관 인 생 략

출 원 번 호 통 지 서

출 원 일 자	2024.08.23
특 기 사 항	심사청구(유) 공개신청(무) 참조번호(W24P0177KR)
출 원 번 호	10-2024-0113268 (접수번호 1-1-2024-0921530-11) (DAS접근코드DBBA)
출원인 명칭	고려대학교 산학협력단(2-2004-017068-0)
대리인 성명	특허법인 위솔(9-2022-100001-9)
발명자 성명	이희조 나윤종
발명의 명칭	재사용된 오픈소스 소프트웨어의 컴포넌트 간 의존성 판별 방법 및 장치

특 허 청 장

<< 안내 >>

1. 귀하의 출원은 위와 같이 정상적으로 접수되었으며, 이후의 심사 진행상황은 출원번호를 이용하여 특허로 홈페이지(www.patent.go.kr)에서 확인하실 수 있습니다.
2. 출원에 따른 수수료는 접수일로부터 다음날까지 동봉된 납입영수증에 성명, 납부자번호 등을 기재하여 가까운 은행 또는 우체국에 납부하여야 합니다.
 ※ 납부자번호 : 0131(기관코드) + 접수번호
3. 귀하의 주소, 연락처 등의 변경사항이 있을 경우, 즉시 [특허고객번호 정보변경(경정), 정정신고서]를 제출하여야 출원 이후의 각종 통지서를 정상적으로 받을 수 있습니다.
4. 기타 심사 절차(제도)에 관한 사항은 특허청 홈페이지를 참고하시거나 특허고객상담센터(☎ 1544-8080)에 문의하여 주시기 바랍니다.
 ※ 심사제도 안내 : https://www.kipo.go.kr-지식재산제도

【서지사항】

【서류명】 특허출원서

【참조번호】 W24P0177KR

【출원구분】 특허출원

【출원인】

【명칭】 고려대학교산학협력단

【특허고객번호】 2-2004-017068-0

【대리인】

【명칭】 특허법인 위솔

【대리인번호】 9-2022-100001-9

【지정된변리사】 나강은, 김경용, 강현모, 편진호

【발명의 국문명칭】 재사용된 오픈소스 소프트웨어의 컴포넌트 간 의존성 판별 방법 및 장치

【발명의 영문명칭】 METHOD AND APPARATUS FOR DETERMINING DEPENDENCIES BETWEEN COMPONENTS OF REUSED OPEN SOURCE SOFTWARE

【발명자】

【성명】 이희조

【성명의 영문표기】 Heejo Lee

【주민등록번호】

【우편번호】

【주소】

【발명자】

【성명】 나윤종

【성명의 영문표기】 NA Yoon Jong

【주민등록번호】

【우편번호】

【주소】

【출원언어】 국어

【심사청구】 청구

【공지예외적용대상증명서류의 내용】

【공개형태】 논문 (https://dl.acm.org/doi/10.1145/3597503.3639209)

【공개일자】 2024.04.12

【공지예외적용대상증명서류의 내용】

【공개형태】 발표 학회 (2024 International Conference on Software Engineering)
(https://conf.researchr.org/home/icse-2024)

【공개일자】 2024.04.19

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 1711193841

【과제번호】 2022-0-00277-002

【부처명】 과학기술정보통신부

【과제관리(전문)기관명】 정보통신기획평가원

【연구사업명】 정보보호핵심원천기술개발(R&D)

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

【과제수행기관명】 고려대학교산학협력단

【연구기간】 2024.01.01 ~ 2024.12.31

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 1711193663

【과제번호】 2020-0-01819-004

【부처명】 과학기술정보통신부

【과제관리(전문)기관명】 정보통신기획평가원

【연구사업명】 정보통신방송혁신인재양성

【연구과제명】 ICT명품인재양성(고려대학교)

【과제수행기관명】 고려대학교산학협력단

【연구기간】 2024.01.01 ~ 2024.12.31

【이 발명을 지원한 국가연구개발사업】

【과제고유번호】 1711193387

【과제번호】 2022-0-01198-002

【부처명】 과학기술정보통신부

【과제관리(전문)기관명】 정보통신기획평가원

【연구사업명】 정보통신방송혁신인재양성

【연구과제명】 융합보안대학원(고려대학교)

【과제수행기관명】 고려대학교산학협력단

【연구기간】 2024.01.01 ~ 2024.12.31

【취지】 위와 같이 특허청장에게 제출합니다.

대리인 특허법인 위솔 (서명 또는 인)

【수수료】

【출원료】 0 면 원

【가산출원료】 42 면 원

【우선권주장료】 0 건 원

【심사청구료】 11 항 원

【합계】 원

【감면사유】 전담조직(50%감면)[1]

【감면후 수수료】 원

【첨부서류】 1.공지예외적용대상(신규성상실의예외, 출원시의특례)규정을 적용받기 위한 증명서류_1통 2.공지예외적용대상(신규성상실의예외, 출원시의특례)규정을 적용받기 위한 증명서류_1통 3.기타첨부서류[개별위임장]_1통

1 : 공지예외적용대상(신규성상실의예외,_출원시의특례)규정을_적용받기_위한_증명서류

PDF 파일 첨부

2 : 공지예외적용대상(신규성상실의예외,_출원시의특례)규정을_적용받기_위한_증명서류

PDF 파일 첨부

3 : 기타첨부서류

PDF 파일 첨부

【발명의 설명】

【발명의 명칭】

재사용된 오픈소스 소프트웨어의 컴포넌트 간 의존성 판별 방법 및 장치 {METHOD AND APPARATUS FOR DETERMINING DEPENDENCIES BETWEEN COMPONENTS OF REUSED OPEN SOURCE SOFTWARE}

【기술분야】

【0001】본 발명은 주어진 소스코드를 라이브러리로 재사용 가능한 단위인 모듈로 재구성하여 재사용된 오픈소스 소프트웨어의 컴포넌트 간 의존성 관계를 판별하는 기술에 관한 것이다.

【발명의 배경이 되는 기술】

【0002】소프트웨어 산업의 발전과 함께, 재사용 가능한 오픈소스 소프트웨어의 컴포넌트 활용이 활발해지고 있다. 컴포넌트란 소프트웨어 개발에서 재사용 가능한 코드의 단위이다.

【0003】즉, 컴포넌트는 하나의 완성된 기능을 가진 작은 프로그램 단위로 다른 소프트웨어에서 재사용될 수 있다. 예를 들어, 웹사이트에서 로그인 기능을 구현하는 코드를 하나의 컴포넌트로 만들면, 이 코드를 다른 웹사이트에서도 그대로 사용할 수 있다. 이러한 컴포넌트를 통해 개발자는 시간을 절약하고, 코드의 일관성을 유지하며, 오류 발생을 줄일 수 있다.

【0004】한편, 컴포넌트 간 의존성을 분석하는 것은 소프트웨어의 안정성, 성능, 보안, 생산성을 높이는 데 필수적이다. 컴포넌트 간 의존성을 기초로 버그 수정과 업데이트가 용이해지고, 불필요한 의존성을 제거해 성능을 최적화하고 소프트웨어 공급망의 투명성을 확보할 수 있기 때문이다.

【0005】그러나, 컴포넌트 간의 의존성은 코드에 대한 정보를 특정하는 메타데이터가 있지 않는 한 직접적으로 분석하는 것이 어렵기 때문에, 컴포넌트 간의 의존성을 분석하는 문제는 중요한 도전 과제로 남아 있다. 특히, 여러 프로젝트에서 다양한 형태로 재사용되는 컴포넌트들 간의 의존성을 정확하게 분석하는 것은 매우 복잡한 작업이다.

【0006】기존의 컴포넌트 간의 의존성을 분석하는 기술들은 식별 가능한 컴포넌트를 식별하고 그 안에서 컴포넌트 간의 의존성 관계를 분석하는 데 초점을 맞추고 있다. 그러나 기존의 기술은 '재사용이 불분명한 파일' 또는 '중복된 컴포넌트인지에 대한 여부'를 식별하지 못하여, 이러한 컴포넌트에 종속된 의존 관계에 대해 낮은 정확도를 보인다는 한계가 있다.

【0007】여기서, '재사용이 불분명한 파일'이란 특정 파일이 오픈소스 라이브러리나 컴포넌트에서 온 것인지, 아니면 프로젝트 내에서 독립적으로 작성된 것인지 등 어떤 출처에서 왔는지 명확하게 식별되지 않는 파일을 의미한다. 이는 컴포넌트가 정확하게 어떤 프로젝트에서 재사용되었는지를 파악하지 못하게 하며, 결과적으로 의존성 분석의 정확도를 저하시킨다. 이러한 파일들은 소스코드 내에서 재사용되었을 가능성이 있지만, 그 출처를 명확히 알 수 없기 때문에 정확한 의존

성 분석이 어렵다. 이러한 파일들은 의존성 분석 과정에서 오탐이나 미탐을 유발할 수 있다.

【0008】또한, '중복된 컴포넌트인지에 대한 여부'란, 소스코드에 포함된 여러 컴포넌트가 동일한 컴포넌트에 해당하는 것인지 별개의 컴포넌트에 해당하는 것인지에 대한 구분을 의미한다. 예를 들어, 동일한 라이브러리 파일이 여러 소스코드 프로젝트에 포함되어 있는 경우, 이를 각각 다른 컴포넌트로 인식하면 의존성 그래프에 중복된 노드가 생성될 수 있다. 이처럼, 컴포넌트 간의 의존성 분석 과정에서 복수 개의 동일한 컴포넌트를 중복으로 인식하지 못하게 되면 의존성 분석 결과에 오차가 발생하게 되어, 의존성에 기반한 소프트웨어의 활용에서 오류가 발생할 수 있다.

