 기술에 대한 구체적인 설명이 본 명세서에 개시된 실시예의 요지를 흐릴 수 있다고 판단되는 경우 그 상세한 설명을 생략한다. 또한, 첨부된 도면은 본 명세서에 개시된 실시예를 쉽게 이해할 수 있도록 하기 위한 것일 뿐, 첨부된 도면에 의해 본 명세서에 개시된 기술적 사상이 제한되지 않는다.

[0038] 비록 제 1, 제 2 등이 다양한 소자나 구성요소들을 서술하기 위해서 사용되나, 이들 소자나 구성요소들은 이들 용어에 의해 제한되지 않음은 물론이다. 이들 용어들은 단지 하나의 소자나 구성요소를 다른 소자나 구성요소와

구별하기 위하여 사용하는 것이다. 따라서, 이하에서 언급되는 제 1 소자나 구성요소는 본 발명의 기술적 사상 내에서 제 2 소자나 구성요소 일 수도 있음은 물론이다.

[0039] 다른 정의가 없다면, 본 명세서에서 사용되는 모든 용어(기술 및 과학적 용어를 포함)는 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에게 공통적으로 이해될 수 있는 의미로 사용될 수 있을 것이다. 또 일반적으로 사용되는 사전에 정의되어 있는 용어들은 명백하게 특별히 정의되어 있지 않는 한 이상적으로 또는 과도하게 해석되지 않는다.

[0040] 더불어, 용어 "또는"은 배타적 "또는"이 아니라 내포적 "또는"을 의미하는 것으로 의도된다. 즉, 달리 특정되지 않거나 문맥상 명확하지 않은 경우에, "X는 A 또는 B를 이용한다"는 자연적인 내포적 치환 중 하나를 의미하는 것으로 의도된다. 즉, X가 A를 이용하거나; X가 B를 이용하거나; 또는 X가 A 및 B 모두를 이용하는 경우, "X는 A 또는 B를 이용한다"가 이들 경우들 어느 것으로도 적용될 수 있다. 또한, 본 명세서에 사용된 "및/또는"이라는 용어는 열거된 관련 아이템들 중 하나 이상의 아이템의 가능한 모든 조합을 지칭하고 포함하는 것으로 이해되어야 한다.

[0041] 또한, "포함한다" 및/또는 "포함하는"이라는 용어는, 해당 특징 및/또는 구성요소가 존재함을 의미하지만, 하나 이상의 다른 특징, 구성요소 및/또는 이들의 그룹의 존재 또는 추가를 배제하지 않는 것으로 이해되어야 한다. 또한, 달리 특정되지 않거나 단수 형태를 지시하는 것으로 문맥상 명확하지 않은 경우에, 본 명세서와 청구범위에서 단수는 일반적으로 "하나 또는 그 이상"을 의미하는 것으로 해석되어야 한다.

[0042] 더불어, 본 명세서에서 사용되는 용어 "정보" 및 "데이터"는 종종 서로 상호교환 가능하도록 사용될 수 있다.

[0043] 어떤 구성요소가 다른 구성요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어" 있다거나 "직접 접속되어"있다고 언급된 때에는, 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다.

[0044] 이하의 설명에서 사용되는 구성요소에 대한 접미사 "모듈" 및 "부"는 명세서 작성의 용이함만이 고려되어 부여되거나 혼용되는 것으로서 그 자체로 서로 구별되는 의미 또는 역할을 갖는 것은 아니다.

[0045] 본 개시의 목적 및 효과, 그리고 그것들을 달성하기 위한 기술적 구성들은 첨부되는 도면과 함께 상세하게 후술되어 있는 실시예들을 참조하면 명확해질 것이다. 본 개시를 설명하는데 있어서 공지 기능 또는 구성에 대한 구체적인 설명이 본 개시의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략할 것이다. 그리고 후술되는 용어들은 본 개시에서의 기능을 고려하여 정의된 용어들로써 이는 사용자, 운용자의 의도 또는 관례 등에 따라 달라질 수 있다.

[0046] 그러나 본 개시는 이하에서 개시되는 실시예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 수 있다. 단지 본 실시예들은 본 개시가 완전하도록 하고, 본 개시가 속하는 기술분야에서 통상의 지식을 가진 자에게 개시의 범주를 완전하게 알려주기 위해 제공되는 것이며, 본 개시는 청구항의 범주에 의해 정의될 뿐이다. 그러므로 그 정의는 본 명세서 전반에 걸친 내용을 토대로 내려져야 할 것이다.

[0048] 본 개시에서, 컴퓨팅 장치의 프로세서는 복수의 오픈소스 소프트웨어(Open Source Software: OSS) 각각에 중복 제거를 수행하여, 컴포넌트 데이터베이스를 구축할 수 있다. 그리고, 프로세서는 구축된 컴포넌트 데이터베이스를 이용하여, 구성요소를 식별하고자 하는 타겟 소프트웨어의 구성요소를 식별할 수 있다. 이하, 본 개시에 따른 오픈소스 소프트웨어 구성요소 식별을 위한 방법에 대해 설명한다.

[0050] 도 1은 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치의 일례를 설명하기 위한 블록 구성도이다.

[0051] 도 1을 참조하면, 컴퓨팅 장치(100)는 프로세서(110) 및 컴포넌트 데이터베이스부(120)를 포함할 수 있다. 다만, 상술한 구성요소들은 컴퓨팅 장치(100)를 구현하는데 있어서 필수적인 것은 아니어서, 컴퓨팅 장치(100)는 위에서 열거된 구성요소들 보다 많거나, 또는 적은 구성요소들을 가질 수 있다.

[0052] 컴퓨팅 장치(100)는 예를 들어, 마이크로프로세서, 메인프레임 컴퓨터, 디지털 프로세서, 휴대용 디바이스 또는 디바이스 제어기 등과 같은 임의의 타입의 컴퓨터 시스템 또는 컴퓨터 디바이스를 포함할 수 있다.

[0053] 한편, 프로세서(110)는 통상적으로 컴퓨팅 장치(100)의 전반적인 동작을 처리할 수 있다. 프로세서(110)는 서버의 구성요소들을 통해 입력 또는 출력되는 신호, 데이터, 정보 등을 처리하거나 컴포넌트 데이터베이스부(120)에 저장된 응용 프로그램을 구동함으로써, 사용자에게 적절한 정보 또는 기능을 제공 또는 처리할 수 있다.

[0054] 일례로, 프로세서(110)는 복수의 오픈소스 소프트웨어 각각에 중복제거를 수행할 수 있다. 그리고, 프로세서(110)는 중복제거를 수행하여 구축된 컴포넌트 데이터베이스를 컴포넌트 데이터베이스부(120)에 저장할 수 있다. 여기서, 중복제거는 복수의 오픈소스 소프트웨어에 공통적으로 포함되어 있는 함수의 중복을 제거하는 것일 수 있다. 그리고, 프로세서(110)는 구축된 컴포넌트 데이터베이스를 이용하여, 타겟 소프트웨어의 구성요소를 식별할 수 있다. 여기서, 타겟 소프트웨어는 구성요소를 식별하고자 하는 오픈소스 소프트웨어일 수 있다. 이하, 본 개시에 따른 프로세서(110)가 수행하는 동작에 대한 내용은 도 2 내지 도 8을 통해 설명한다.

[0055] 한편, 컴포넌트 데이터베이스부(120)는 메모리 및/또는 영구저장매체를 포함할 수 있다. 메모리는 플래시 메모리 타입(flash memory type), 하드디스크 타입(hard disk type), 멀티미디어 카드 마이크로 타입(multimedia card micro type), 카드 타입의 메모리(예를 들어 SD 또는 XD 메모리 등), 램(Random Access Memory, RAM), SRAM(Static Random Access Memory), 롬(Read-Only Memory, ROM), EEPROM(Electrically Erasable Programmable Read-Only Memory), PROM(Programmable Read-Only Memory), 자기 메모리, 자기 디스크, 광디스크 중 적어도 하나의 타입의 저장매체를 포함할 수 있다. 다만, 이에 한정되는 것은 아니다.

[0057] 도 2는 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 오픈소스 소프트웨어의 구성요소를 식별하기 위한 방법의 일례를 설명하기 위한 흐름도이다.

