소프트웨어 성능 가시화를 위한 툴 체인 개발

2016년 01월 29일

석사과정 강건희
홍익대학교 소프트웨어공학 연구실
Kang@selab.hongik.ac.kr
지도교수 : 김영철

목차

- 연구 배경 및 목적
- 관련연구
- 소프트웨어 성능 가시화를 위한 툴 체인 개발
- 결론 및 향후 연구

연구 배경

- 우리나라 소프트웨어의 생산70%가 중소기업에 편중
  - 중소기업은 개발 과정에서 구현에만 치중
  - 소프트웨어 공학적으로 고품질화 되고 있지 않음
  - 소프트웨어 고품질화를 위한 소프트웨어 공학적 인력 및 비용을 충당할 수 있는 회사가 그렇게 많지 않은 현실

- 소프트웨어의 고품질을 위해서
  - 인증을 통한 SW의 성숙도 향상
  - SW개발 방법론이나 Process를 통한 고품질 SW개발
  - Test Process 및 Testing을 통한 품질 향상 이 필요하다.

- But! 위의 방법은
  - 개발 시작 단계에서 적용이 되어야 오류를 만들지 않는 방법
  - 개발 중간 혹은 이미 만들어진 SW의 변경이나 복구가 어려움

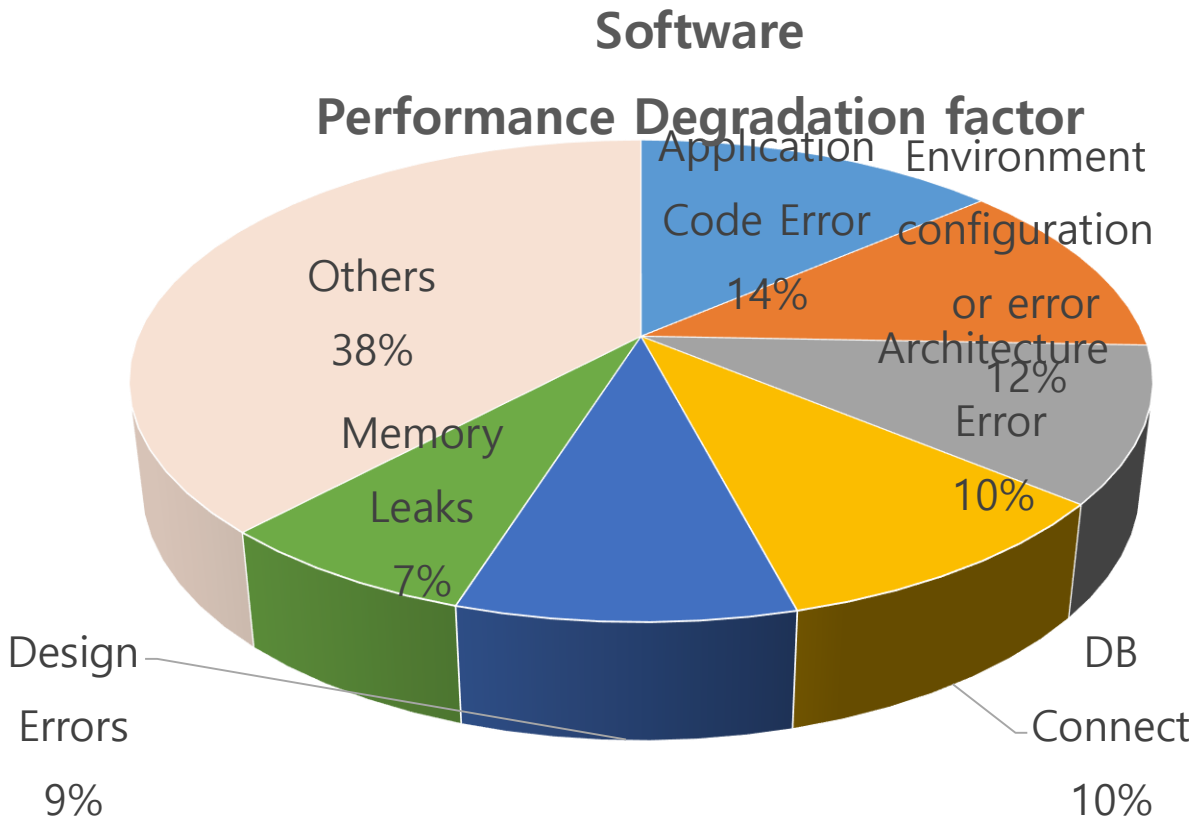
연구 배경

- 현재 우리나라의 소프트웨어 시장의 문제점을 해결하기 위해서
  - 기존 코드의 문제점을 발견하고 이를 변경하는 것 까지 가능해야 함

→ 소프트웨어 가시화가 필요!

- 기존연구에서
  - 소프트웨어의 재 사용성 관점에서 소프트웨어 복잡도에 대한 가시화 연구를 진행 함.