【0009】이와 같은 문제들은 소프트웨어 공급망의 투명성을 저해하고, 잠재적인 보안 취약점을 식별하는 데 어려움을 초래한다. 이에 따라, 소스코드에 대한 메타데이터 없는 상황에서도 재사용이 불분명한 파일이나 중복된 컴포넌트인지에 대한 여부를 구분하여, 보다 정확하고 신뢰성을 지닌 의존성 분석 방법이 필요한 실정이다.

【선행기술문헌】

【특허문헌】

【0010】(특허문헌 0001) 대한민국 등록특허공보 제10-2104322호

【발명의 내용】

【해결하고자 하는 과제】

【0011】본 발명은 소스코드 중 '재사용이 불분명한 파일'과 '중복된 컴포넌트'를 정확하게 식별하여, 재사용 오픈소스 소프트웨어의 컴포넌트 간 의존성을 보다 정확하게 분석하는 방법을 제안하고자 한다.

【0012】이를 위해, 본 발명은 함수의 선언을 가진 헤더 파일과 해당 선언 함수의 정의를 가진 소스 파일을 하나의 모듈로 구성하는 모듈 단위 기반의 분석을 통해 재사용이 불분명한 파일을 정확하게 식별하고자 한다.

【0013】또한, 본 발명은 코드 재사용과 라이브러리 재사용의 두 가지 의존성 패턴을 모두 고려한 통합적인 의존성 분석을 제공하고자 한다. 이를 위해, 모듈 간의 코드 재사용 의존성과 라이브러리 재사용 의존성을 각각 분석하고, 이를 통합한 의존성 그래프를 생성함으로써 보다 정확하고 신뢰성 있는 의존성 분석을 가능하게 하고자 한다.

【0014】또한, 본 발명은 유사한 컴포넌트 간의 디렉토리 경로, 다른 컴포넌트에 의한 의존성, 중복된 파일의 보유 여부를 바탕으로 중복된 컴포넌트를 효과적으로 식별하고자 한다.

【0015】이를 통해, 본 발명은 높은 정확도와 정밀도로 소프트웨어 컴포넌트 간의 의존성을 분석할 수 있게 하며, 소프트웨어 개발 및 유지보수 과정에서 발생할 수 있는 다양한 문제를 예방하고, 보다 안전한 소프트웨어 환경을 구축하는 데

기여하고자 한다.

【0016】한편, 본 발명의 기술적 과제들은 이상에서 언급한 기술적 과제들로 제한되지 않으며, 언급되지 않은 또 다른 기술적 과제들은 아래의 기재로부터 통상의 기술자에게 명확하게 이해될 수 있을 것이다.

【과제의 해결 수단】

【0017】일 실시예에 따른 프로세서에 의해 동작하는 의존성 판별 장치가 수행하는 방법에 있어서, 소정 함수에 대한 헤더파일 및 소스파일을 포함하는 모듈 단위로 타겟 소스코드를 구분하는 동작; 각 모듈에 포함된 함수의 컴포넌트명을 판별하는 동작; 각 모듈에 포함된 컴포넌트명의 개수를 기초로 모듈이 재사용한 제1 컴포넌트 및 모듈에 포함된 함수가 의존하는 제2 컴포넌트를 판별하는 동작; 상기 제1 컴포넌트 또는 상기 제2 컴포넌트가 임포트하는 헤더파일을 기초로 상기 모듈이 의존하는 라이브러리를 판별하는 동작; 및 상기 제2 컴포넌트에 대한 상기 제1 컴포넌트의 의존성 및 상기 라이브러리에 대한 상기 제1 컴포넌트 또는 상기 제2 컴포넌트의 의존성을 기초로 컴포넌트 간 의존성 그래프를 생성하는 동작을 포함할 수 있다.

【0018】또한, 상기 헤더파일은 상기 소정 함수를 선언하는 선언문을 포함하고, 상기 소스파일은 상기 헤더파일에서 선언된 함수를 정의하는 정의문을 포함할 수 있다.

【0019】또한, 상기 컴포넌트명을 판별하는 동작은 함수에 대한 컴포넌트명을 저장하는 컴포넌트 데이터베이스를 상기 각 모듈에 포함된 함수와 비교하여 상기 각 모듈에 포함된 함수에 컴포넌트명을 매핑시키는 동작을 포함할 수 있다.

【0020】또한, 상기 제1 컴포넌트 및 제2 컴포넌트를 판별하는 동작은 제1 모듈에 포함된 컴포넌트명 중 포함된 개수가 가장 많은 컴포넌트명을 선별하는 동작; 및 상기 제1 모듈의 헤더파일이 상기 개수가 가장 많은 컴포넌트명의 원본 프로젝트에 포함되어 있는 경우, 상기 개수가 가장 많이 포함된 컴포넌트명을 상기 제1 컴포넌트로 결정하는 동작을 포함할 수 있다.

【0021】또한, 상기 제1 컴포넌트 및 제2 컴포넌트를 판별하는 동작은 상기 제1 모듈의 헤더파일이 상기 개수가 가장 많은 컴포넌트명의 원본 프로젝트에 포함되어 있지 않은 경우, 상기 제1 모듈에 포함된 컴포넌트명 중 포함된 개수가 많은 순서로 컴포넌트명의 원본 프로젝트에 상기 제1 모듈의 헤더파일의 포함 여부를 검색하는 동작; 및 상기 제1 모듈에 포함된 컴포넌트명 중 상기 제1 모듈의 헤더파일이 검색된 컴포넌트명을 상기 제1 컴포넌트로 결정하는 동작을 포함할 수 있다.

【0022】또한, 상기 제1 컴포넌트 및 제2 컴포넌트를 판별하는 동작은 상기 제1 컴포넌트로 결정하는 동작 이후에, 상기 제1 모듈에 포함된 컴포넌트명 중 상기 제1 컴포넌트를 뺀 컴포넌트명을 상기 제2 컴포넌트로 결정하는 동작을 포함할 수 있다.

【0023】또한, 상기 제1 컴포넌트 및 제2 컴포넌트를 판별하는 동작은 모듈에 포함된 함수 중 컴포넌트명이 판별되지 않은 함수가 존재하는 경우, 상기 컴포넌트명이 판별되지 않은 함수의 컴포넌트를 제1 컴포넌트로 설정하는 동작을 포함할 수 있다.

【0024】또한, 상기 라이브러리를 판별하는 동작은 상기 모듈이 임포트하는 제1 헤더파일과 상기 제1 컴포넌트 또는 상기 제2 컴포넌트가 임포트하는 제2 헤더파일이 상이한 경우, 상기 모듈이 상기 제2 헤더파일의 라이브러리를 의존하는 것으로 판별하는 동작을 포함할 수 있다.

【0025】또한, 상기 의존성 그래프를 생성하는 동작은 상기 제1 컴포넌트 또는 제2 컴포넌트 중 컴포넌트명이 동일한 중복명 컴포넌트가 복수 개가 있는 경우, 상기 중복명 컴포넌트 각각의 디렉토리 경로, 상기 중복명 컴포넌트 각각에 대한 다른 컴포넌트에 의한 의존성 및 상기 중복명 컴포넌트 각각이 중복된 파일을 포함하는지의 여부 중 적어도 하나를 기초로, 상기 중복명 컴포넌트의 병합 여부를 결정하여 컴포넌트 간 의존성 그래프를 생성하는 동작을 포함할 수 있다.

【0026】또한, 상기 의존성 그래프를 생성하는 동작은 상기 중복명 컴포넌트의 디렉토리 경로가 서로 동일하거나 부모관계인 경우 상기 중복명 컴포넌트를 병합하는 동작; 상기 디렉토리 경로가 상이한 경우, 상기 중복명 컴포넌트에 의존하는 컴포넌트의 동일성 여부를 판단하는 동작; 상기 중복명 컴포넌트에 의존하는 컴포넌트가 동일하지 않은 경우, 상기 중복명 컴포넌트를 유지하는 동작; 상기 중복명 컴포넌트에 의존하는 컴포넌트가 동일한 경우, 상기 중복명 컴포넌트 간에 동일

한 파일이 존재하지 않으면 상기 중복명 컴포넌트를 병합하는 동작; 및 상기 중복명 컴포넌트에 의존하는 컴포넌트가 동일한 경우, 상기 중복명 컴포넌트 간에 똥일한 파일이 존재하면 상기 중복명 컴포넌트를 유지하는 동작을 포함할 수 있다.

【0027】일 실시예에 따른 의존성 판별 장치는 명령어를 포함하는 메모리; 및 상기 명령어를 기초로 소정의 동작을 수행하는 프로세서를 포함하고, 상기 프로세서의 동작은 소정 함수에 대한 헤더파일 및 소스파일을 포함하는 모듈 단위로 타겟 소스코드를 구분하는 동작; 각 모듈에 포함된 함수의 컴포넌트명을 판별하는 동작; 각 모듈에 포함된 컴포넌트명의 개수를 기초로 모듈이 재사용한 제1 컴포넌트 및 모듈에 포함된 함수가 의존하는 제2 컴포넌트를 판별하는 동작; 상기 제1 컴포넌트 또는 상기 제2 컴포넌트가 임포트하는 헤더파일을 기초로 상기 모듈이 의존하는 라이브러리를 판별하는 동작; 및 상기 제2 컴포넌트에 대한 상기 제1 컴포넌트의 의존성 및 상기 라이브러리에 대한 상기 제1 컴포넌트 또는 상기 제2 컴포넌트의 의존성을 기초로 컴포넌트 간 의존성 그래프를 생성하는 동작을 포함할 수 있다.