[0058] 도 2를 참조하면, 컴퓨팅 장치(100)의 프로세서(110)는 복수의 오픈소스 소프트웨어 각각에 중복제거를 수행하여, 컴포넌트 데이터베이스를 구축할 수 있다(S100).

[0059] 일례로, 컴포넌트 데이터베이스부(120)에 저장된 복수의 오픈소스 소프트웨어 중 적어도 하나의 오픈소스 소프트웨어는 버전별에 따라 복수개로 존재할 수 있다. 이때, 상기 오픈소스 소프트웨어는 버전이 업데이트될 때, 소스코드가 매번 새로 작성되지 않았을 수 있다. 따라서, 상기 오픈소스 소프트웨어의 각 버전에는 공통적인 코드 부분이 존재할 수 있다. 이러한 공통적인 코드 부분들은 프로세서(110)가 오픈소스 소프트웨어의 구성요소를 탐지할 때 중복되어 매칭에 사용될 수 있다. 이 경우, 프로세서(110)가 구성요소를 탐지하는 동작에 있어서, 탐지 동작이 오래 걸리는 문제가 발생하거나 또는 오류가 발생될 수 있다. 따라서, 프로세서(110)는 복수의 오픈소스 소프트웨어 각각에 중복제거를 수행할 수 있다.

[0060] 구체적으로, 프로세서(110)는 복수의 오픈소스 소프트웨어 중 제 1 오픈소스 소프트웨어에 대한 중복 제거를 수행하는 경우, 제 1 오픈소스 소프트웨어의 하나 이상의 버전에 등장하는 적어도 하나의 함수를 인식할 수 있다. 그리고, 프로세서는 상기 적어도 하나의 함수 및 상기 적어도 하나의 함수가 등장한 버전에 기초하여, 제 1 오픈소스 소프트웨어에 대한 제 1 딕셔너리(dictionary) 자료구조를 생성할 수 있다. 여기서, 딕셔너리 자료구조는 Key-Value 형태의 값을 저장할 수 있는 자료구조일 수 있다. 이때, 제 1 딕셔너리 자료구조는 함수가 등장하는 버전의 개수에 따라 서로 다른 그룹으로 구분될 수 있다. 이 경우, 하나 이상의 버전에 등장한 적어도 하나의 함수 각각은 하나의 딕셔너리 자료구조로 저장될 수 있다. 다만, 이에 한정되는 것은 아니다. 이하, 본 개시에 따른 프로세서(110)가 중복제거를 수행하는 방법은 도 3 및 도 4를 통해 좀 더 설명한다.

[0061] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 컴포넌트 데이터베이스를 이용하여, 타겟 소프트웨어의 구성요소를 식별할 수 있다(S200).

[0062] 구체적으로, 프로세서(110)는 컴포넌트 데이터베이스부(120)에 저장된 복수의 오픈소스 소프트웨어 중 타겟 소프트웨어와 코드 유사도 값이 기 설정된 값 이상인 적어도 하나의 오픈소스 소프트웨어를 인식할 수 있다. 여기서, 코드 유사도 값은 타겟 소스 코드에 포함된 함수와 복수의 오픈소스 소프트웨어 각각의 고유한 코드 부분의 함수에 공통적으로 포함되는 함수의 개수 및 복수의 오픈소스 소프트웨어 각각의 고유한 코드 부분의 함수의 개수에 기초하여, 결정되는 값일 수 있다. 일례로, 프로세서(110)는 코드 유사도가 10% 이상인 적어도 하나의 오픈소스 소프트웨어를 타겟 소프트웨어의 구성요소라고 판단할 수 있다. 이하, 본 개시에 따른 코드 유사도 값에 대한 내용은 도 7을 통해 좀 더 설명한다.

[0064] 도 3은 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 컴포넌트 데이터베이스를 구축하는 방법의 일례를 설명하기 위한 흐름도이다. 도 4는 본 개시의 몇몇 실시예에 따른 딕셔너리 자료구조를 설명하기 위한 도면이다.

[0065] 도 3을 참조하면, 컴퓨팅 장치(100)의 프로세서(110)는 복수의 오픈소스 소프트웨어 중 제 1 오픈소스 소프트웨어에 대한 중복 제거를 수행할 수 있다. 이때, 프로세서(110)는 제 1 오픈소스 소프트웨어의 적어도 하나의 버전 각각에 등장하는 적어도 하나의 함수를 인식할 수 있다(S110).

[0066] 구체적으로, 프로세서(110)는 적어도 하나의 함수 각각의 해시 값을 키(key)로 설정하고, 적어도 하나의 함수

각각이 등장하는 하나 이상의 버전을 키에 대한 값(value)으로 설정할 수 있다. 그리고, 프로세서(110)는 상기 해시 값 및 상기 하나 이상의 버전을 제 1 오픈소스 소프트웨어에 대한 제 1 딕셔너리(dictionary) 자료구조에 저장할 수 있다(S120).

[0067] 예를 들어, 도 4를 참조하면, 프로세서(110)는 제 1 오픈소스 소프트웨어(200)에 등장한 i 함수의 해시 값(210)을 키로 설정할 수 있다. 또한, 프로세서(110)는 i 함수가 등장하는 하나 이상의 버전(220)을 값으로 설정할 수 있다. 이 경우, i 함수의 해시 값(210) 및 i 함수가 등장한 하나 이상의 버전(220)이 제 1 딕셔너리 자료구조(230)에 저장될 수 있다. 다만, 이에 한정되는 것은 아니다.

[0068] 한편, 본 개시의 몇몇 실시예에 따르면, 제 1 딕셔너리 자료구조(230)는, 함수가 등장하는 버전의 개수에 따라 서로 다른 그룹으로 구분될 수 있다.

[0069] 일례로, 제 1 그룹(240)을 참조하면, 제 1 그룹(240)에는 하나의 버전에서만 등장한 함수를 포함하는 적어도 하나의 딕셔너리 자료구조가 구분되어 있을 수 있다.

[0070] 다른 일례로, 제 2 그룹(250)을 참조하면, 제 2 그룹(250)에는 두개의 버전에서 등장한 함수를 포함하는 적어도 하나의 딕셔너리 자료구조가 구분되어 있을 수 있다.

[0071] 즉, 프로세서(110)는 제 1 오픈소스 소프트웨어(200)에 포함된 적어도 하나의 함수 각각을 함수가 등장하는 버전의 개수에 따라 서로 다른 그룹으로 구분할 수 있다. 따라서, 프로세서(110)는 제 1 오픈소스 소프트웨어(200)의 여러 버전에 포함된 적어도 하나의 함수가 한번만 등장하도록 컴포넌트 데이터베이스를 구축함으로써, 중복 제거를 수행할 수 있다. 다만, 이에 한정되는 것은 아니다.

[0072] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 컴포넌트 데이터베이스부(120)에 저장된 모든 오픈소스 소프트웨어 각각에 상술한 동작을 수행하여, 컴포넌트 데이터베이스를 구축할 수 있다. 다만, 이에 한정되는 것은 아니다.

[0073] 상술한 구성에 따르면, 오픈소스 소프트웨어의 각 버전에 공통적으로 등장하는 함수들은 중복 제거가 수행되어 적어도 하나의 딕셔너리 자료구조로 구분되어 있을 수 있다. 따라서, 컴퓨팅 장치(100)의 프로세서(110)가 타겟 소프트웨어의 구성요소를 식별하는데 있어서 연산에 소요되는 시간이 감소되고 오류의 발생도 줄어들 수 있다. 또한, 중복 제거를 통해 보다 많은 수의 오픈소스 소프트웨어를 수집할 수 있기 때문에, 확장성 및 성능에 있어서 우수한 데이터베이스가 구축될 수 있다.

[0074] 한편, 본 개시의 몇몇 실시예에 따르면, 컴퓨팅 장치(100)의 프로세서(110)는 구축된 컴포넌트 데이터베이스를 이용하여, 타겟 소프트웨어의 구성요소를 식별할 수 있다. 이하, 본 개시에 따른 프로세서(110)가 타겟 소프트웨어의 구성요소를 식별하는 방법의 일례를 설명한다.

[0076] 도 5는 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 타겟 소프트웨어의 구성요소를 식별하는 방법의 일례를 설명하기 위한 흐름도이다.