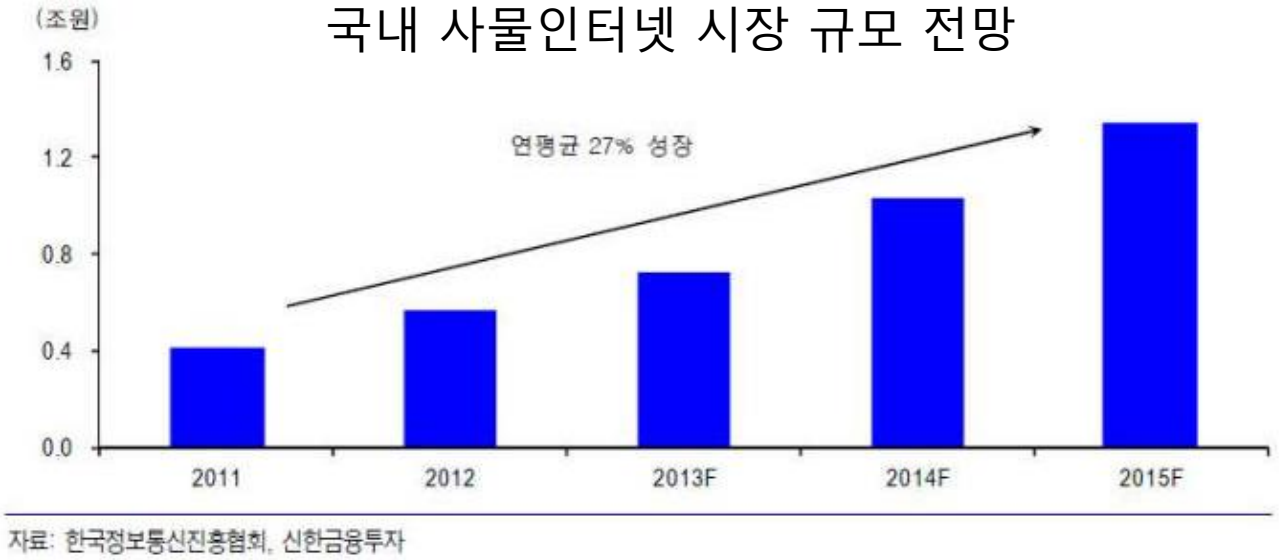
연구 배경



- SW의 성능품질 저하의 요소는 메모리부족, 자바 가상 머신 문제 등 환경적인 문제 가 존재.
- 하지만 SW 성능품질저하의 33%는 SW설계, 소스코드 오류

연구 배경

- IOT산업의 급상승
  - 임베디드 시스템에서 구동되는 소프트웨어의 성능에 대한 이슈가 대두 되고 있음

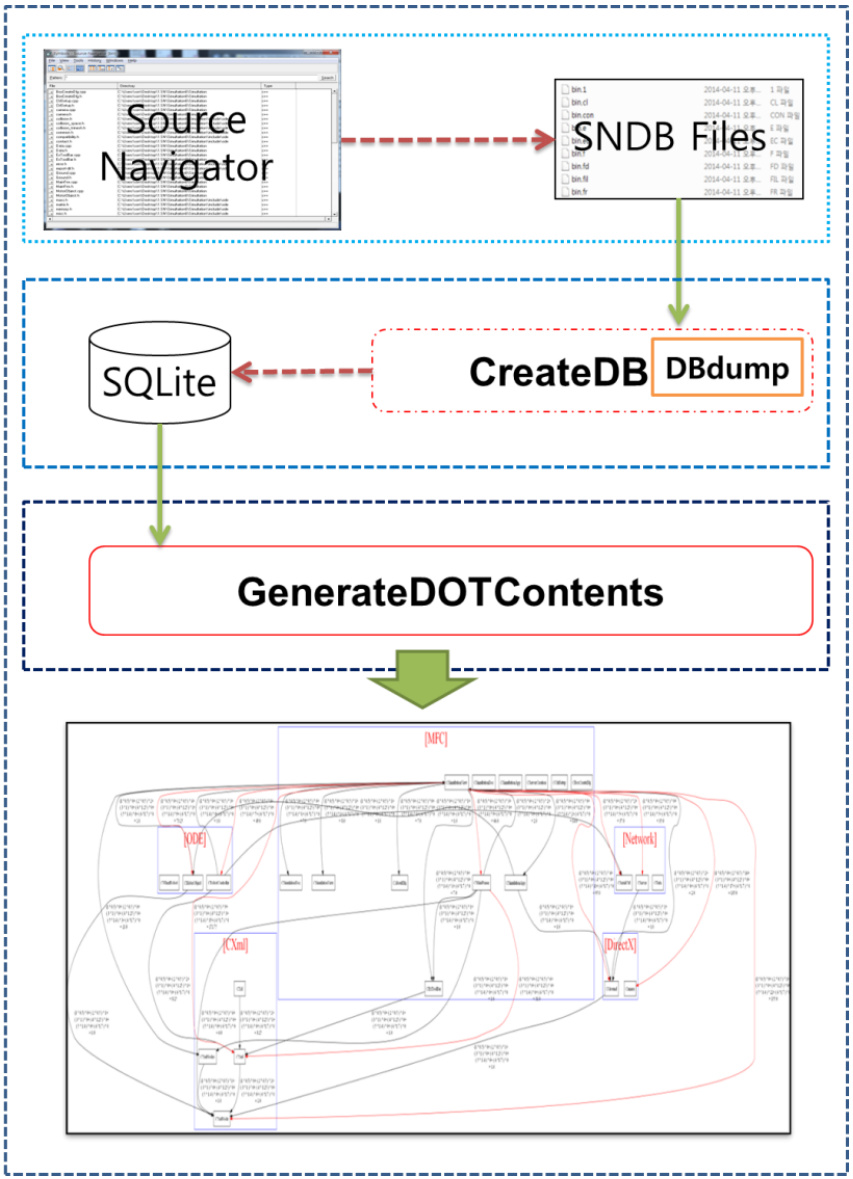


  - 소프트웨어 성능품질의 추출 및 가시화를 함으로서 국내 소프트웨어 의 성능품질의 경쟁력을 높여 보고자 함.