【발명의 효과】

【0028】본 발명은 재사용이 불분명한 파일에 대한 컴포넌트 식별 정확도를 획기적으로 향상시킬 수 있다. 이를 위해, 본 발명은 타겟 소스코드에서 모든 함수의 선언과 정의를 추출하고, 동일한 함수의 선언을 가진 헤더 파일과 해당 선언 함수의 정의를 가진 소스 파일을 모듈로 구성함으로써, 모듈 단위로 컴포넌트 간의 의존성 관계를 분석함에 따라 재사용이 불분명한 파일의 컴포넌트를 특정할 수 있

게 한다.

【0029】 또한, 본 발명은 코드 재사용과 라이브러리 재사용의 두 가지 의존성 패턴을 모두 고려한 통합적인 의존성 분석을 제공할 수 있다. 이를 위해, 본 발명은 모듈 간의 코드 재사용 의존성과 라이브러리 재사용 의존성을 각각 분석하고, 이를 통합하여 의존성 그래프를 생성할 수 있다. 이러한 통합 분석 방법은 다양한 재사용 패턴을 포괄적으로 고려함으로써 기존 기술보다 더 높은 정확도와 정밀도를 제공할 수 있다.

【0030】 또한, 본 발명은 중복된 컴포넌트에 대한 문제를 해결하여 의존성 그래프의 정확도를 향상시킬 수 있다. 이를 위해, 본 발명은 유사한 컴포넌트 간의 디렉토리 경로, 다른 컴포넌트에 의한 의존성, 중복된 파일의 보유 여부를 바탕으로 중복된 컴포넌트를 효과적으로 식별함으로써, 보다 정확하고 신뢰성 있는 의존성 분석을 가능하게 한다.

【0031】 이러한 효과를 통해 본 발명은 소프트웨어 개발 및 유지보수 과정에서 발생할 수 있는 다양한 문제를 예방하게 한다. 본 발명은 높은 정확도와 정밀도로 소프트웨어 컴포넌트 간의 의존성을 분석함으로써, 개발 단계에서 발생할 수 있는 의존성 관련 오류를 사전에 식별하고 수정할 수 있게 한다. 또한, 유지보수 과정에서도 의존성 분석 결과를 바탕으로 효율적인 업데이트와 패치 적용이 가능해진다.

【0032】결론적으로, 본 발명은 컴포넌트 간의 의존성 분석을 통해 소프트웨어의 품질과 안전성을 높이고, 공급망의 투명성을 확보하며, 보안 취약점을 효과적으로 관리할 수 있는 강력한 도구를 제공할 수 있다. 이는 소프트웨어 산업 전반에 걸쳐 중요한 발전을 이룰 수 있는 기회를 제공하며, 궁극적으로 더 안전하고 신뢰성 있는 소프트웨어 환경을 구축하는 데 기여할 것이다.

【0033】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】

【0034】도 1은 일 실시예에 따른 의존성 판별 장치의 구성도이다.

도 2는 일 실시예에 따른 의존성 판별 장치가 수행하는 동작의 단계를 나타낸 흐름도이다.

도 3은 일 실시예에 따른 의존성 판별 장치가 수행하는 동작의 모습을 간단히 나타낸 개념도이다.

도 4는 일 실시예에 따른 모듈 단위를 설명하기 위한 예시도이다.

도 5는 일 실시예에 따라 제2 컴포넌트에 대한 제1 컴포넌트의 의존성 및 라이브러리에 대한 제1 컴포넌트 또는 제2 컴포넌트의 의존성을 기초로 컴포넌트 간 의존성 그래프를 생성한 예시도이다.

도 6은 일 실시예에 따라 중복명 컴포넌트 각각의 디렉토리 경로를 기초로

중복명 컴포넌트 간의 중복 여부를 판별하는 동작을 설명하기 위한 예시도이다.

도 7은 일 실시예에 따라 중복명 컴포넌트에 대한 다른 컴포넌트의 의존성을 기초로 중복명 컴포넌트 간의 중복 여부를 판별하는 동작을 설명하기 위한 예시도이다.

도 8은 일 실시예에 따라 중복명 컴포넌트 각각이 중복된 파일을 포함하는지의 여부를 기초로 중복명 컴포넌트 간의 중복 여부를 판별하는 동작을 설명하기 위한 예시도이다.

도 9는 일 실시예에 따라 중복된 컴포넌트를 병합하여 컴포넌트 간 의존성 그래프를 생성한 예시도이다.

도 10은 재사용 오픈소스 소프트웨어 구성 요소 간의 의존성을 분석하는 기존 방법인 CENTRIS와 본 문서의 실시예의 방법인 CNEPS의 성능 비교표이다.

【발명을 실시하기 위한 구체적인 내용】

【0035】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.

【0036】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 않는다.

【0037】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.

【0038】또한, 어떤 구성요소들을 포함한다는 표현은 "개방형"의 표현으로서 해당 구성요소들이 존재하는 것을 단순히 지칭할 뿐이며, 추가적인 구성요소들을 배제하는 것으로 이해되어서는 안 된다.

【0039】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.

【0040】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.

【0041】 도 1은 일 실시예에 따른 의존성 판별 장치(100)(이하, '장치(100)'로 지칭)의 구성도이다.

【0042】 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.

【0043】 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 후술할 타겟 소스코드, 컴포넌트 데이터베이스 등을 저장할 수 있다.

【0044】 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.

【0045】 입출력 인터페이스(130)는 정보를 입력하거나 출력하는 하드웨어 인터페이스 또는 소프트웨어 인터페이스를 포함할 수 있다.

【0046】 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.

【0047】 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.

【0048】 도 2는 일 실시예에 따른 장치(100)가 수행하는 동작의 흐름도이다. 도 3은 일 실시예에 따른 장치(100)가 수행하는 동작의 모습을 간단히 나타낸 개념도이다. 도 2 및 도 3의 실시예에 따른 장치(100)의 동작은 프로세서(120)에 의해 수행되는 동작으로 이해될 수 있다.

【0049】 도 2 및 도 3에 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 2 및 도 3에 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.

【0050】 도 2 및 도 3을 함께 참조하면, S1010 단계에서, 장치(100)는 타겟 소스코드에 포함된 컴포넌트 간의 의존성을 분석하기 위해 소스코드를 '모듈'이라는 단위로 구분한다. 이하, 타겟 소스코드, 컴포넌트, 컴포넌트 간의 의존성이라는 용어에 대한 정의를 먼저 설명한다.

【0051】 타겟 소스코드란 컴포넌트 간의 의존성을 분석하기 위한 대상이 되는 소스코드이다.

【0052】 컴포넌트란 소프트웨어 개발에서 재사용 가능한 코드의 단위이다. 컴포넌트는 하나의 완성된 기능을 가진 작은 프로그램 단위로서, 다른 소프트웨어에서 재사용될 수 있다. 예를 들어, 웹사이트에서 로그인 기능을 구현하는 코드를 하나의 컴포넌트로 만들면, 이 코드를 다른 웹사이트에서도 그대로 사용할 수 있다.

【0053】 컴포넌트 간의 의존성은 소프트웨어에서 한 컴포넌트가 다른 컴포넌트에 의존하는 관계를 의미한다. 예를 들어, 프로그램 A가 특정 기능을 수행하기 위해 라이브러리 B를 필요로 한다면, A는 B에 의존한다고 말하고, 본 문서의 도면에서는 'A->B'의 방향의 화살표로 의존 관계를 표현하기로 한다. 이러한 의존 관계는 컴포넌트 간의 연결을 나타내며, 하나의 컴포넌트가 업데이트되거나 변경될 때, 그에 의존하는 다른 컴포넌트들도 영향을 받을 수 있다. 따라서, 의존성을 정확하게 분석하는 것은 소프트웨어 시스템의 안정성과 유지보수에 매우 중요하다.

【0054】 본 발명은 후술할 도 4의 실시예에 따른 모듈의 단위를 사용하여 타겟 소스코드 중 재사용이 불분명한 파일의 컴포넌트와, 중복된 컴포넌트의 여부를 판별하여, 재사용 오픈소스 소프트웨어의 컴포넌트 간 의존성을 보다 정확하게 분석하는 방법을 제안하고자 한다.

【0055】 도 4는 일 실시예에 따른 모듈 단위를 설명하기 위한 예시도이다.

【0056】도 4를 참조하면, 본 문서의 실시예에 따른 모듈은 소정 함수를 선언하는 선언문을 포함하는 헤더파일과, 헤더파일에서 선언된 함수를 정의하는 정의문을 포함하는 소스파일을 합친 단위로 정의될 수 있다. 예를 들어, ares.h라는 헤더파일이 함수 ares_gethostbyaddr를 선언하고, ares_gethostbyaddr.c라는 소스파일이 그 함수를 정의하는 경우, 장치(100)는 ares.h라는 헤더파일과 ares_gethostbyaddr.c라는 소스파일을 하나의 모듈 단위로 구분할 수 있다.