[0077] 도 5를 참조하면, 컴퓨팅 장치(100)의 프로세서(110)는 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를 고유한 코드 부분 및 빌려온 코드 부분으로 세분화할 수 있다(S210). 여기서, 고유한 코드 부분은 다른 오픈소스 소프트웨어에서 작성되지 않았다고 인식한 코드일 수 있다. 그리고, 빌려온 코드 부분은 다른 오픈소스 소프트웨어에서 먼저 작성된 코드라고 인식한 코드일 수 있다.

[0078] 구체적으로, 프로세서(110)는 복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 오픈소스 소프트웨어 및 상기 오픈소스 소프트웨어와 비교 분석할 오픈소스 소프트웨어를 선정할 수 있다. 그리고, 프로세서(110)는 코드 세분화를 수행할 오픈소스 소프트웨어 및 비교 분석할 오픈소스 소프트웨어에 공통적으로 등장한 하나 이상의 함수를 인식할 수 있다. 이때, 프로세서(110)는 상기 하나 이상의 함수가 비교 분석할 오픈소스 소프트웨어에서 먼저 등장했다고 인식할 수 있다. 이 경우, 프로세서(110)는 상기 하나 이상의 함수는 코드 세분화를 수행할 오픈소스 소프트웨어에 있어서, 빌려온 코드 부분이라고 인식할 수 있다. 만약, 프로세서(110)는 상기 하나 이상의 함수가 비교 분석할 오픈소스 소프트웨어에서 늦게 등장했다고 인식한 경우, 상기 하나 이상의 함수는 코드 세분화를 수행할 오픈소스 소프트웨어에 있어서, 고유한 코드 부분이라고 인식할 수 있다. 이하, 본 개시에 따른 코드 세분화를 수행하는 방법은 도 6을 통해 좀 더 설명한다.

[0079] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 타겟 소프트웨어에 포함된 타겟 소스 코드 및 복수의 오픈소스 소프트웨어 각각에 포함된 고유한 코드 부분을 비교할 수 있다. 그리고, 프로세서(110)는 복수의 오픈소스 소프트웨어 중 타겟 소프트웨어 작성 시 사용된 적어도 하나의 제 2 오픈소스 소프트웨어를 추출할 수 있다(S220).

[0080] 구체적으로, 프로세서(110)는 타겟 소프트웨어에 포함된 타겟 소스 코드와 복수의 오픈소스 소프트웨어 각각에 포함된 고유한 코드 부분을 비교하여, 코드 유사도 값을 인식할 수 있다. 그리고, 프로세서(110)는 코드 유사도 값이 기 설정된 값 이상인 적어도 하나의 제 2 오픈소스 소프트웨어를 추출할 수 있다. 일례로, 프로세서(110)는 타겟 소프트웨어와 적어도 하나의 제 2 오픈소스 소프트웨어의 코드 유사도 값이 10% 이상인 경우, 적어도 하나의 제 2 오픈소스 소프트웨어를 추출할 수 있다. 다만, 이에 한정되는 것은 아니다. 이하, 본 개시에 따른 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 방법은 도 7을 통해 좀 더 설명한다.

[0081] 상술한 구성에 따르면, 컴퓨팅 장치(100)의 프로세서(110)는 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를 고유한 코드 부분 및 빌려온 코드 부분으로 세분화할 수 있다. 이 경우, 프로세서(110)는 복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드의 고유한 코드 부분 만을 이용하여, 타겟 소프트웨어 작성 시 사용된 적어도 하나의 제 2 오픈소스 소프트웨어를 추출할 수 있다. 따라서, 프로세서(110)가 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는데 소요되는 시간이 줄어들 수 있다.

[0082] 한편, 본 개시의 몇몇 실시예에 따르면, 컴퓨팅 장치(100)의 프로세서(110)는 복수의 오픈소스 소프트웨어 각각에 코드 세분화를 수행함에 있어서, 빌려온 코드 부분에 포함되는 함수라고 인식한 함수는 제거할 수 있다. 이하, 본 개시에 따른 프로세서(110)가 코드 세분화를 수행하는 방법의 일례를 설명한다.

[0084] 도 6은 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 코드 세분화를 수행하는 방법의 일례를 설명하기 위한 흐름도이다.

[0085] 도 6을 참조하면, 컴퓨팅 장치(100)의 프로세서(110)는 복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수를 인식할 수 있다(S211). 여기서, 제 3 오픈소스 소프트웨어는 중복 제거가 수행되어 컴포넌트 데이터베이스부(120)에 저장된 오픈소스 소프트웨어일 수 있다.

[0086] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 복수의 오픈소스 소프트웨어 중 제 3 오픈소스 소프트웨어와 비교분석할 제 4 오픈소스 소프트웨어를 선정할 수 있다(S212). 여기서, 제 4 오픈소스 소프트웨어는 중복 제거가 수행되어 컴포넌트 데이터베이스부(120)에 저장된 적어도 하나의 오픈소스 소프트웨어일 수 있다.

[0087] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 제 4 오픈소스 소프트웨어에 포함된 제 2 소스 코드에 등장하는 하나 이상의 제 2 함수를 인식할 수 있다(S213).

[0088] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 하나 이상의 제 1 함수와 하나 이상의 제 2 함수를 비교하여 동일하다고 인식되는 하나 이상의 공통 함수를 추출할 수 있다(S214).

[0089] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 하나 이상의 공통 함수 중 제 4 오픈소스 소프트웨어에 먼저 등장한 하나 이상의 제 3 함수를 인식할 수 있다(S215).

[0090] 구체적으로, 컴포넌트 데이터베이스부(120)에는 복수의 오픈소스 소프트웨어 각각의 릴리즈(release) 정보가 저장되어 있을 수 있다. 여기서, 릴리즈 정보는 오픈소스 소프트웨어가 배포되는 날짜에 대한 정보일 수 있다. 따라서, 프로세서(110)는 제 3 오픈소스 소프트웨어 및 제 4 오픈소스 소프트웨어 중 어느 오픈소스 소프트웨어가 먼저 배포되었는지를 인식할 수 있다. 그리고, 프로세서(110)는 제 4 오픈소스 소프트웨어가 먼저 배포되었다고 인식한 경우, 하나 이상의 공통 함수 중 제 4 소프트웨어 포함된 하나 이상의 제 3 함수를 인식할 수 있다. 다만, 이에 한정되는 것은 아니다.

[0091] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 컴포넌트 데이터베이스 내의 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에서 하나 이상의 제 3 함수를 제거할 수 있다(S216).

[0092] 이 경우, 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에는 제 3 오픈소스 소프트웨어의 고유한 코드 부분만이 남을 수 있다. 다만, 이에 한정되는 것은 아니다.

[0093] 한편, 본 개시의 몇몇 실시예에 따르면, 컴퓨팅 장치(100)의 프로세서(110)는 하나 이상의 제 2 함수의 제 1 개수와 하나 이상의 제 3 함수의 제 2 개수를 인식할 수 있다. 그리고, 프로세서(110)는 제 1 개수와 제 2 개수에 기초하여 산출된 값이 기 설정된 값 이상인 경우, 제 2 딕셔너리 자료구조에 포함된 하나 이상의 제 1 함수에서 상기 하나 이상의 제 3 함수를 제거할 수 있다.

[0094] 구체적으로, 프로세서(110)는 하나 이상의 제 2 함수의 제 1 개수를 하나 이상의 제 3 함수의 제 2 개수로 나눌 수 있다. 그리고, 프로세서는 제 1 개수를 제 2 개수로 나눈 값이, 0.1 이상이라고 인식한 경우, 코드 세분화를

수행할 제 3 오픈소스 소프트웨어의 빌려온 부분에 제 2 함수가 포함되었다고 인식할 수 있다. 제 3 오픈소스 소프트웨어의 코드베이스의 10% 이상이 제 4 오픈소스 소프트웨어에 포함되어 있다면, 재사용 가능성이 있다고 판단할 수 있기 때문이다. 이 경우, 프로세서는 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수에서, 제 3 함수를 제거할 수 있다. 다만, 이에 한정되는 것은 아니다.