관련연구

1. 소프트웨어 가시화(Software Visualization)
2. 소프트웨어 성능(Software Performance)

소프트웨어 가시화



- 소프트웨어 가시화
  - 대상시스템을 분석하여 시스템의 구성요소와 관계 파악
  - 시스템의 구성요소들의 대한 관계와 구조를 시각적으로 표현

- 소프트웨어 가시화 수행이유
  - SW를 개발한 프로그래머의 도움 없이 SW에 대한 구조 혹은 기능의 이해를 돕기 위해 수행됨

코드의 복잡도 식별

->Refactoring issue

소프트웨어 성능

- 알고리즘
  - 중요한 명령어의 수행횟수 및 메모리 공간 점유율

- 컴퓨터 사이언스의 성능
  - 프로그램의 실행 시간
  - 메모리의 점유량
  - 전력 소모량.. 등등

- 소프트웨어 의 성능
  - 소프트웨어 가용률
  - 이식의 용의성
  - 사용성
  - 동시성 / 처리량
  - 반응속도

소프트웨어 성능

- 소프트웨어 가용률(Availability)
  - 소프트웨어가 해당 시점에서 요구사항에 따라 운영되는 확률

$$MTBF = \frac{Total\ Running\ Time - Total\ Faults\ Time}{Number\ of\ falults}$$

$$MTTR = \frac{Total\ Faults\ Time}{Number\ of\ Faults}$$

$$Availability = \frac{MTBF - MTTR}{MTBF} \times 100(\%)$$

MRBF : 평균 고장 발생간격
MTTR : 평균 복구시간

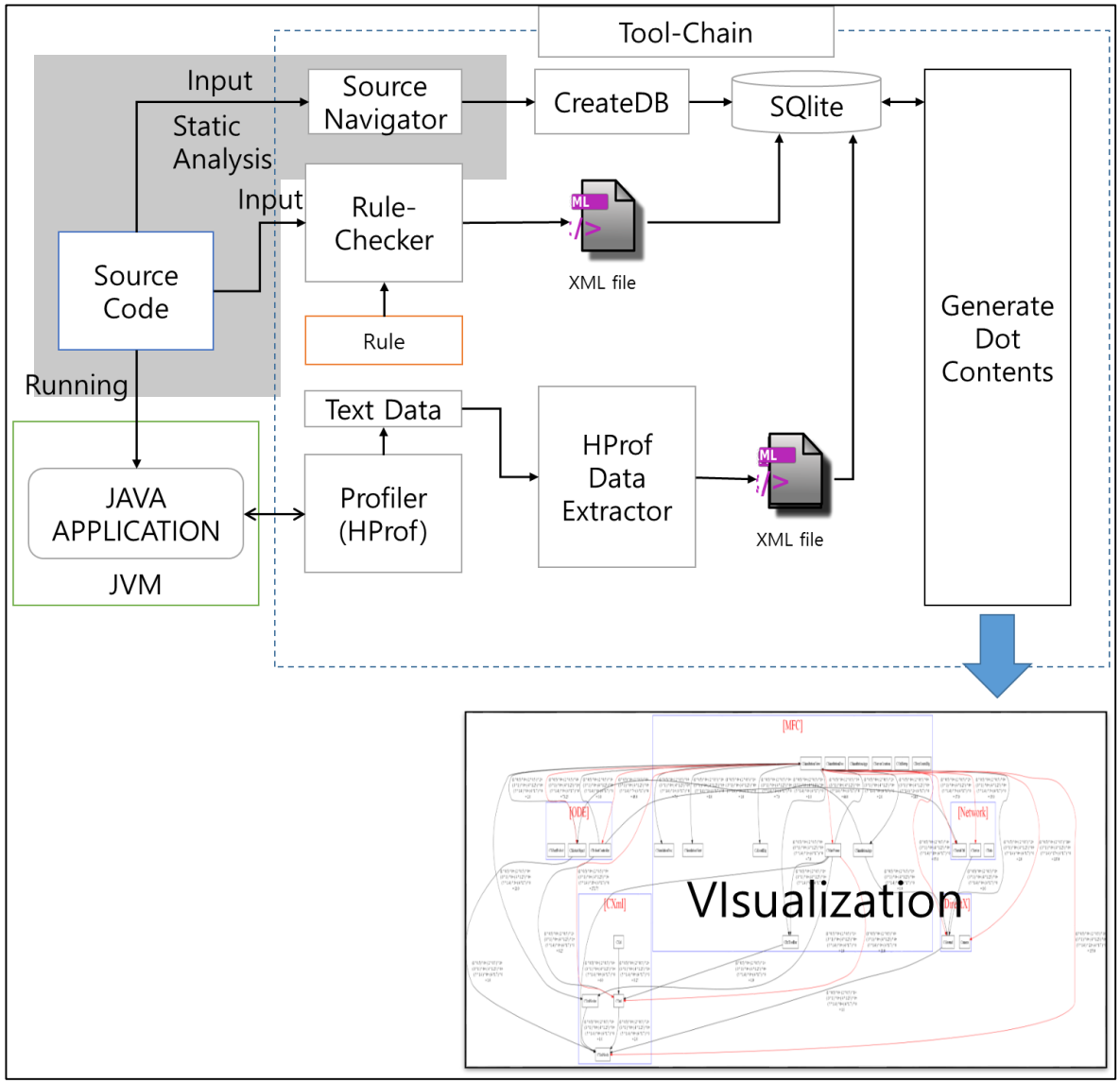
  - 소프트웨어가 지속적으로 동작하는 상황에서 측정이 되어야 하기 때문에 가시화 하기 어려움