【0057】장치(100)는 타겟 소스코드 전체를 상술한 모듈 단위로 구분하여, 타겟 소스코드를 복수의 모듈로 구분하고 각각의 모듈에 대하여 후술할 동작을 수행할 수 있다. 본 문서의 실시예는 이러한 모듈 단위를 기준으로 이후의 분석을 수행함으로써, 재사용이 불분명한 파일과 중복된 컴포넌트에 대한 의존성 분석을 가능하게 한다.

【0058】S1020 단계에서, 장치(100)는 각 모듈에 포함된 함수의 컴포넌트명을 판별할 수 있다. 이를 위해, 장치(100)는 함수에 대한 컴포넌트명을 저장하는 컴포넌트 데이터베이스와 각 모듈에 포함된 함수를 비교하여, 각 함수가 의존하는 컴포넌트명을 매핑시킬 수 있다. 예를 들어, 장치(100)는 IntelliJ IDEA, Visual Studio Code, Eclipse와 같은 통합 개발 환경(IDE)이나, SonarQube, ESLint, Pylint 등과 같은 분석 도구를 이용하여 모듈에 포함된 각 함수가 재사용하는 컴포넌트명을 검색하고, 검색된 컴포넌트명을 각 함수에 매핑시킬 수 있다.

【0059】예를 들어, 도 3에서 장치(100)는 제1 모듈에 포함된 모든 함수(A, B, C, D)가 재사용하는 컴포넌트를 검색하여 매핑한다. 이에 따라, A 함수가 의존

하는 컴포넌트는 aaa 컴포넌트, B 함수가 의존하는 컴포넌트는 aaa 컴포넌트, C 함수가 의존하는 컴포넌트는 '검색결과 없음', D 함수가 의존하는 컴포넌트는 bbb 컴포넌트로 검색하고, 각 함수에 검색된 컴포넌트 명을 매핑시킬 수 있다. 이때 각 함수에 매칭된 컴포넌트 명은 각 함수가 의존하는 컴포넌트를 의미한다. 도 3에서, 'C 함수'는 컴포넌트가 검색되지 않았으므로, '재사용이 불분명한 파일'에 해당한다. 재사용이 불분명한 파일에 해당하는 'C 함수'의 컴포넌트를 결정하는 방법은 후술할 S1030 단계에서 설명하기로 한다.

【0060】S1030 단계에서, 장치(100)는 각 모듈에 포함된 컴포넌트명의 개수를 기초로 모듈이 재사용하고 있는 제1 컴포넌트, 및 모듈에 포함된 함수가 의존하는 제2 컴포넌트를 판별할 수 있다. 즉, 제1 컴포넌트는 특정 모듈 자체가 재사용하고 있는 컴포넌트를 의미하며, 제2 컴포넌트는 제1 컴포넌트가 의존하고 있는 컴포넌트를 의미한다.

【0061】일 예로, 장치(100)는 제1 모듈에 포함된 컴포넌트명 중 포함된 개수가 가장 많은 컴포넌트명(이하, '최다 컴포넌트'로 지칭)을 선별하고, 제1 모듈의 헤더파일이 최다 컴포넌트의 원본 프로젝트에 포함되어 있는 경우, 최다 컴포넌트를 제1 컴포넌트로 결정할 수 있다. 또한, 제1 컴포넌트가 결정되면, 장치(100)는 제1 모듈에 포함된 컴포넌트명 중 제1 컴포넌트를 제외한 나머지 컴포넌트명을 제2 컴포넌트로 결정할 수 있다.

【0062】예를 들어, 도 3의 "코드 재사용 분석"에서 장치(100)는 제1 모듈에 'aaa 컴포넌트'에 의존하는 함수가 2개, 'bbb 컴포넌트'에 의존하는 함수가 1개 포

함되어 있으므로, 제1 모듈이 'aaa 컴포넌트'(제1 모듈 -> aaa)에 의존하는 것과, 제1 모듈이 'bbb 컴포넌트'에 의존(제1 모듈 -> bbb)하는 것으로 판별할 수 있다. 또한, 제1 모듈의 헤더파일이 'aaa 컴포넌트'의 원본 프로젝트에 포함되어 있다면, 'aaa 컴포넌트'를 제1 컴포넌트로 결정할 수 있다. 또한, 장치(100)는 제1 컴포넌트인 'aaa 컴포넌트'를 제외한 나머지 컴포넌트인 'bbb 컴포넌트'를 제2 컴포넌트로 결정할 수 있다. 장치(100)는 제1 컴포넌트가 제2 컴포넌트를 재사용하고 있으므로, 제1 컴포넌트가 제2 컴포넌트에 의존하는 것, 즉, 'aaa 컴포넌트'가 'bbb 컴포넌트'에 의존(aaa -> bbb)하는 것으로 판별할 수 있다. 이 경우, 해당 모듈 자체가 재사용하고 있는 컴포넌트는 'aaa 컴포넌트'이며, 'bbb 컴포넌트'는 'aaa 컴포넌트'가 의존하고 있는 컴포넌트를 의미한다.

【0063】만약, 제1 모듈의 헤더파일이 최다 컴포넌트명의 원본 프로젝트에 포함되어 있지 않은 경우, 장치(100)는 제1 모듈에 포함된 컴포넌트명 중 포함된 개수가 많은 순서로 컴포넌트명의 원본 프로젝트에 제1 모듈의 헤더파일의 포함 여부를 검색하고, 제1 모듈에 포함된 컴포넌트명 중 제1 모듈의 헤더파일이 검색된 컴포넌트명을 제1 컴포넌트로 결정할 수 있다.

【0064】예를 들어, 도 3의 "코드 재사용 분석"에서 제1 모듈의 헤더파일이 'aaa 컴포넌트'의 원본 프로젝트에 포함되어 않다면, 장치(100)는 제1 모듈의 헤더파일이 'aaa 컴포넌트' 다음으로 포함된 개수가 많은 'bbb 컴포넌트'의 원본 프로젝트에 포함되어 있는지 검사할 수 있다. 이때 제1 모듈의 헤더파일이 'bbb 컴포넌트'의 원본 프로젝트에 포함되어 있다면, 'bbb 컴포넌트'를 제1 컴포넌트로 결정할

수 있다. 또한, 장치(100)는 제1 컴포넌트인 'bbb 컴포넌트'를 제외한 나머지 컴포넌트명인 'aaa 컴포넌트'를 제2 컴포넌트로 결정할 수 있다. 이 경우, 해당 모듈 자체가 재사용하고 있는 컴포넌트는 'aaa 컴포넌트'이며, 'bbb 컴포넌트'는 'aaa 컴포넌트'가 의존하고 있는 컴포넌트를 의미한다.

【0065】이후의 실시예에서는 이해의 편의를 위해, 제1 컴포넌트가 'aaa 컴포넌트', 제2 컴포넌트가 'bbb 컴포넌트'로 판별된 것으로 예시하여 이후의 실시예를 설명한다.

【0066】한편, 장치(100)는 모듈에 포함된 함수 중 컴포넌트명이 판별되지 않은 함수인 '재사용이 불분명한 파일'가 존재하는 경우, 컴포넌트명이 판별되지 않은 함수의 컴포넌트는, 해당 함수가 포함된 모듈 자체가 재사용하는 것으로 판별된 제1 컴포넌트에 의존하는 것으로 판별할 수 있다.

【0067】예를 들어, 도 3의 경우, 'C 함수'는 컴포넌트명이 판별되지 않은 재사용이 불분명한 파일에 해당한다. 이 경우, 장치(100)는 컴포넌트명이 판별되지 않은 'C 함수'의 컴포넌트를 제1 컴포넌트인 'aaa 컴포넌트'에 의존하는 것으로 설정할 수 있다.

【0068】이처럼, 컴포넌트명이 판별되지 않은 함수의 컴포넌트를 제1 컴포넌트로 설정하는 이유는, 도 3의 'C 함수'는 라이브러리로 재사용 가능한 단위인 제1 모듈 단위 내에 포함되어 있다는 점, 또한 제1 모듈 자체는 'aaa 컴포넌트'의 원본 프로젝트로부터 재사용된 것으로 판별되었으므로 'C 함수'가 aaa 컴포넌트로부터 재사용된 것일 확률이 매우 높다는 점 때문이다. 실제로, 어떠한 모듈에 대한 재사

용 컴포넌트의 출처가 밝혀진 경우, 해당 모듈에 포함된 재사용이 불분명한 함수를 직접 컴포넌트의 원본 프로젝트에서 검색하는 경우, 해당 컴포넌트에 포함되어 있을 확률은 91.5%로 나타난다.

【0069】S1040 단계에서, 장치(100)는 제1 컴포넌트 또는 제2 컴포넌트가 임포트하는 헤더파일을 기초로 상기 모듈이 의존하는 라이브러리를 판별할 수 있다.

【0070】일 예로, 제1 모듈이 임포트하는 헤더파일과, 제1 컴포넌트가 임포트하는 헤더파일이 상이한 경우, 장치(100)는 제1 컴포넌트가 제1 컴포넌트가 임포트하는 헤더파일의 라이브러리에 의존하는 것으로 판별할 수 있다. 또한, 제1 모듈이 임포트하는 헤더파일과, 제2 컴포넌트가 임포트하는 헤더파일이 상이한 경우, 장치(100)는 제2 컴포넌트가 제2 컴포넌트가 임포트하는 헤더파일의 라이브러리에 의존하는 것으로 판별할 수 있다. 즉, 장치(100)는 제1 모듈이 임포트하는 제1 헤더파일과, 제1 컴포넌트 또는 제2 컴포넌트가 임포트하는 제2 헤더파일이 상이한 경우, 제1 모듈이 제2 헤더파일의 라이브러리를 의존하는 것으로 판별할 수 있다.