[0095] 상술한 구성에 따르면, 컴포넌트 데이터베이스부(120) 내의 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에는 제 3 오픈소스 소프트웨어의 고유한 코드 부분만이 남을 수 있다. 따라서, 컴퓨팅 장치(100)의 프로세서(110)가 타겟 소프트웨어의 구성요소를 식별하는데 있어서 연산에 소요되는 시간이 감소되고 또한 오류의 발생도 줄어들 수 있다.

[0096] 한편, 본 개시의 몇몇 실시예에 따르면, 컴퓨팅 장치(100)의 프로세서(110)는 코드 유사도 값을 이용하여, 적어도 하나의 제 2 오픈소스 소프트웨어를 추출할 수 있다. 이하, 본 개시에 따른 프로세서(110)가 코드 유사도 값을 이용하여, 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 방법의 일례를 설명한다.

[0098] 도 7은 본 개시의 몇몇 실시예에 따른 컴퓨팅 장치가 코드 유사도 값을 이용하여, 적어도 하나의 제 2 오픈소스 소프트웨어를 추출하는 방법의 일례를 설명하기 위한 흐름도이다.

[0099] 도 7을 참조하면, 컴퓨팅 장치(100)의 프로세서(110)는 타겟 소스 코드에 포함된 함수와 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 고유한 코드 부분의 함수에 공통적으로 포함되는 함수의 제 3 개수를 인식할 수 있다. 그리고, 프로세서(110)는 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 고유한 코드 부분의 함수의 제 4 개수를 인식할 수 있다. 이 경우, 프로세서(110)는 제 3 개수 및 제 4 개수에 기초하여 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각과 타겟 소프트웨어 사이의 코드 유사도 값 각각을 인식할 수 있다(S221).

[0100] 여기서, 코드 유사도 값은 아래의 수학식과 같이 정의될 수 있다.

수학식 1

[0102] $$\Phi = \frac{|T \cap S|}{|S|}$$

[0103] 여기서, Φ는 코드유사도 값일 수 있다. 그리고, T는 타겟 소프트웨어의 함수일 수 있다. 또한, S는 코드 세분화를 수행한 복수의 오픈소스 소프트웨어의 고유한 코드 부분의 함수일 수 있다.

[0104] 구체적으로, 프로세서(110)는 제 3 개수를 제 4 개수로 나눈 값을 코드 유사도 값이라고 인식할 수 있다. 다만, 이에 한정되는 것은 아니다.

[0105] 한편, 컴퓨팅 장치(100)의 프로세서(110)는 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 중 코드 유사도 값이 기 설정된 값 이상인 적어도 하나의 제 2 오픈소스 소프트웨어를 추출할 수 있다(S222). 여기서, 기 설정된 값은 10%일 수 있다. 다만, 이에 한정되는 것은 아니다. 그리고, 프로세서(110)는 추출한 제 2 오픈소스 소프트웨어를 타겟 소프트웨어의 구성요소라고 판단할 수 있다. 다만, 이에 한정되는 것은 아니다.

[0107] 도 8은 본 개시의 몇몇 실시예에 따른 본 발명과 종래의 기술과의 비교를 위한 도면이다.

[0108] 본 발명과 관련된 종래의 대표적인 기술로는 Ruian Duan 외 4인의 “Identifying Open-Source License Violation and 1-day Security Risk at Large Scale.”(OSSPolice) 기술이 있을 수 있다.

[0109] 도 8을 참조하면, 분석 레벨에 있어서, 종래의 OSSPolice는 소프트웨어 바이너리 레벨로 분석을 수행할 수 있었다. 반면, 본 발명은 소프트웨어 소스코드 레벨로 분석을 수행할 수 있다.

[0110] 한편, 활용 세분성 유닛에 있어서, 종래의 OSSPolice는 피처(feature) 단위(예를 들어, 문자열 또는 함수 이름 등)를 활용할 수 있었다. 다만, 피처 단위로 오픈소스 소프트웨어 구성요소 탐지를 수행하는 경우, 피처의 수정된 재사용은 고려되지 않을 수 있다. 예를 들어, 해당 피처가 사용되지 않았거나(부분재사용), 또는 오픈소스 소프트웨어 내 소스파일들이 기존과 다른 위치에서 재사용될 수도 있다. 이 경우, 정확도가 떨어질 수 있다. 반면, 본 발명은 일부 피처가 아닌 소스코드 내의 전체 함수를 기본 단위로 한할 수 있다. 따라서 소스코드의 일부분만 재사용하더라도 해당 부분만을 고려한 탐지가 가능할 수 있다. 또한, 탐지과정에서 소스코드의 구조정보(예를 들어, 파일의 경로)를 활용하지 않기 때문에, 구조 변경 여부에 관계없이 구성요소 탐지가 가능할 수 있

다.

[0111] 한편, 확장성의 비교를 위해 OSSPolice의 경우 30GB의 오픈소스 소프트웨어 집합(C와 C++ 소프트웨어만 고려한 전체 약20억 줄의 라인)을 데이터셋으로 선정했다. 이 경우, OSSPolice를 통해 각 소프트웨어로부터 피처를 추출하는데 평균 1,000초가 소요되었으며, 평균적으로 피처의 개수에 비례한 탐지 시간(예를 들어, 피처가 10,000개면 평균 탐지 시간 역시 약 10,000초가 소요됨)이 소요될 수 있었다. 반면, 본 발명의 경우, 13TB의 오픈소스 소프트웨어 집합(전체 약 800억줄의 라인)을 초기 데이터셋으로 선정하여 종래에 비해 약 40배 이상 확장할 수 있었다. 또한 함수를 추출하고 전처리 과정을 거치는 데에 소프트웨어당 평균 320초가 소요되었고, 타겟 소프트웨어에서 오픈소스 소프트웨어 구성요소를 탐지할 때 평균 100초 이하의 시간이 소요될 수 있었다. 따라서, OSSPolice의 확장성이 준수하다고 평가되는 경우, 본 발명의 확장성은 높다고 평가될 수 있다.

[0112] 한편, 상술한 테스트를 통해 정확도를 비교해본 결과, OSSPolice의 경우 구조가 수정된 경우를 배제한 C/C++ OSS 구성요소 탐지 정확도가 82%일 수 있었다. 반면, 본 발명의 경우, 코드 및 구조의 수정을 모두 고려하고도 91% 이상의 구성요소 탐지 정확도를 달성할 수 있었다. 따라서, OSSPolice의 정확도가 준수하다고 평가되는 경우, 본 발명의 확장성은 높다고 평가될 수 있다.

[0113] 한편, 상술한 테스트를 통해 재현율을 비교해볼 수 있었다. 여기서, 재현율은 타겟 소프트웨어의 구성요소가 맞음에도 불구하고 찾지 못한 오픈소스 소프트웨어의 비율을 나타내기 위한 값을 의미할 수 있다. 즉, 재현율이 높을수록 찾지 못한 오픈소스 소프트웨어의 비율이 적은 것을 의미할 수 있다. 재현율에 있어서, OSSPolice의 경우, 87% 수준의 결과가 도출될 수 있었다. 반면, 본 발명의 경우, 94%의 결과를 도출할 수 있었다.

[0114] 상술한 바와 같이 종래의 대표적인 기술인 OSSPolice와 본 발명을 비교하여 실험해본 결과, 본 발명은 종래 기술에 비해 효과적으로 오픈소스 소프트웨어 구성요소 탐지가 수행될 수 있었다.

[0116] 도 9는 본 개시내용의 실시예들이 구현될 수 있는 예시적인 컴퓨팅 환경에 대한 일반적인 개략도를 도시한다.

[0117] 본 개시내용이 일반적으로 하나 이상의 컴퓨터 상에서 실행될 수 있는 컴퓨터 실행가능 명령어와 관련하여 전술되었지만, 당업자라면 본 개시내용 기타 프로그램 모듈들과 결합되어 및/또는 하드웨어와 소프트웨어의 조합으로서 구현될 수 있다는 것을 잘 알 것이다.