- 동시성, 이식성, 사용성 등은 측정하기가 매우 어려움

소프트웨어의 반응속도 가시화

소프트웨어 성능 가시화를 위한 툴 체인 개발

성능 가시화 자동화 메커니즘

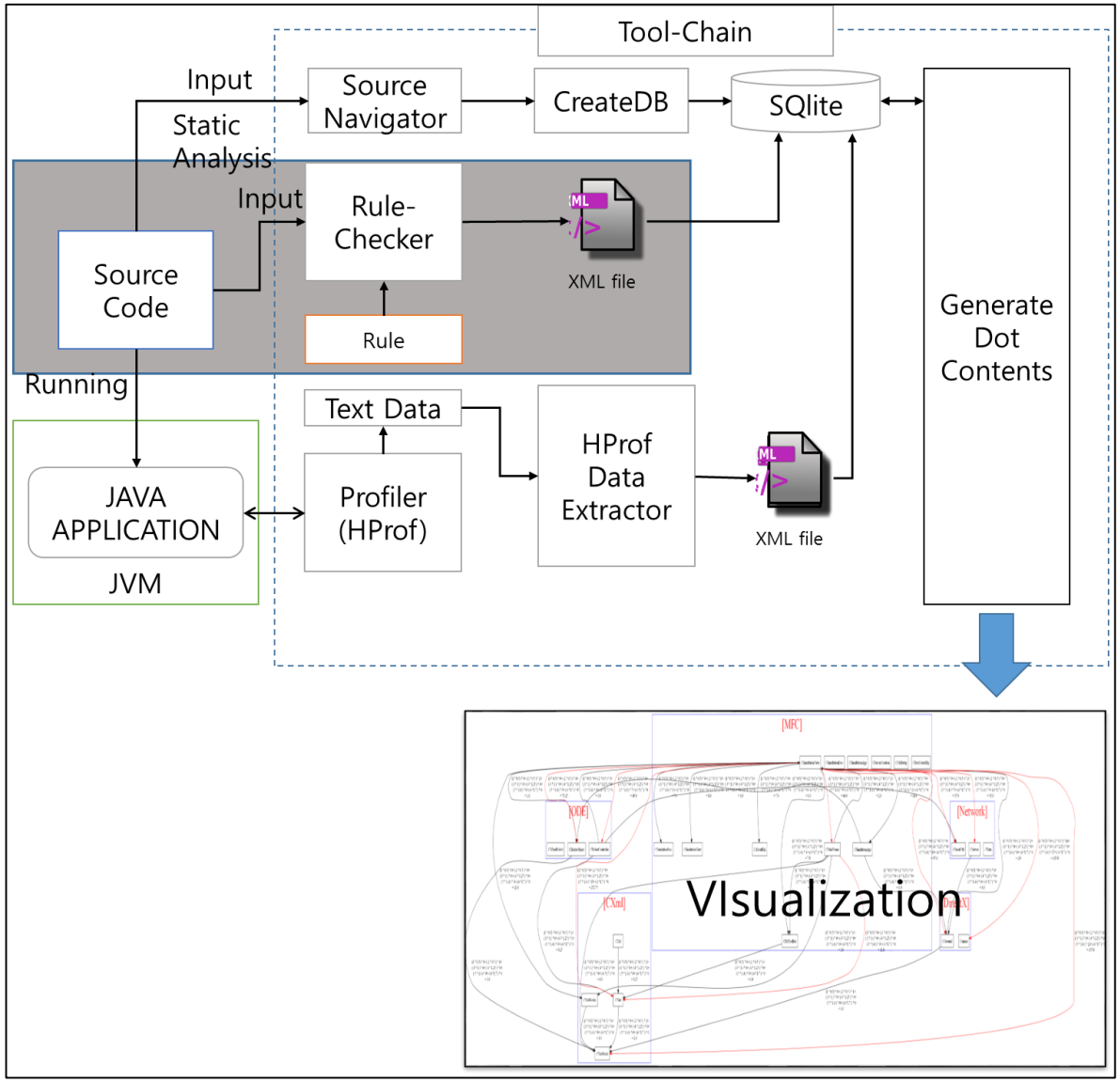


1단계: 소프트웨어의 정보추출

- 소프트웨어 성능저해요소 정보:

- Source Navigator에 소스코드입력
- SNDB파일을 추출
- SNDB(binary)파일 DBdump.exe에 입력
- 문자열로 변경, 추출