【0071】예를 들어, 도 3의 "라이브러리 재사용 분석"에서 제1 모듈의 제1 헤더파일이 'stdio.h를 임포트'하고 있고, 'aaa 컴포넌트'의 제2 헤더파일이 'xxx.h'라는 헤더파일(xxx.h 헤더의 컴포넌트명은 ccc)을 임포트하고 있으며, 'bbb 컴포넌트'의 제2 헤더파일이 'yyy.h'라는 헤더파일(yyy.h 헤더의 컴포넌트명은 ddd)을 임포트하고 있는 것으로 가정한다. 이 경우, 제1 헤더파일인 'stdio.h'와 제2 헤더파일인 'xxx.h'는 상이하며, 제1 헤더파일인 'stdio.h'와 제2 헤더파일인 'yyy.h'는 상이하므로, 장치(100)는 'aaa 컴포넌트'가 'xxx.h' 헤더의 컴포넌트명

인 ccc라는 라이브러리에 의존(aaa->ccc)하고 있으며, 'bbb 컴포넌트'가 'yyy.h' 헤더의 컴포넌트명인 ddd라는 라이브러리에 의존(bbb->ddd)하는 것으로 판별할 수 있다.

【0072】S1050 단계에서, 장치(100)는 S1030 단계에서 판별한 제2 컴포넌트에 대한 제1 컴포넌트의 의존성, 및 S1040 단계에서 판별한 라이브러리에 대한 제1 컴포넌트 또는 제2 컴포넌트의 의존성을 기초로 컴포넌트 간 의존성 그래프를 생성할 수 있다.

【0073】도 5는 일 실시예에 따라 제2 컴포넌트에 대한 제1 컴포넌트의 의존성 및 라이브러리에 대한 제1 컴포넌트 또는 제2 컴포넌트의 의존성을 기초로 컴포넌트 간 의존성 그래프를 생성한 예시도이다.

【0074】도 5를 참조하면, 장치(100)는 S1030 단계에서 제1 모듈이 aaa 컴포넌트(제1 모듈 -> aaa)에 의존하는 것, 제1 모듈이 bbb 컴포넌트에 의존(제1 모듈 -> bbb)하는 것, 제1 컴포넌트가 제2 컴포넌트에 의존하는 것으로서 aaa 컴포넌트가 bbb 컴포넌트에 의존(aaa -> bbb)하는 것으로 판별하였다. 또한, 장치(100)는 S1040 단계에서 aaa 컴포넌트가 xxx.h 헤더의 컴포넌트명인 ccc라는 라이브러리에 의존(aaa->ccc)하고 있으며, bbb 컴포넌트가 yyy.h 헤더의 컴포넌트명인 ddd라는 라이브러리에 의존(bbb->ddd)하는 것을 판별하였다. 장치(100)는 이러한 의존 관계를 전체적으로 종합하여 도 5와 같은 의존 그래프를 생성할 수 있다.

【0075】또한, 도 5에서 제1 컴포넌트 또는 제2 컴포넌트 중 이름이 동일한 중복 컴포넌트(ex. 도 3의 A 함수에 매핑된 aaa, B 함수에 매핑된 aaa)가 복수 개 판별된 경우, 장치(100)는 이름이 동일한 중복 컴포넌트의 인덱스를 구분(ex. aaa, aaa')하여 표현할 수 있다.

【0076】한편, 도 5에서 이름이 동일한 컴포넌트인 aaa(도 3의 A 함수에 매핑된 aaa, B 함수에 매핑된 aaa)는 '중복된 컴포넌트인지에 대한 여부'의 확인이 필요하다. 컴포넌트 간의 의존성 분석 과정에서 복수 개의 동일한 컴포넌트를 중복으로 인식하지 못하게 되면 의존성 분석 결과에 오차가 발생하게 되어, 의존성에 기반한 소프트웨어의 활용에서 오류가 발생할 수 있기 때문이다.

【0077】이처럼, 제1 컴포넌트 또는 제2 컴포넌트 중 컴포넌트명이 동일한 중복명 컴포넌트가 복수 개 있는 경우, 장치(100)는 다음 도 6 내지 도 8에 따른 중복명 컴포넌트 각각의 디렉토리 경로, 중복명 컴포넌트에 대한 다른 컴포넌트의 의존성 및 중복된 파일의 존재 여부의 검사를 통해, 중복명 컴포넌트의 중복 여부를 결정하여, 도 9와 같이 중복을 제거한 컴포넌트 간 의존성 그래프를 생성할 수 있다.

【0078】도 6은 일 실시예에 따라 중복명 컴포넌트 각각의 디렉토리 경로를 기초로 중복명 컴포넌트 간의 중복 여부를 판별하는 동작을 설명하기 위한 예시도이다.

【0079】도 6을 참조하면, 장치(100)는 컴포넌트명이 동일한 중복명 컴포넌트 aaa와 aaa'(윗첨자는 중복명 컴포넌트를 구분하기 위한 인덱스)에 대하여, 원본 프로젝트에서 각 컴포넌트가 위치하는 디렉토리 경로를 확인할 수 있다. 장치(100)는 중복명 컴포넌트 각각의 디렉토리 경로가 서로 동일하거나, 디렉토리가 부모 관계인 경우인지, aaa와 aaa'가 중복된 컴포넌트인 것으로 판별하여 병합할 수 있다. 장치(100)는 중복명 컴포넌트 각각의 디렉토리 경로가 서로 상이하다면, aaa와 aaa'가 중복된 컴포넌트가 아닌 것으로 판별하여, 둘을 병합하지 않고 의존성 그래프에서 aaa와 aaa'를 구분하여 유지할 수 있다.

【0080】 도 7은 일 실시예에 따라 중복명 컴포넌트에 대한 다른 컴포넌트의 의존성을 기초로 중복명 컴포넌트 간의 중복 여부를 판별하는 동작을 설명하기 위한 예시도이다.

【0081】도 7을 참조하면, 장치(100)는 컴포넌트명이 동일한 중복명 컴포넌트 aaa와 aaa'에 의존하는 다른 컴포넌트를 확인할 수 있다. 만약, aaa와 aaa'에 둘 다 의존하고 있는 컴포넌트인 ccc가 존재한다면, 장치(100)는 aaa와 aaa'가 중복된 컴포넌트인 것으로 판별하여 병합할 수 있다. 장치(100) aaa와 aaa' 중 어느 하나에만 의존하고 있는 컴포넌트가 있거나, 둘 다 의존하는 컴포넌트가 존재하지 않는다면, aaa와 aaa'가 서로 중복된 컴포넌트가 아닌 것으로 판별하여, 의존성 그래프에서 둘을 병합하지 않고 aaa와 aaa'를 유지할 수 있다.

【0082】도 8은 일 실시예에 따라 중복명 컴포넌트 각각이 중복된 파일을 포함하는 지의 여부를 기초로 중복명 컴포넌트 간의 중복 여부를 판별하는 동작을 설

명하기 위한 예시도이다.

【0083】도 8을 참조하면, 장치(100)는 컴포넌트명이 동일한 중복명 컴포넌트 aaa와 aaa'의 원본 프로젝트 폴더의 파일을 확인할 수 있다. aaa와 aaa' 각각의 원본 프로젝트 폴더 간에 동일한 파일이 존재하지 않는다면, 장치(100)는 aaa와 aaa'가 중복된 컴포넌트인 것으로 판별하여 병합할 수 있다. 만약, aaa와 aaa' 각각의 원본 프로젝트 폴더 간에 동일한 파일이 존재한다면, 장치(100)는 aaa와 aaa'가 아닌 것으로 판별하여, 의존성 그래프에서 둘을 병합하지 않고 aaa와 aaa'를 유지할 수 있다.

【0084】본 문서의 실시예는 상기 도 6 내지 도 8의 판단을 다양한 방식으로 조합하여 중복명 컴포넌트의 중복 여부를 판별하는 3단계 검토 알고리즘을 설계할 수 있다.

【0085】예를 들어, 장치(100)는 1차 검토 과정으로서, 중복명 컴포넌트의 디렉토리 경로를 확인하여, 중복명 컴포넌트의 디렉토리 경로가 서로 동일하거나 부모관계인 경우에 중복명 컴포넌트를 서로 병합하고 검토 과정을 종료하며, 디렉토리 경로가 상이한 경우 병합을 보류하고 다음의 2차 검토 과정을 수행할 수 있다.

【0086】장치(100)는 2차 검토과정으로서, 중복명 컴포넌트 각각을 의존하는 다른 컴포넌트를 확인하여, 중복명 컴포넌트를 의존하는 컴포넌트가 서로 동일하지 않은 경우, 중복명 컴포넌트를 병합하지 않고 유지하며 검토 과정을 종료할 수 있다. 만약, 중복명 컴포넌트에 의존하는 컴포넌트가 동일한 경우, 병합을 보류하고

다음의 3차 검토 과정을 수행할 수 있다.