[0118] 일반적으로, 본 명세서에서의 모듈은 특정의 태스크를 수행하거나 특정의 추상 데이터 유형을 구현하는 루틴, 프로시저, 프로그램, 컴포넌트, 데이터 구조, 기타 등등을 포함한다. 또한, 당업자라면 본 개시의 방법이 단일-프로세서 또는 멀티프로세서 컴퓨터 시스템, 미니컴퓨터, 메인프레임 컴퓨터는 물론 퍼스널 컴퓨터, 핸드헬드 컴퓨팅 장치, 마이크로프로세서-기반 또는 프로그램가능 가전 제품, 기타 등등(이들 각각은 하나 이상의 연관된 장치와 연결되어 동작할 수 있음)을 비롯한 다른 컴퓨터 시스템 구성으로 실시될 수 있다는 것을 잘 알 것이다.

[0119] 본 개시의 설명된 실시예들은 또한 어떤 태스크들이 통신 네트워크를 통해 연결되어 있는 원격 처리 장치들에 의해 수행되는 분산 컴퓨팅 환경에서 실시될 수 있다. 분산 컴퓨팅 환경에서, 프로그램 모듈은 로컬 및 원격 메모리 저장 장치 둘다에 위치할 수 있다.

[0120] 컴퓨터는 통상적으로 다양한컴퓨터 판독가능 매체를 포함한다. 컴퓨터에 의해 액세스 가능한 매체 로서, 휘발성 및 비휘발성 매체, 일시적(transitory) 및 비일시적(non-transitory) 매체, 이동식 및 비-이동식 매체를 포함한다. 제한이 아닌 예로서, 컴퓨터 판독가능 매체는 컴퓨터 판독가능 저장 매체 및 컴퓨터 판독가능 전송 매체를 포함할 수 있다.

[0121] 컴퓨터 판독가능 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터와 같은 정보를 저장하는 임의의 방법 또는 기술로 구현되는 휘발성 및 비휘발성 매체, 일시적 및 비-일시적 매체, 이동식 및 비이동식 매체를 포함한다. 컴퓨터 판독가능 저장 매체는 RAM, ROM, EEPROM, 플래시 메모리 또는 기타 메모리 기술, CD-ROM, DVD(digital video disk) 또는 기타 광 디스크 저장 장치, 자기 카세트, 자기 테이프, 자기 디스크 저장 장치 또는 기타 자기 저장 장치, 또는 컴퓨터에 의해 액세스될 수 있고 원하는 정보를 저장하는 데 사용될 수 있는 임의의 기타 매체를 포함하지만, 이에 한정되지 않는다.

[0122] 컴퓨터 판독가능 전송 매체는 통상적으로 반송파(carrier wave) 또는 기타 전송 메커니즘(transport mechanism)과 같은 피변조 데이터 신호(modulated data signal)에 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터등을 구현하고 모든 정보 전달 매체를 포함한다. 피변조 데이터 신호라는 용어는 신호 내에 정보를 인코딩하도록 그 신호의 특성들 중 하나 이상을 설정 또는 변경시킨 신호를 의미한다. 제한이 아닌 예로서, 컴퓨터 판독가능 전송 매체는 유선 네트워크 또는 직접 배선 접속(direct-wired connection)과 같은 유

선 매체, 그리고 음향, RF, 적외선, 기타 무선 매체와 같은 무선 매체를 포함한다. 상술된 매체들 중 임의의 것의 조합도 역시 컴퓨터 판독가능 전송 매체의 범위 안에 포함되는 것으로 한다.

[0123] 컴퓨터(1102)를 포함하는 본 개시의 여러가지 측면들을 구현하는 예시적인 환경(1100)이 나타내어져 있으며, 컴퓨터(1102)는 처리 장치(1104), 시스템 메모리(1106) 및 시스템 버스(1108)를 포함한다. 시스템 버스(1108)는 시스템 메모리(1106)(이에 한정되지 않음)를 비롯한 시스템 컴포넌트들을 처리 장치(1104)에 연결시킨다. 처리 장치(1104)는 다양한 상용 프로세서들 중 임의의 프로세서일 수 있다. 듀얼 프로세서 및 기타 멀티프로세서 아키텍처도 역시 처리 장치(1104)로서 이용될 수 있다.

[0124] 시스템 버스(1108)는 메모리 버스, 주변장치 버스, 및 다양한 상용 버스 아키텍처 중 임의의 것을 사용하는 로컬 버스에 추가적으로 상호 연결될 수 있는 몇 가지 유형의 버스 구조 중 임의의 것일 수 있다. 시스템 메모리(1106)는 판독 전용 메모리(ROM)(1110) 및 랜덤 액세스 메모리(RAM)(1112)를 포함한다. 기본 입/출력 시스템(BIOS)은 ROM, EPROM, EEPROM 등의 비휘발성 메모리(1110)에 저장되며, 이 BIOS는 시동 중과 같은 때에 컴퓨터(1102) 내의 구성요소들 간에 정보를 전송하는 일을 돕는 기본적인 루틴을 포함한다. RAM(1112)은 또한 데이터를 캐싱하기 위한 정적 RAM 등의 고속 RAM을 포함할 수 있다.

[0125] 컴퓨터(1102)는 또한 내장형 하드 디스크 드라이브(HDD)(1114)(예를 들어, EIDE, SATA)-이 내장형 하드 디스크 드라이브(1114)는 또한 적당한 섀시(도시 생략) 내에서 외장형 용도로 구성될 수 있음-, 자기 플로피 디스크 드라이브(FDD)(1116)(예를 들어, 이동식 디스켓(1118)으로부터 판독을 하거나 그에 기록을 하기 위한 것임), 및 광 디스크 드라이브(1120)(예를 들어, CD-ROM 디스크(1122)를 판독하거나 DVD 등의 기타 고용량 광 매체로부터 판독을 하거나 그에 기록을 하기 위한 것임)를 포함한다. 하드 디스크 드라이브(1114), 자기 디스크 드라이브(1116) 및 광 디스크 드라이브(1120)는 각각 하드 디스크 드라이브 인터페이스(1124), 자기 디스크 드라이브 인터페이스(1126) 및 광 드라이브 인터페이스(1128)에 의해 시스템 버스(1108)에 연결될 수 있다. 외장형 드라이브 구현을 위한 인터페이스(1124)는 예를 들어, USB(Universal Serial Bus) 및 IEEE 1394 인터페이스 기술 중 적어도 하나 또는 그 둘 다를 포함한다.

[0126] 이들 드라이브 및 그와 연관된 컴퓨터 판독가능 매체는 데이터, 데이터 구조, 컴퓨터 실행가능 명령어, 기타 등등의 비휘발성 저장을 제공한다. 컴퓨터(1102)의 경우, 드라이브 및 매체는 임의의 데이터를 적당한 디지털 형식으로 저장하는 것에 대응한다. 상기에서의 컴퓨터 판독가능 저장 매체에 대한 설명이 HDD, 이동식 자기 디스크, 및 CD 또는 DVD 등의 이동식 광 매체를 언급하고 있지만, 당업자라면 집 드라이브(zip drive), 자기 카세트, 플래쉬 메모리 카드, 카트리지, 기타 등등의 컴퓨터에 의해 판독가능한 다른 유형의 저장 매체도 역시 예시적인 운영 환경에서 사용될 수 있으며 또 임의의 이러한 매체가 본 개시의 방법들을 수행하기 위한 컴퓨터 실행가능 명령어를 포함할 수 있다는 것을 잘 알 것이다.

[0127] 운영 체제(1130), 하나 이상의 애플리케이션 프로그램(1132), 기타 프로그램 모듈(1134) 및 프로그램 데이터(1136)를 비롯한 다수의 프로그램 모듈이 드라이브 및 RAM(1112)에 저장될 수 있다. 운영 체제, 애플리케이션, 모듈 및/또는 데이터의 전부 또는 그 일부분이 또한 RAM(1112)에 캐싱될 수 있다. 본 개시가 여러가지 상업적으로 이용가능한 운영 체제 또는 운영 체제들의 조합에서 구현될 수 있다는 것을 잘 알 것이다.

[0128] 사용자는 하나 이상의 유선/무선 입력 장치, 예를 들어, 키보드(1138) 및 마우스(1140) 등의 포인팅 장치를 통해 컴퓨터(1102)에 명령 및 정보를 입력할 수 있다. 기타 입력 장치(도시 생략)로는 마이크, IR 리모콘, 조이스틱, 게임 패드, 스타일러스 펜, 터치 스크린, 기타 등등이 있을 수 있다. 이들 및 기타 입력 장치가 종종 시스템 버스(1108)에 연결되어 있는 입력 장치 인터페이스(1142)를 통해 처리 장치(1104)에 연결되지만, 병렬 포트, IEEE 1394 직렬 포트, 게임 포트, USB 포트, IR 인터페이스, 기타 등등의 기타 인터페이스에 의해 연결될 수 있다.