성능 가시화 자동화 메커니즘

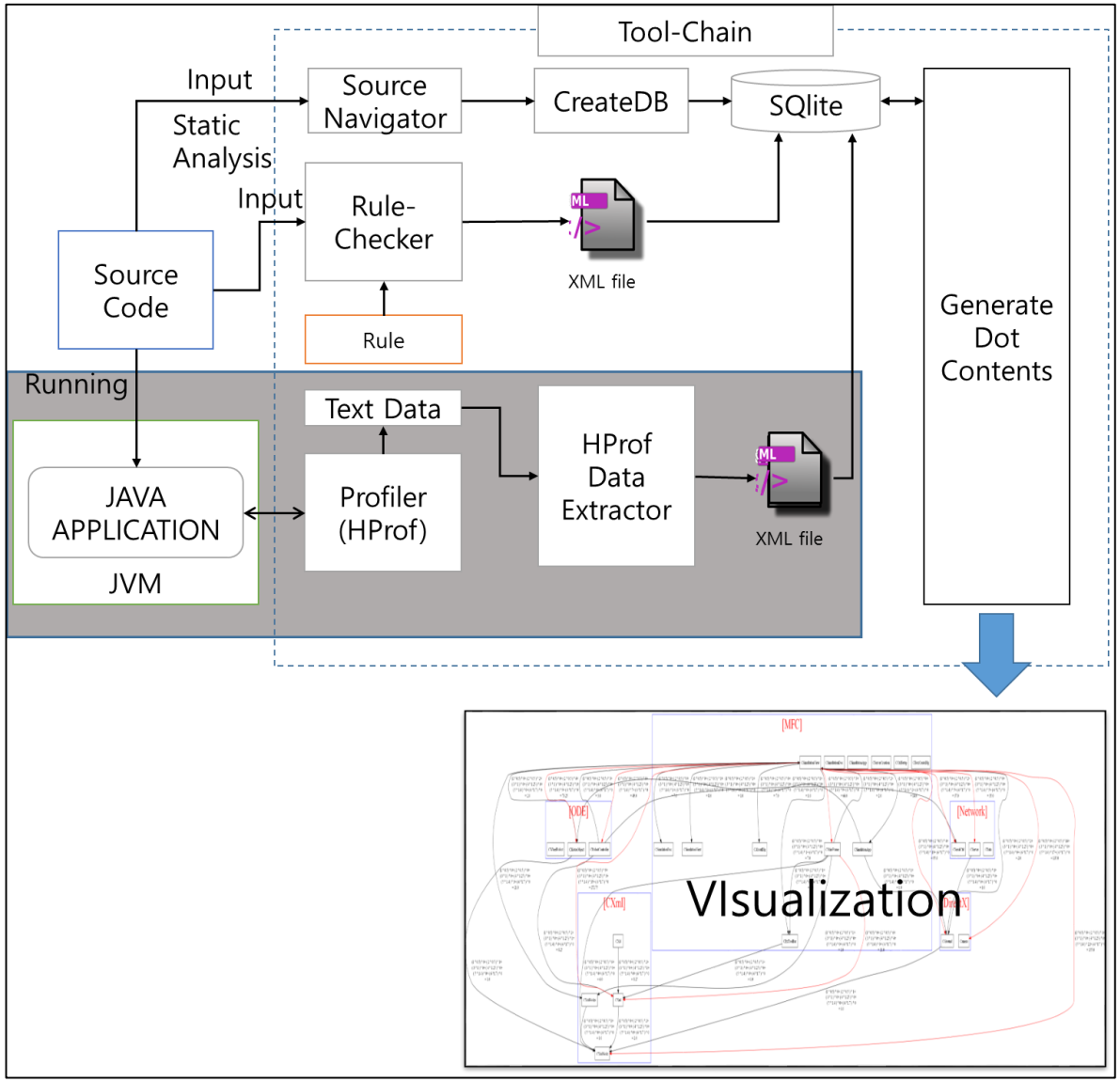


1단계: 소프트웨어의 정보추출

- 소프트웨어 성능저해요소 정보:

- 성능저해요소의 대한 패턴정의(Xpath)
- RuleChecker(PMD)에 적용
- XML정보로 추출

성능 가시화 자동화 메커니즘

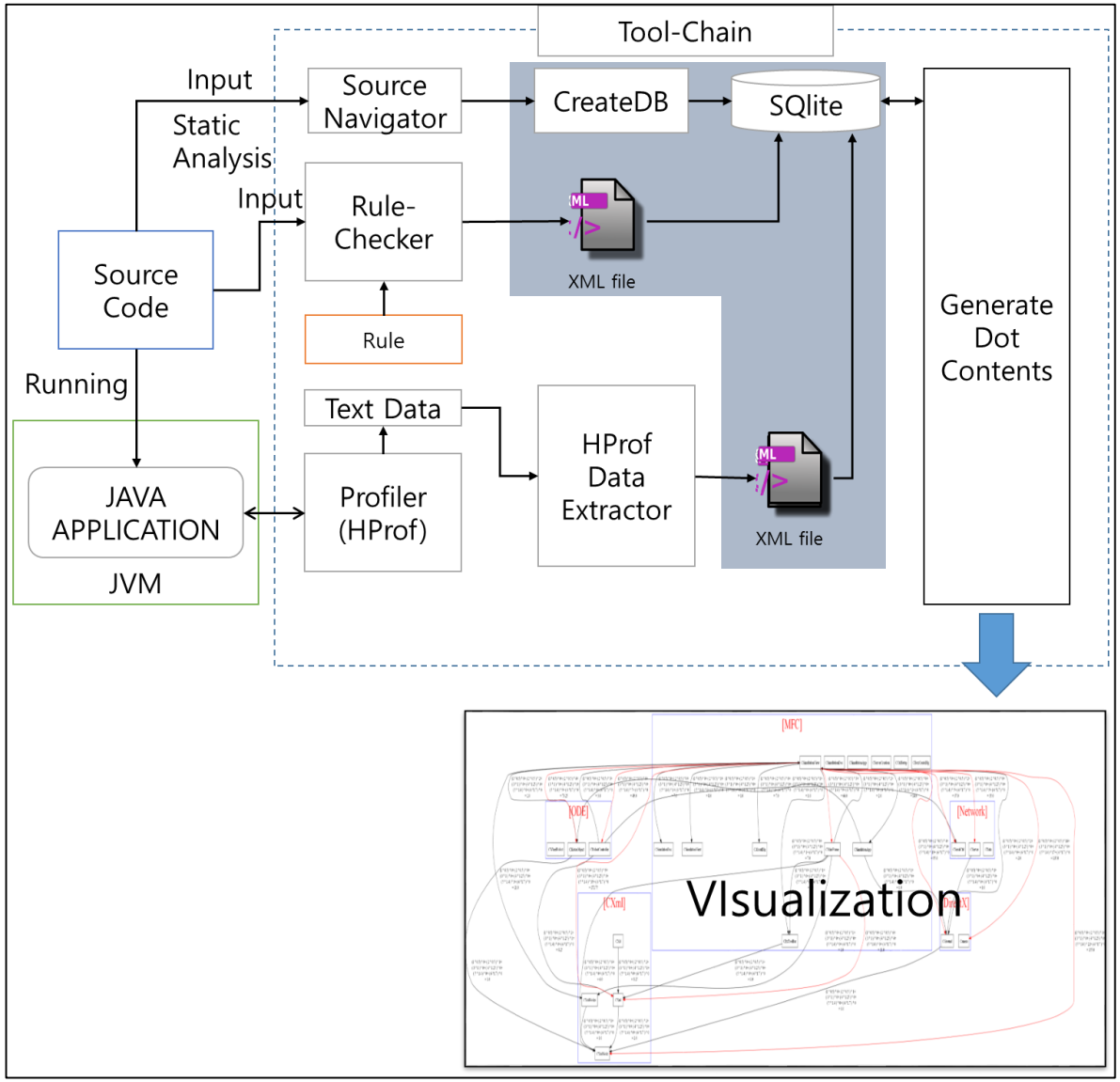


1단계: 소프트웨어의 정보추출

- 소프트웨어 성능정보:

- 소프트웨어를 프로파일러 (Hprof)와 함께 구동
- 메소드 단위의 구동시간과 호출횟수정보를 추출
- HprofDataExtrator를 통한 Xml형식의 정제정보 추출