【0087】장치(100)는 3차 검토 과정으로서, 중복명 컴포넌트 각각의 원본 프로젝트 폴더 간에 동일한 파일이 존재하는지 확인하여, 동일한 파일이 존재하지 않는다면 중복명 컴포넌트를 병합하고 검토 과정을 종료할 수 있다, 동일한 파일이 존재한다면 중복이 아닌 것으로 판단하여 중복명 컴포넌트를 유지하고 검토 과정을 종료할 수 있다.

【0088】도 9는 일 실시예에 따라 중복된 컴포넌트를 병합하여 컴포넌트 간 의존성 그래프를 생성한 예시도이다.

【0089】도 9를 참조하면, 장치(100)는 컴포넌트명이 동일한 중복명 컴포넌트 aaa와 aaa'가 서로 도 6 내지 도 8의 과정을 통해 중복된 컴포넌트인 것으로 판별한 경우, 도 5의 aaa와 aaa'가 서로 중복된 것이므로 둘을 병합하여 도 9와 같이 나타낼 수 있다.

【0090】도 10은 재사용 오픈소스 소프트웨어 구성 요소 간의 의존성을 분석하는 기존 방법인 CENTRIS와 본 문서의 실시예의 방법인 CNEPS의 성능 비교표이다. 도 10의 비교는 정확도(Precision)와 재현율(Recall) 관점에서 성능이 측정되었다.

【0091】도 10을 참조하면, 세 가지 그래프 분류(Small, Moderate, Large)에 따른 두 접근법의 세부 성능을 비교하였을 때, CNEPS 접근법은 총 100개의 노드에서 75,193개의 재사용 파일과 534개의 의존성을 식별했으며, 정확도는 89.9%, 재현율은 93.2%로 매우 높은 성능을 나타냈다. 반면 CENTRIS 접근법은 동일한 100개의

노드에서 40,582개의 재사용 파일과 345개의 의존성을 식별했으며, 정확도는 63.5%, 재현율은 42.5%로 비교적 낮은 성능을 보였다.

【0092】 또한, 도 10의 정확도 측면에서 CNEPS 접근법은 89.9%로, CENTRIS의 63.5% 보다 훨씬 높은 성능을 보인다. 이는 CNEPS가 의존성을 식별할 때 더 많은 True Positive를 올바르게 판별함을 의미한다. 재현율 측면에서도 CNEPS 접근법은 93.2%로, CENTRIS의 42.5%를 크게 앞선다. 이는 CNEPS가 True Positive 중 더 많은 부분을 정확히 찾아냈다는 것을 나타낸다.

【0093】 이처럼 도 10의 결과는 의존성 분석에서 CNEPS 접근법이 CENTRIS 접근법보다 더 높은 정확도와 재현율을 제공함을 의미한다. 이는 소프트웨어 구성 요소 간의 의존성을 분석하는 데 있어 CNEPS 접근법이 보다 효과적임을 시사한다.

【0094】 상술한 실시예에 따르면, 본 발명은 재사용이 불분명한 파일에 대한 컴포넌트 식별 정확도를 획기적으로 향상시킬 수 있다. 이를 위해, 본 발명은 타겟 소스코드에서 모든 함수의 선언과 정의를 추출하고, 동일한 함수의 선언을 가진 헤더 파일과 해당 선언 함수의 정의를 가진 소스 파일을 모듈로 구성함으로써, 모듈 단위로 컴포넌트 간의 의존성 관계를 분석함에 따라 재사용이 불분명한 파일의 컴포넌트를 특정할 수 있게 한다.

【0095】 또한, 본 발명은 코드 재사용과 라이브러리 재사용의 두 가지 의존성 패턴을 모두 고려한 통합적인 의존성 분석을 제공할 수 있다. 이를 위해, 본 발명은 모듈 간의 코드 재사용 의존성과 라이브러리 재사용 의존성을 각각 분석하고, 이를 통합하여 의존성 그래프를 생성할 수 있다. 이러한 통합 분석 방법은 다양한

재사용 패턴을 포괄적으로 고려함으로써 기존 기술보다 더 높은 정확도와 정밀도를 제공할 수 있다.

【0096】또한, 본 발명은 중복된 컴포넌트에 대한 문제를 해결하여 의존성 그래프의 정확도를 향상시킬 수 있다. 이를 위해, 본 발명은 유사한 컴포넌트 간의 디렉토리 경로, 다른 컴포넌트에 의한 의존성, 중복된 파일의 보유 여부를 바탕으로 중복된 컴포넌트를 효과적으로 식별함으로써, 보다 정확하고 신뢰성 있는 의존성 분석을 가능하게 한다.

【0097】이러한 효과를 통해 본 발명은 소프트웨어 개발 및 유지보수 과정에서 발생할 수 있는 다양한 문제를 예방하게 한다. 본 발명은 높은 정확도와 정밀도로 소프트웨어 컴포넌트 간의 의존성을 분석함으로써, 개발 단계에서 발생할 수 있는 의존성 관련 오류를 사전에 식별하고 수정할 수 있게 한다. 또한, 유지보수 과정에서도 의존성 분석 결과를 바탕으로 효율적인 업데이트와 패치 적용이 가능해진다.

【0098】결론적으로, 본 발명은 컴포넌트 간의 의존성 분석을 통해 소프트웨어의 품질과 안전성을 높이고, 공급망의 투명성을 확보하며, 보안 취약점을 효과적으로 관리할 수 있는 강력한 도구를 제공할 수 있다. 이는 소프트웨어 산업 전반에 걸쳐 중요한 발전을 이룰 수 있는 기회를 제공하며, 궁극적으로 더 안전하고 신뢰성 있는 소프트웨어 환경을 구축하는 데 기여할 것이다.

【0099】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.

【0100】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.

【0101】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.

【0102】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 등등을 포함할 수 있다.

【0103】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 등등 일 수 있다.

【0104】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 용어는 데이터가 저장매체에 반영구적으로 저장되는 경우와 임시적으로 저장되는 경

우를 구분하지 않는다.

【0105】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.

【0106】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 실시예들에 따르면, 모듈, 프로그램 또는 다른 구성요소에 의해 수행되는 동작들은 순

차적으로, 병렬적으로, 반복적으로, 또는 휴리스틱하게 실행되거나, 동작들 중 하나 이상이 다른 순서로 실행되거나, 생략되거나, 또는 하나 이상의 다른 동작들이 추가될 수 있다.

【부호의 설명】

【0107】100: 장치

110: 메모리

120: 프로세서

130: 입출력 인터페이스

140: 통신 인터페이스

【청구범위】

【청구항 1】

프로세서에 의해 동작하는 의존성 판별 장치가 수행하는 방법에 있어서,

소정 함수에 대한 헤더파일 및 소스파일을 포함하는 모듈 단위로 타겟 소스코드를 구분하는 동작;

각 모듈에 포함된 함수의 컴포넌트명을 판별하는 동작;

각 모듈에 포함된 컴포넌트명의 개수를 기초로 모듈이 재사용한 제1 컴포넌트 및 모듈에 포함된 함수가 의존하는 제2 컴포넌트를 판별하는 동작;

상기 제1 컴포넌트 또는 상기 제2 컴포넌트가 임포트하는 헤더파일을 기초로 상기 모듈이 의존하는 라이브러리를 판별하는 동작; 및

상기 제2 컴포넌트에 대한 상기 제1 컴포넌트의 의존성 및 상기 라이브러리에 대한 상기 제1 컴포넌트 또는 상기 제2 컴포넌트의 의존성을 기초로 컴포넌트간 의존성 그래프를 생성하는 동작을 포함하는,

방법.

【청구항 2】

제1항에 있어서,

상기 헤더파일은

상기 소정 함수를 선언하는 선언문을 포함하고,

상기 소스파일은

상기 헤더파일에서 선언된 함수를 정의하는 정의문을 포함하는,

방법.

【청구항 3】

제1항에 있어서,

상기 컴포넌트명을 판별하는 동작은

함수에 대한 컴포넌트명을 저장하는 컴포넌트 데이터베이스를 상기 각 모듈에 포함된 함수와 비교하여 상기 각 모듈에 포함된 함수에 컴포넌트명을 매핑시키는 동작을 포함하는,

방법.

【청구항 4】

제1항에 있어서,

상기 제1 컴포넌트 및 제2 컴포넌트를 판별하는 동작은

제1 모듈에 포함된 컴포넌트명 중 포함된 개수가 가장 많은 컴포넌트명을 선별하는 동작; 및

상기 제1 모듈의 헤더파일이 상기 개수가 가장 많은 컴포넌트명의 원본 프로젝트에 포함되어 있는 경우, 상기 개수가 가장 많이 포함된 컴포넌트명을 상기 제1 컴포넌트로 결정하는 동작을 포함하는

방법.

【청구항 5】

제4항에 있어서,

상기 제1 컴포넌트 및 제2 컴포넌트를 판별하는 동작은

상기 제1 모듈의 헤더파일이 상기 개수가 가장 많은 컴포넌트명의 원본 프로젝트에 포함되어 있지 않은 경우,

상기 제1 모듈에 포함된 컴포넌트명 중 포함된 개수가 많은 순서로 컴포넌트명의 원본 프로젝트에 상기 제1 모듈의 헤더파일의 포함 여부를 검색하는 동작; 및

상기 제1 모듈에 포함된 컴포넌트명 중 상기 제1 모듈의 헤더파일이 검색된 컴포넌트명을 상기 제1 컴포넌트로 결정하는 동작을 포함하는

방법.