[0129] 모니터(1144) 또는 다른 유형의 디스플레이 장치도 역시 비디오 어댑터(1146) 등의 인터페이스를 통해 시스템 버스(1108)에 연결된다. 모니터(1144)에 부가하여, 컴퓨터는 일반적으로 스피커, 프린터, 기타 등등의 기타 주변 출력 장치(도시 생략)를 포함한다.

[0130] 컴퓨터(1102)는 유선 및/또는 무선 통신을 통한 원격 컴퓨터(들)(1148) 등의 하나 이상의 원격 컴퓨터로의 논리적 연결을 사용하여 네트워크화된 환경에서 동작할 수 있다. 원격 컴퓨터(들)(1148)는 워크스테이션, 서버 컴퓨터, 라우터, 퍼스널 컴퓨터, 휴대용 컴퓨터, 마이크로프로세서-기반 오락 기기, 피어 장치 또는 기타 통상의 네트워크 노드일 수 있으며, 일반적으로 컴퓨터(1102)에 대해 기술된 구성요소들 중 다수 또는 그 전부를 포함하지만, 간략함을 위해, 메모리 저장 장치(1150)만이 도시되어 있다. 도시되어 있는 논리적 연결은 근거리 통신망

(LAN)(1152) 및/또는 더 큰 네트워크, 예를 들어, 원거리 통신망(WAN)(1154)에의 유선/무선 연결을 포함한다. 이러한 LAN 및 WAN 네트워킹 환경은 사무실 및 회사에서 일반적인 것이며, 인트라넷 등의 전사적 컴퓨터 네트워크(enterprise-wide computer network)를 용이하게 해주며, 이들 모두는 전세계 컴퓨터 네트워크, 예를 들어, 인터넷에 연결될 수 있다.

[0131] LAN 네트워킹 환경에서 사용될 때, 컴퓨터(1102)는 유선 및/또는 무선 통신 네트워크 인터페이스 또는 어댑터(1156)를 통해 로컬 네트워크(1152)에 연결된다. 어댑터(1156)는 LAN(1152)에의 유선 또는 무선 통신을 용이하게 해줄 수 있으며, 이 LAN(1152)은 또한 무선 어댑터(1156)와 통신하기 위해 그에 설치되어 있는 무선 액세스 포인트를 포함하고 있다. WAN 네트워킹 환경에서 사용될 때, 컴퓨터(1102)는 모뎀(1158)을 포함할 수 있거나, WAN(1154) 상의 통신 서버에 연결되거나, 또는 인터넷을 통하는 등, WAN(1154)을 통해 통신을 설정하는 기타 수단을 갖는다. 내장형 또는 외장형 및 유선 또는 무선 장치일 수 있는 모뎀(1158)은 직렬 포트 인터페이스(1142)를 통해 시스템 버스(1108)에 연결된다. 네트워크화된 환경에서, 컴퓨터(1102)에 대해 설명된 프로그램 모듈들 또는 그의 일부분이 원격 메모리/저장 장치(1150)에 저장될 수 있다. 도시된 네트워크 연결이 예시적인 것이며 컴퓨터들 사이에 통신 링크를 설정하는 기타 수단이 사용될 수 있다는 것을 잘 알 것이다.

[0132] 컴퓨터(1102)는 무선 통신으로 배치되어 동작하는 임의의 무선 장치 또는 개체, 예를 들어, 프린터, 스캐너, 데스크톱 및/또는 휴대용 컴퓨터, PDA(portable data assistant), 통신 위성, 무선 검출가능 태그와 연관된 임의의 장비 또는 장소, 및 전화와 통신을 하는 동작을 한다. 이것은 적어도 Wi-Fi 및 블루투스 무선 기술을 포함한다. 따라서, 통신은 종래의 네트워크에서와 같이 미리 정의된 구조이거나 단순하게 적어도 2개의 장치 사이의 애드혹 통신(ad hoc communication)일 수 있다.

[0133] Wi-Fi(Wireless Fidelity)는 유선 없이도 인터넷 등으로의 연결을 가능하게 해준다. Wi-Fi는 이러한 장치, 예를 들어, 컴퓨터가 실내에서 및 실외에서, 즉 기지국의 통화권 내의 아무 곳에서나 데이터를 전송 및 수신할 수 있게 해주는 셀 전화와 같은 무선 기술이다. Wi-Fi 네트워크는 안전하고 신뢰성 있으며 고속인 무선 연결을 제공하기 위해 IEEE 802.11(a,b,g, 기타)이라고 하는 무선 기술을 사용한다. 컴퓨터를 서로에, 인터넷에 및 유선 네트워크(IEEE 802.3 또는 이더넷을 사용함)에 연결시키기 위해 Wi-Fi가 사용될 수 있다. Wi-Fi 네트워크는 비인가 2.4 및 5 GHz 무선 대역에서, 예를 들어, 11Mbps(802.11a) 또는 54 Mbps(802.11b) 데이터 레이트로 동작하거나, 양 대역(듀얼 대역)을 포함하는 제품에서 동작할 수 있다.

[0134] 본 개시의 기술 분야에서 통상의 지식을 가진 자는 여기에 개시된 실시예들과 관련하여 설명된 다양한 예시적인 논리 블록들, 모듈들, 프로세서들, 수단들, 회로들 및 알고리즘 단계들이 전자 하드웨어, (편의를 위해, 여기에서 "소프트웨어"로 지칭되는) 다양한 형태들의 프로그램 또는 설계 코드 또는 이들 모두의 결합에 의해 구현될 수 있다는 것을 이해할 것이다. 하드웨어 및 소프트웨어의 이러한 상호 호환성을 명확하게 설명하기 위해, 다양한 예시적인 컴포넌트들, 블록들, 모듈들, 회로들 및 단계들이 이들의 기능과 관련하여 위에서 일반적으로 설명되었다. 이러한 기능이 하드웨어 또는 소프트웨어로서 구현되는지 여부는 특정한 애플리케이션 및 전체 시스템에 대하여 부과되는 설계 제약들에 따라 좌우된다. 본 개시의 기술 분야에서 통상의 지식을 가진 자는 각각의 특정한 애플리케이션에 대하여 다양한 방식들로 설명된 기능을 구현할 수 있으나, 이러한 구현 결정들은 본 개시의 범위를 벗어나는 것으로 해석되어서는 안 될 것이다.

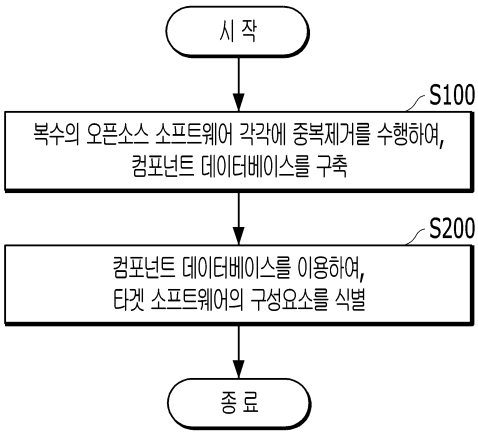
[0135] 여기서 제시된 다양한 실시예들은 방법, 장치, 또는 표준 프로그래밍 및/또는 엔지니어링 기술을 사용한 제조 물품(article)으로 구현될 수 있다. 용어 "제조 물품"은 임의의 컴퓨터-판독가능 장치로부터 액세스 가능한 컴퓨터 프로그램 또는 매체(media)를 포함한다. 예를 들어, 컴퓨터-판독가능 저장 매체는 자기 저장 장치(예를 들면, 하드 디스크, 플로피 디스크, 자기 스트립, 등), 광학 디스크(예를 들면, CD, DVD, 등), 스마트 카드, 및 플래쉬 메모리 장치(예를 들면, EEPROM, 카드, 스틱, 키 드라이브, 등)를 포함하지만, 이들로 제한되는 것은 아니다. 용어 "기계-판독가능 매체"는 명령(들) 및/또는 데이터를 저장, 보유, 및/또는 전달할 수 있는 무선 채널 및 다양한 다른 매체를 포함하지만, 이들로 제한되는 것은 아니다.