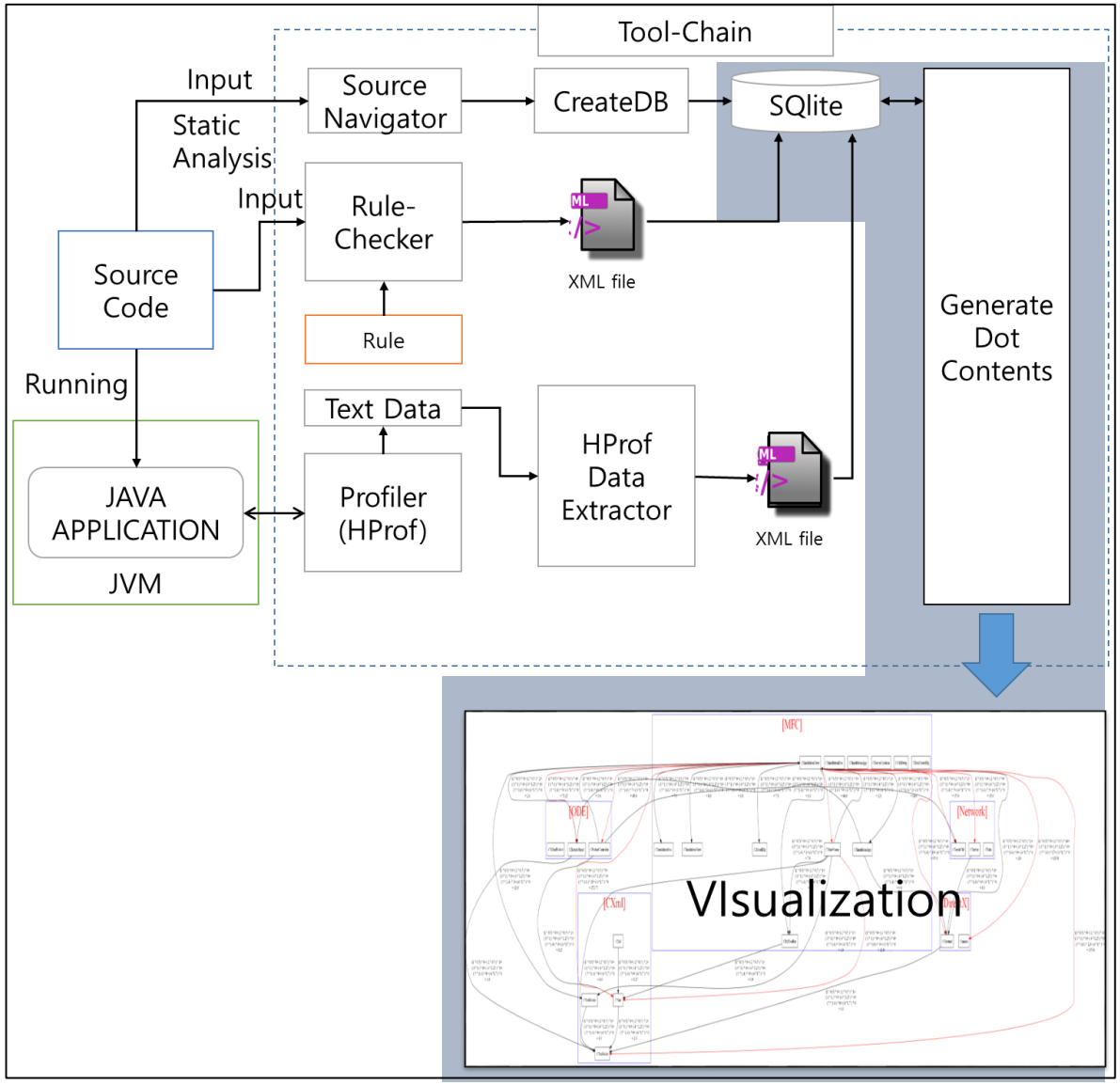
성능 가시화 자동화 메커니즘



2단계: 소프트웨어의 정보저장

- 소프트웨어 추출정보 (구조, 성능저해요소, 성능정보)를 SQLite에 저장

성능 가시화 자동화 메커니즘



3단계: 구조분석 및 품질지표 적용

- SQLite에 저장된 정보 →소프트웨어 품질지표 측정 및 저장(결합도, 소프트웨어 속도)