【청구항 6】

제4항에 있어서,

상기 제1 컴포넌트 및 제2 컴포넌트를 판별하는 동작은

상기 제1 컴포넌트로 결정하는 동작 이후에,

상기 제1 모듈에 포함된 컴포넌트명 중 상기 제1 컴포넌트를 뺀 컴포넌트명을 상기 제2 컴포넌트로 결정하는 동작을 포함하는,

방법.

【청구항 7】

제1항에 있어서,

상기 제1 컴포넌트 및 제2 컴포넌트를 판별하는 동작은

모듈에 포함된 함수 중 컴포넌트명이 판별되지 않은 함수가 존재하는 경우, 상기 컴포넌트명이 판별되지 않은 함수의 컴포넌트를 제1 컴포넌트로 설정하는 동작을 포함하는,

방법.

【청구항 8】

제1항에 있어서,

상기 라이브러리를 판별하는 동작은

상기 모듈이 임포트하는 제1 헤더파일과 상기 제1 컴포넌트 또는 상기 제2 컴포넌트가 임포트하는 제2 헤더파일이 상이한 경우, 상기 모듈이 상기 제2 헤더파일의 라이브러리를 의존하는 것으로 판별하는 동작을 포함하는,

방법.

【청구항 9】

제1항에 있어서,

상기 의존성 그래프를 생성하는 동작은

상기 제1 컴포넌트 또는 제2 컴포넌트 중 컴포넌트명이 동일한 중복명 컴포넌트가 복수 개가 있는 경우, 상기 중복명 컴포넌트 각각의 디렉토리 경로, 상기 중복명 컴포넌트 각각에 대한 다른 컴포넌트에 의한 의존성 및 상기 중복명 컴포넌트 각각이 중복된 파일을 포함하는지의 여부 중 적어도 하나를 기초로, 상기 중복

명 컴포넌트의 병합 여부를 결정하여 컴포넌트 간 의존성 그래프를 생성하는 동작을 포함하는,

방법.

【청구항 10】

제9항에 있어서,

상기 의존성 그래프를 생성하는 동작은

상기 중복명 컴포넌트의 디렉토리 경로가 서로 동일하거나 부모관계인 경우 상기 중복명 컴포넌트를 병합하는 동작;

상기 디렉토리 경로가 상이한 경우, 상기 중복명 컴포넌트에 의존하는 컴포넌트의 동일성 여부를 판단하는 동작;

상기 중복명 컴포넌트에 의존하는 컴포넌트가 동일하지 않은 경우, 상기 중복명 컴포넌트를 유지하는 동작;

상기 중복명 컴포넌트에 의존하는 컴포넌트가 동일한 경우, 상기 중복명 컴포넌트 간에 동일한 파일이 존재하지 않으면 상기 중복명 컴포넌트를 병합하는 동작; 및

상기 중복명 컴포넌트에 의존하는 컴포넌트가 동일한 경우, 상기 중복명 컴포넌트 간에 똥일한 파일이 존재하면 상기 중복명 컴포넌트를 유지하는 동작을 포함하는,

방법.

【청구항 11】

명령어를 포함하는 메모리; 및

상기 명령어를 기초로 소정의 동작을 수행하는 프로세서를 포함하고,

상기 프로세서의 동작은,

소정 함수에 대한 헤더파일 및 소스파일을 포함하는 모듈 단위로 타겟 소스코드를 구분하는 동작;

각 모듈에 포함된 함수의 컴포넌트명을 판별하는 동작;

각 모듈에 포함된 컴포넌트명의 개수를 기초로 모듈이 재사용한 제1 컴포넌트 및 모듈에 포함된 함수가 의존하는 제2 컴포넌트를 판별하는 동작;

상기 제1 컴포넌트 또는 상기 제2 컴포넌트가 임포트하는 헤더파일을 기초로 상기 모듈이 의존하는 라이브러리를 판별하는 동작; 및

상기 제2 컴포넌트에 대한 상기 제1 컴포넌트의 의존성 및 상기 라이브러리에 대한 상기 제1 컴포넌트 또는 상기 제2 컴포넌트의 의존성을 기초로 컴포넌트간 의존성 그래프를 생성하는 동작을 포함하는,

의존성 판별 장치.

【요약서】

【요약】

일 실시예에 따른 의존성 판별 장치는 소정 함수에 대한 헤더파일 및 소스파일을 포함하는 모듈 단위로 타겟 소스코드를 구분하는 동작; 각 모듈에 포함된 함수의 컴포넌트명을 판별하는 동작;각 모듈에 포함된 컴포넌트명의 개수를 기초로 모듈이 재사용한 제1 컴포넌트 및 모듈에 포함된 함수가 의존하는 제2 컴포넌트를 판별하는 동작; 상기 제1 컴포넌트 또는 상기 제2 컴포넌트가 임포트하는 헤더파일을 기초로 상기 모듈이 의존하는 라이브러리를 판별하는 동작; 및 상기 제2 컴포넌트에 대한 상기 제1 컴포넌트의 의존성 및 상기 라이브러리에 대한 상기 제1 컴포넌트 또는 상기 제2 컴포넌트의 의존성을 기초로 컴포넌트 간 의존성 그래프를 생성하는 동작을 수행할 수 있다.