[0137] 제시된 실시예들에 대한 설명은 임의의 본 개시의 기술 분야에서 통상의 지식을 가진 자가 본 개시를 이용하거나 또는 실시할 수 있도록 제공된다. 이러한 실시예들에 대한 다양한 변형들은 본 개시의 기술 분야에서 통상의 지식을 가진 자에게 명백할 것이며, 여기에 정의된 일반적인 원리들은 본 개시의 범위를 벗어남이 없이 다른 실시예들에 적용될 수 있다. 그리하여, 본 개시는 여기에 제시된 실시예들로 한정되는 것이 아니라, 여기에 제시된 원리들 및 신규한 특징들과 일관되는 최광의의 범위에서 해석되어야 할 것이다.

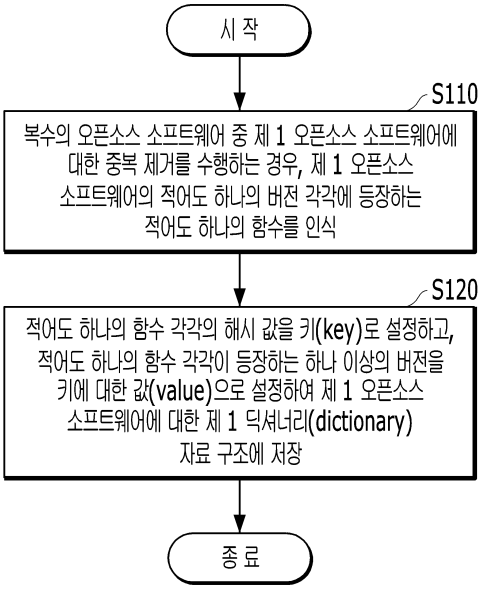
도면

도면1

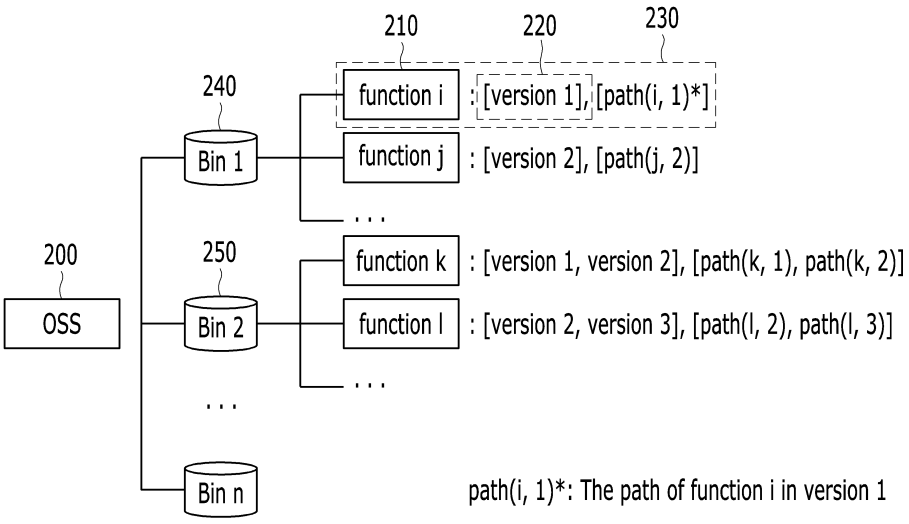
도면2



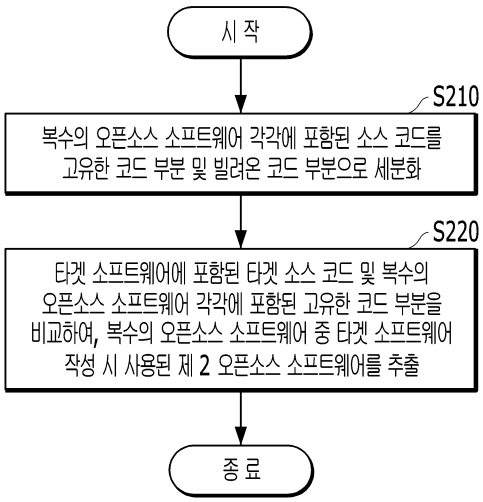
도면3



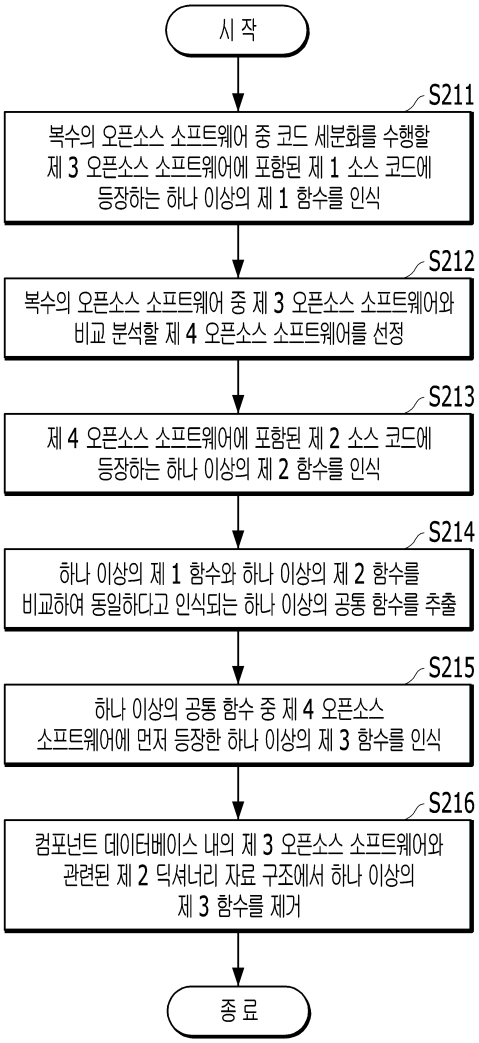
도면4


210
220
230
240
function i
: [version 1], [path(i, 1)*]
Bin 1
function j
: [version 2], [path(j, 2)]
200
250
function k
: [version 1, version 2], [path(k, 1), path(k, 2)]
OSS
Bin 2
function l
: [version 2, version 3], [path(l, 2), path(l, 3)]
Bin n
path(i, 1)*: The path of function i in version 1

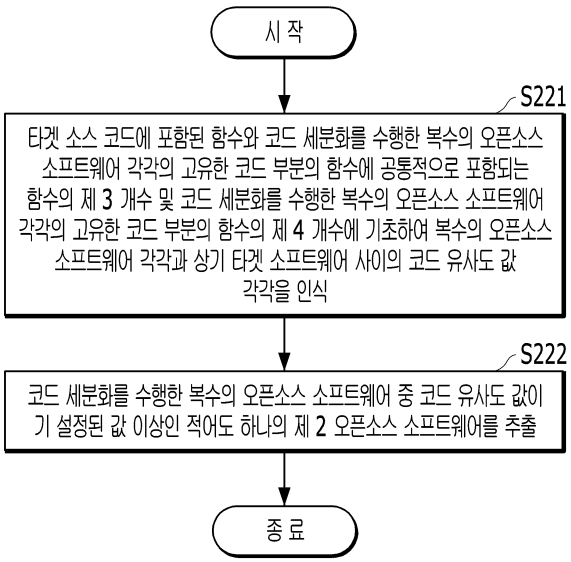
도면5


시 작
S210
복수의 오픈소스 소프트웨어 각각에 포함된 소스 코드를
고유한 코드 부분 및 빌려온 코드 부분으로 세분화
S220
타겟 소프트웨어에 포함된 타겟 소스 코드 및 복수의
오픈소스 소프트웨어 각각에 포함된 고유한 코드 부분을
비교하여, 복수의 오픈소스 소프트웨어 중 타겟 소프트웨어
작성 시 사용된 제 2 오픈소스 소프트웨어를 추출
종 료