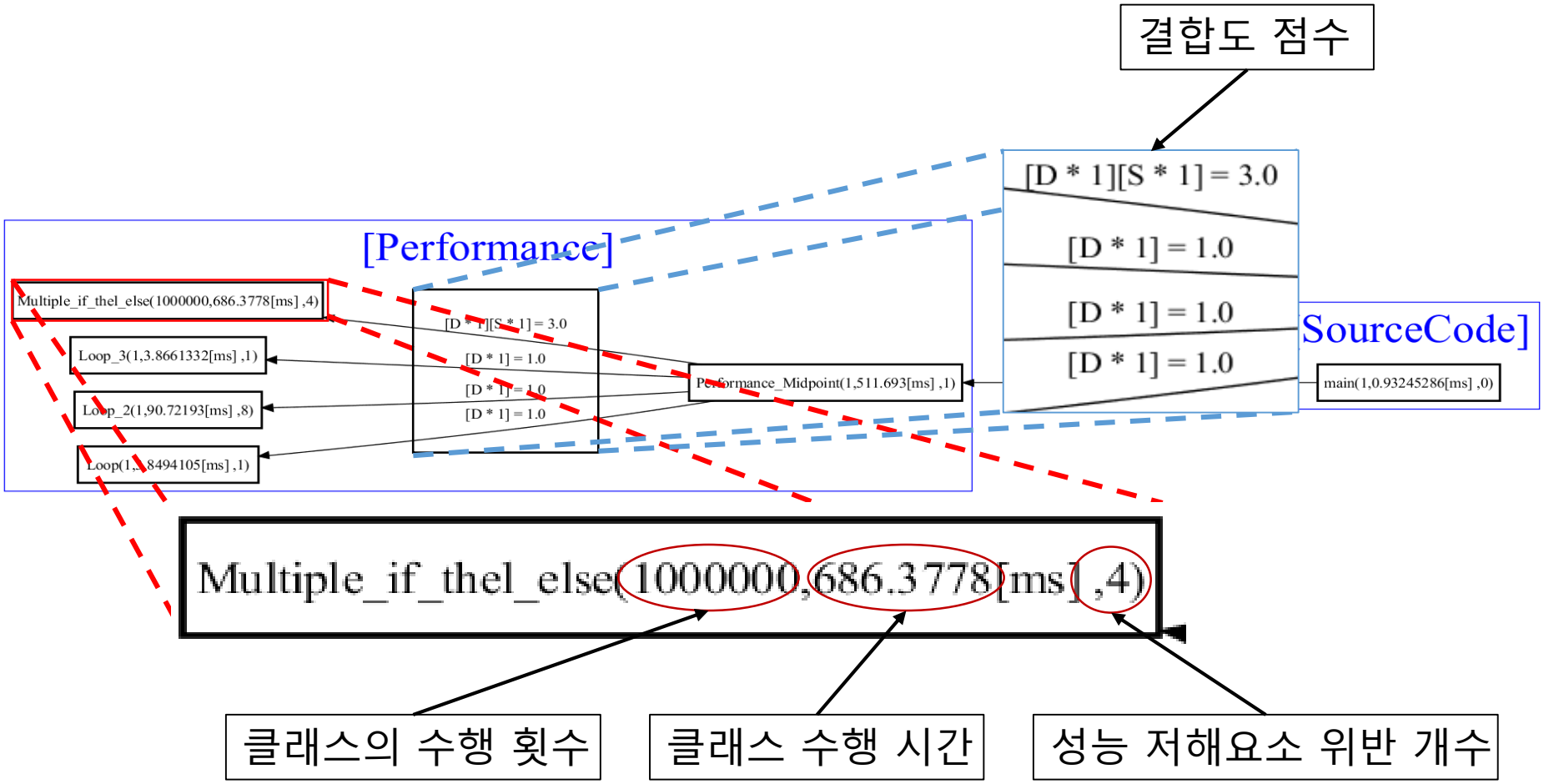
4단계: 가시화

- GenerateDotContents 사용 → 소스코드의 아키텍쳐, 정량화된 품질지표 수치의 가시화를 통한 DOT 스크립트로 생성

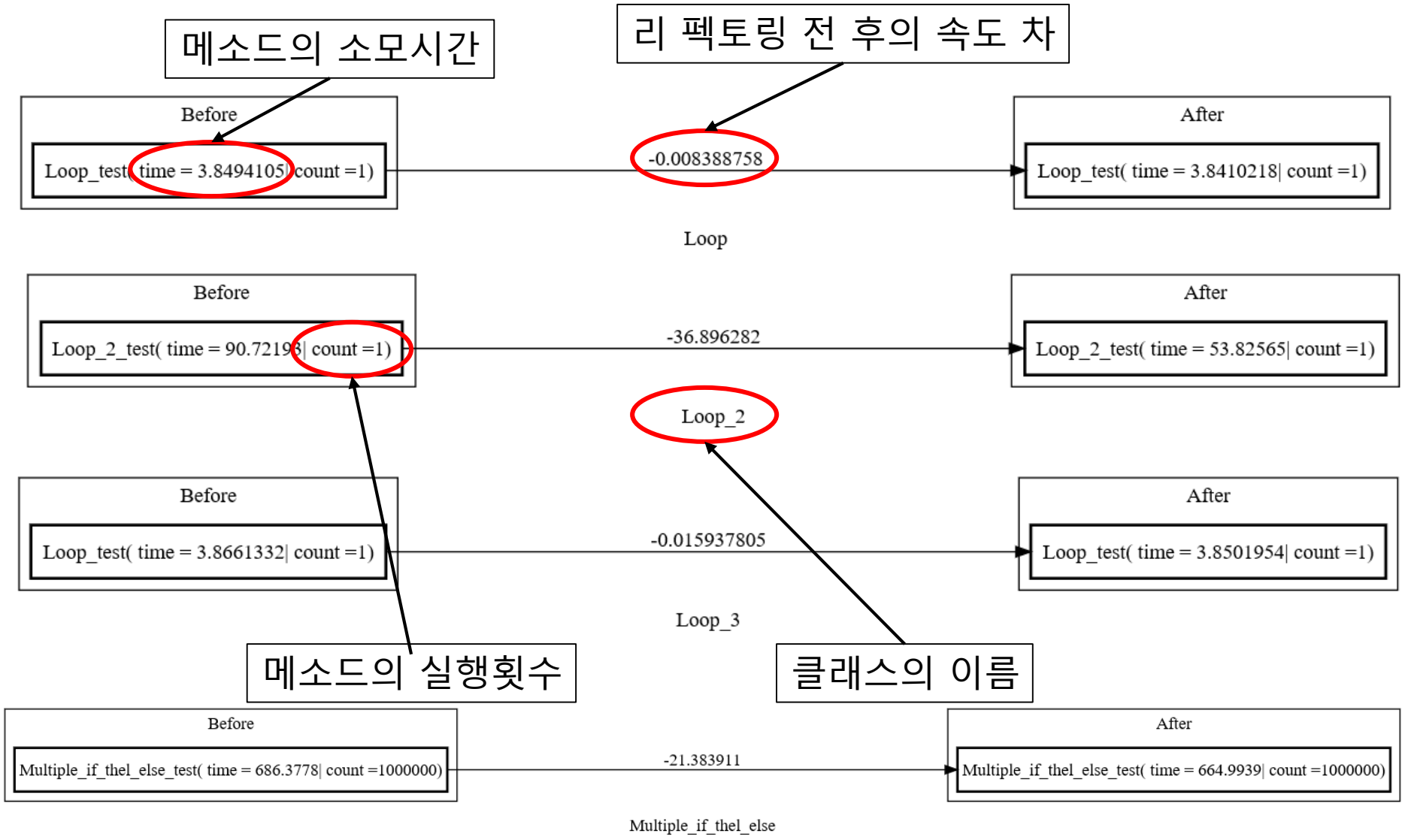
- graphViz도구를 통해 가시화

.

가시화 결과 그래프(Architecture view)



가시화 결과 그래프 (class 상세 성능 비교)



결론 및 향후연구

결론

- 역 공학 및 정적 분석을 통해
  - 소프트웨어의 재 사용성을 높이기 위한 결합도 측정 및 가시화
  - 소프트웨어 성능 저해요소의 대한 룰을 만들고

- 프로 파일러(Hprof)를 통한 동적 분석을 동시 수행

  - 정적 분석에서 얻을 수 없는 성능(메소드의 속도, 수행 횟수)에 대한 정보를 얻어 가시화 가능.

  - 한번의 Iteration이 돌 때 마다, 데이터베이스에 성능데이터를 누적하여 이전의 성능과 비교하여 볼 수 있음(클래스 상세 가시화 그래프)

향 후 연구

- 현재 C, C++에 대한 프로파일러의 적용이 되지 않아 Java 만 가능한 상태

  →C,C++언어에 대한 성능 가시화 적용

- 프로파일러(HProf)에서 리소스 점유 비율이 너무 작으면 측정이 되지 않는 문제 해결

- 향후 현재 각 언어에 의존되는 성능적인 문제를 찾아 추출하고 가시화 적용 예정

- 임베디드 시스템의 성능 가시화 중 프로 파일러를 통한 속도 측정 뿐만 아니라 실제 디바이스에서 작동할 때 사용 전력을 측정

  - 전력이 많이 소모되는 소스코드의 패턴을 찾아내고 실제 사용 전력과 함께 가시화 적용 예정