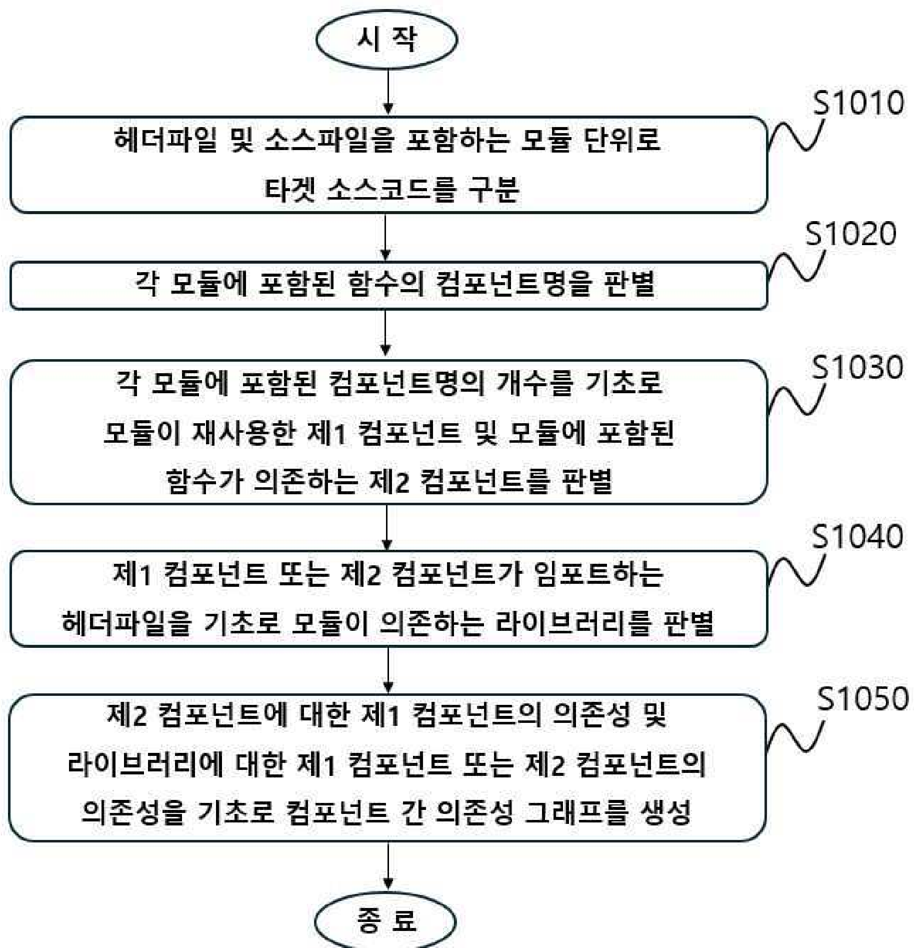
【대표도】

도 3

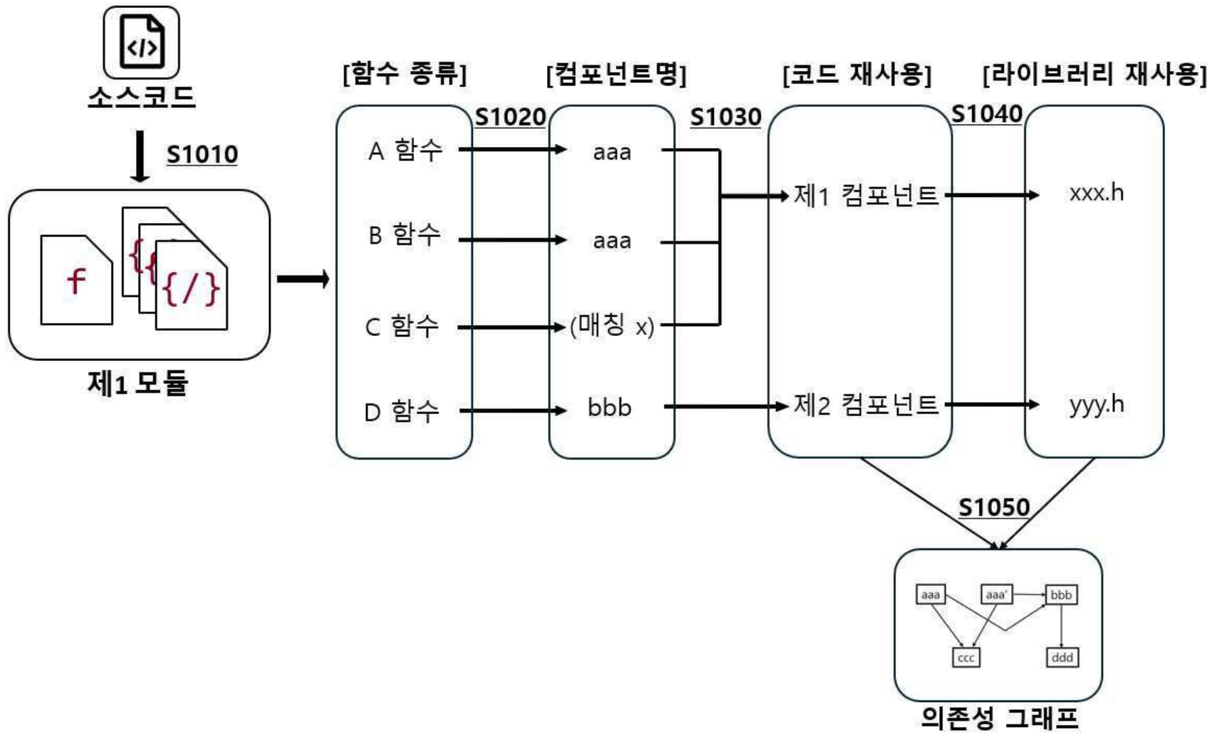
【도면】

【도 1】

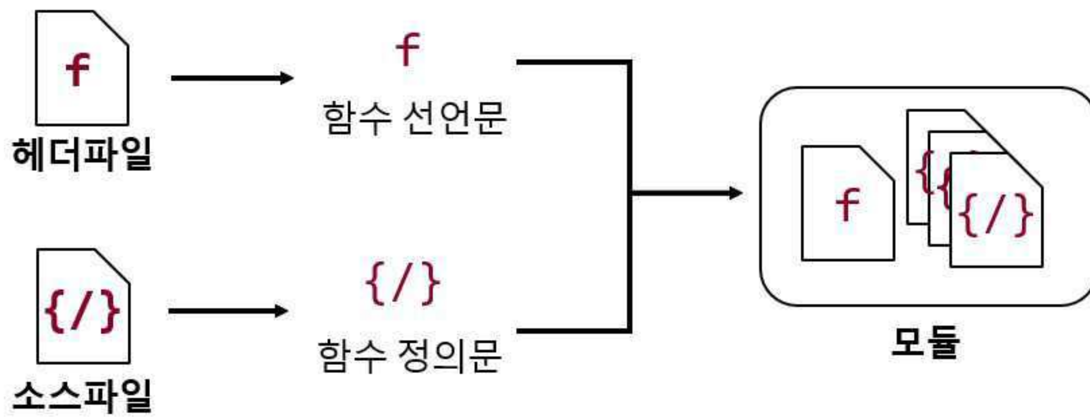
【도 2】



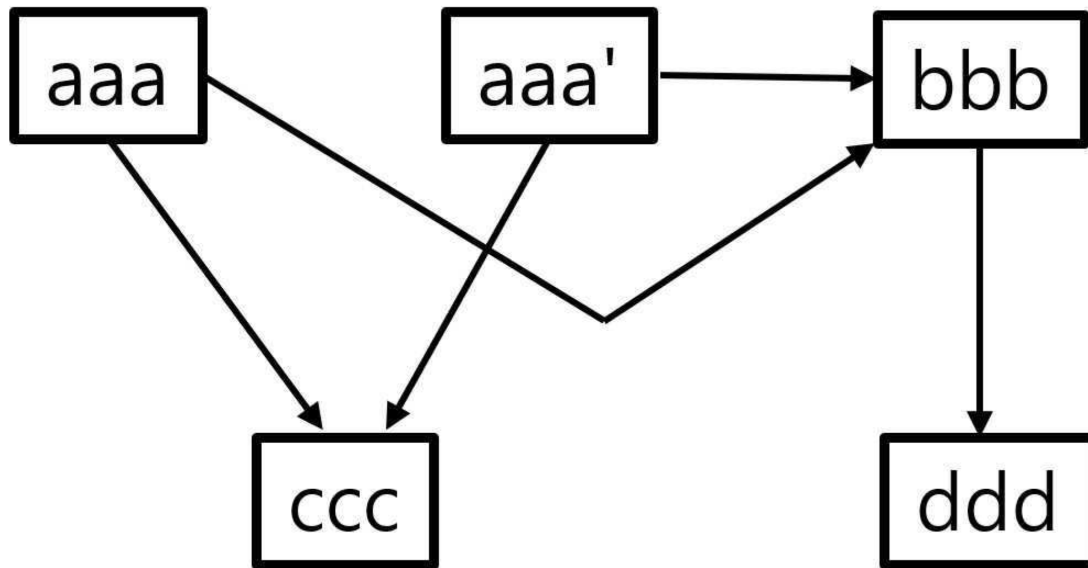
【도 3】



【도 4】

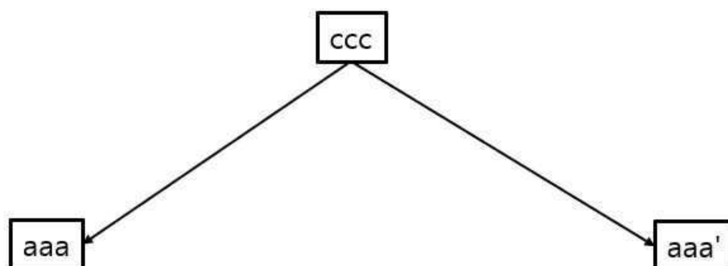


【도 5】

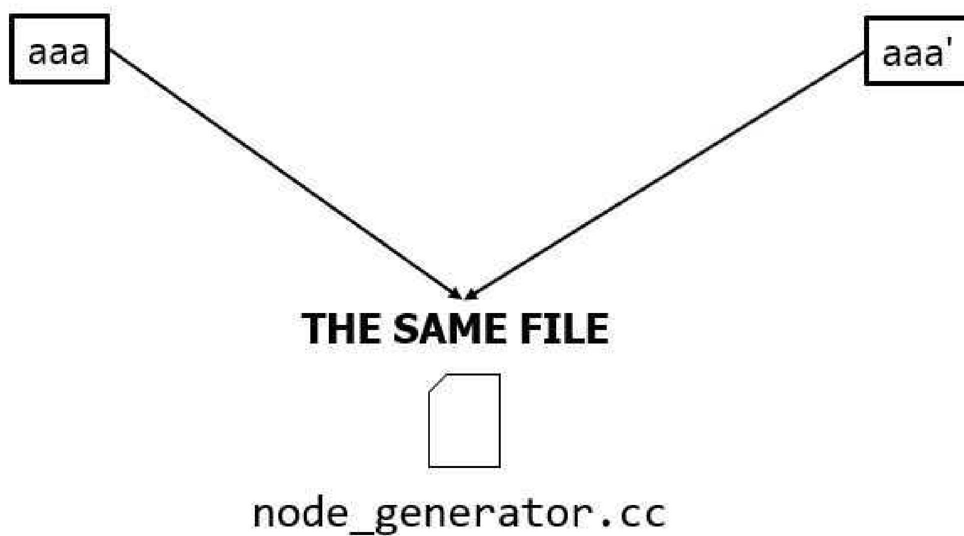


【도 6】

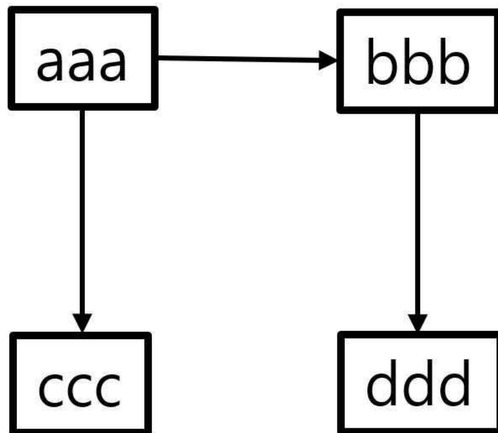
【도 7】



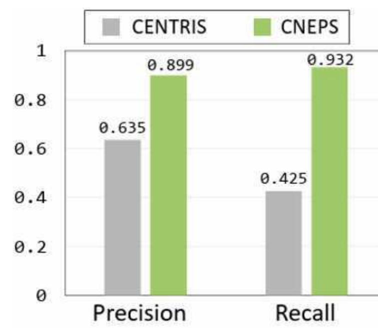
【도 8】



【도 9】



【도 10】



Approach	Graph classification*	#Included nodes	#Identified reused files	#Identified dependencies	#TPs	#FPs	#FNs	Precision	Recall
CENTRIS	*Small*	68	8,843	17	11	6	0	64.7%	100%
	Moderate	21	7,741	102	56	46	52	54.9%	51.9%
	Large	11	23,998	226	152	74	244	67.3%	38.4%
	Total	100	40,582	345	219	126	296	63.5%	42.5%
CNEPS	*Small*	68	18,212	11	11	0	0	100%	100%
	Moderate	21	15,160	108	106	2	2	98.1%	98.1%
	Large	11	41,821	415	363	52	33	87.5%	92.8%
	Total	**100**	75,193	**534**	**480**	**54**	**35**	89.9%	93.2%