도면6


시 작
S211
복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수를 인식
S212
복수의 오픈소스 소프트웨어 중 제 3 오픈소스 소프트웨어와 비교 분석할 제 4 오픈소스 소프트웨어를 선정
S213
제 4 오픈소스 소프트웨어에 포함된 제 2 소스 코드에 등장하는 하나 이상의 제 2 함수를 인식
S214
하나 이상의 제 1 함수와 하나 이상의 제 2 함수를 비교하여 동일하다고 인식되는 하나 이상의 공통 함수를 추출
S215
하나 이상의 공통 함수 중 제 4 오픈소스 소프트웨어에 먼저 등장한 하나 이상의 제 3 함수를 인식
S216
컴포넌트 데이터베이스 내의 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료 구조에서 하나 이상의 제 3 함수를 제거
종 료

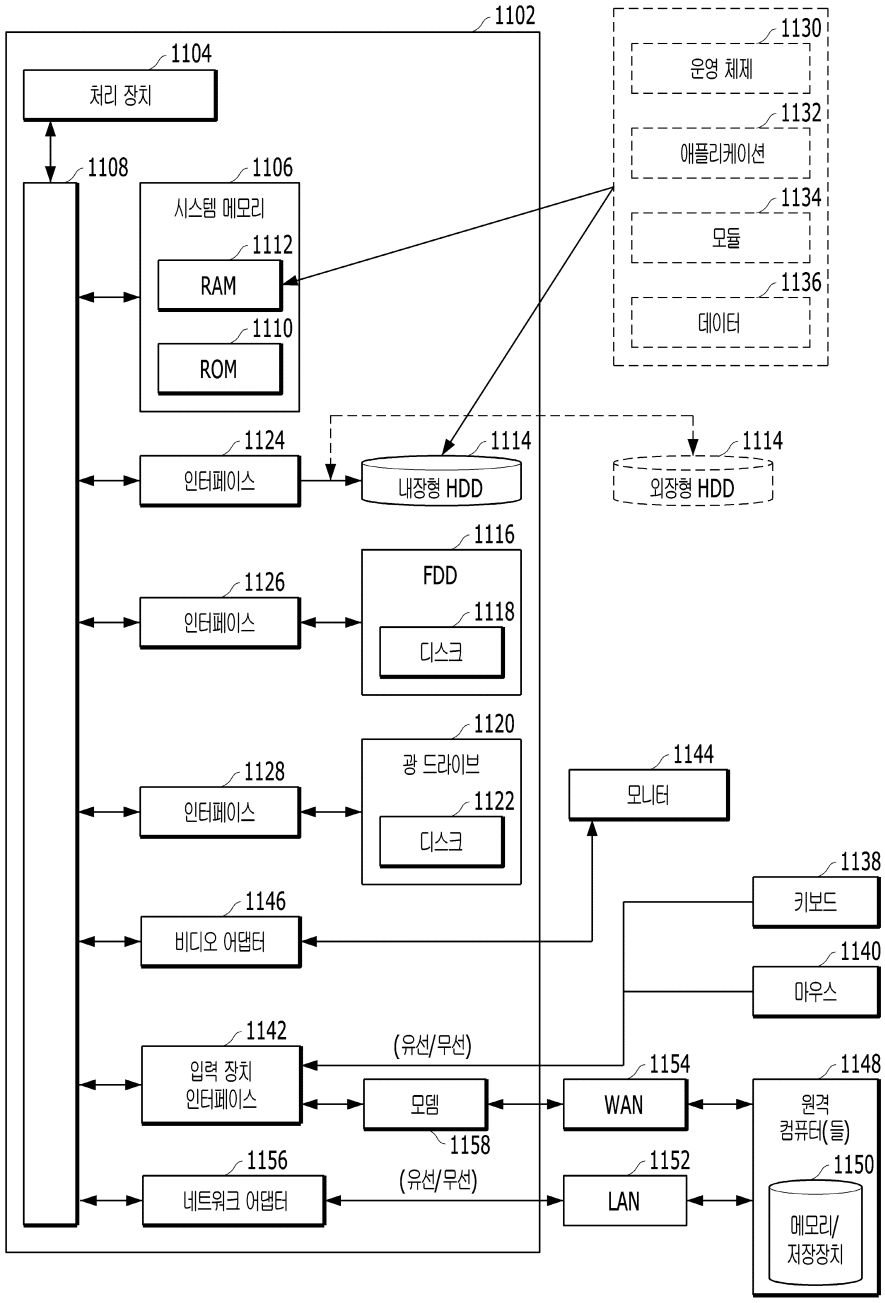
도면7


시 작
S221
타겟 소스 코드에 포함된 함수와 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 고유한 코드 부분의 함수에 공통적으로 포함되는 함수의 제 3 개수 및 코드 세분화를 수행한 복수의 오픈소스 소프트웨어 각각의 고유한 코드 부분의 함수의 제 4 개수에 기초하여 복수의 오픈소스 소프트웨어 각각과 상기 타겟 소프트웨어 사이의 코드 유사도 값 각각을 인식
S222
코드 세분화를 수행한 복수의 오픈소스 소프트웨어 중 코드 유사도 값이 기 설정된 값 이상인 적어도 하나의 제 2 오픈소스 소프트웨어를 추출
종 료

도면8

	OSSPolice	본 발명
분석 레벨	소프트웨어 바이너리	소프트웨어 소스코드
활용 세분성 유닛	피쳐(feature)단위 (예: 문자열, 함수 이름 등)	함수 단위
확장성	준수함	높음
정확도	준수함 (수정된 구성요소 탐지 불가)	높음 (수정된 구성요소 탐지 가능)
정량적 비교	1. 2B LoC 데이터 셋 활용 2. OSS하나당 피쳐 추출 시간 평균 1,000초 3. 탐지 시간 ≒OSS의 피쳐 개수 4. 재현율 87%, 정밀도 82% (수정된 구성요소 배제)	1. 80B LoC 데이터 셋 활용 (약 40배) 2. OSS하나당 함수 추출 시간 평균 320초 3. 탐지 시간 ≤ 100초 4. 재현율 94%, 정밀도 91% (수정된 구성요소 고려)

도면9



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 청구항 13

【변경전】

제 12 항에 있어서,

상기 프로세서는,

상기 복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수를 인식하고,

상기 복수의 오픈소스 소프트웨어 중 상기 제 3 오픈소스 소프트웨어와 비교 분석할 제 4 오픈소스 소프트웨어

를 선정하고,

상기 제 4 오픈소스 소프트웨어에 포함된 제 2 소스 코드에 등장하는 하나 이상의 제 2 함수를 인식하고,

상기 하나 이상의 제 1 함수와 상기 하나 이상의 제 2 함수를 비교하여 동일하다고 인식되는 하나 이상의 공통 함수를 추출하고,

상기 하나 이상의 공통 함수 중 상기 제 4 오픈소스 소프트웨어에 먼저 등장한 하나 이상의 제 3 함수를 인식하고,

상기 컴포넌트 데이터베이스부 내의 상기 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에서 상기 하나 이상의 제 3 함수를 제거하는,

를 포함하는,

오픈소스 소프트웨어 구성요소 식별을 위한 장치.

【변경후】

제 12 항에 있어서,

상기 프로세서는,

상기 복수의 오픈소스 소프트웨어 중 코드 세분화를 수행할 제 3 오픈소스 소프트웨어에 포함된 제 1 소스 코드에 등장하는 하나 이상의 제 1 함수를 인식하고,

상기 복수의 오픈소스 소프트웨어 중 상기 제 3 오픈소스 소프트웨어와 비교 분석할 제 4 오픈소스 소프트웨어를 선정하고,

상기 제 4 오픈소스 소프트웨어에 포함된 제 2 소스 코드에 등장하는 하나 이상의 제 2 함수를 인식하고,

상기 하나 이상의 제 1 함수와 상기 하나 이상의 제 2 함수를 비교하여 동일하다고 인식되는 하나 이상의 공통 함수를 추출하고,

상기 하나 이상의 공통 함수 중 상기 제 4 오픈소스 소프트웨어에 먼저 등장한 하나 이상의 제 3 함수를 인식하고,

상기 컴포넌트 데이터베이스부 내의 상기 제 3 오픈소스 소프트웨어와 관련된 제 2 딕셔너리 자료구조에서 상기 하나 이상의 제 3 함수를 제거하는 오픈소스 소프트웨어 구성요소 식별을 위한 장치.