Pusan National University
Computer Science and Engineering
Technical Report 2025

2025년 전기 졸업과제 중간보고서

LLM을 활용한 SW Refactoring

팀명: L팩토링
분과명: 데이터베이스/그래픽스 및 비젼 (B)
202055518 김병현
202014511 박준하

지도교수 채흥석

목 차

1 요구조건 및 제약사항 분석에 대한 수정사항

1.1 기존 요구조건

목적	평가기준						
	LOC	CC	응집도	FAN IN	FAN OUT	Meaning Name	Duplicate Code
Analyzability	✓	✓	✓			✓	
Maintainability		✓		✓	✓		✓

표 1: 소프트웨어 평가 기준

착수보고서에서는 위 표에 따른 평가 기준을 세웠다.

이전 평가 기준(LOC, Cyclomatic Complexity, 코드 중복률 등 단일 정량적 지표 중심)은 코드의 근본적인 유지보수성, 실제 업무 환경에서의 품질 개선, 코드의 의미와 기능 보존, 실무 적용성을 충분히 설명하지 못하거나, 도메인 특수성 · 프로젝트 맥락 · 정성적 요소(이해용이성, 협업 난이도, 주석 품질) 등을 제대로 반영하지 못하는 한계가 있다.

1.2 요구조건에 대한 수정사항

다른 방안으로 Maintainability Index는 코드 유지보수성의 단일 수치적 평가지표로 의미가 있으나, 코드 스멜, 중복, 모듈화, 설계 패턴, 정성적 협업 역량 등 실질 품질은 충분히 반영하지 못한다.

보다 객관적이고 신뢰도 높게 평가하기 위해 SonarCloud 기반의 정적 분석을 도입하였다. SonarCloud는 국제 표준(ISO/IEC 25010 등)에 부합하는 코드 품질 규칙을 내장하고 있으며, LOC, Cyclomatic Complexity, 코드 스멜, 중복 코드, 함수/파일 구조, 주석 품질 등 다양한 품질 지표를 자동으로 계산해준다. 이를 통해 리팩토링 전후의 품질 변화, 기술 부채 감소 효과, 코드 유지보수성 개선 등을 일관적이고 재현 가능하게 평가할 수 있다.

또한, SonarCloud는 코드 품질을 다음 세 가지 관점에서 다섯 단계로 분석한다.

- 평가 관점: Security, Reliability, Maintainability
- 등급 기준: Block, High, Medium, Low, Info

분석 결과는 직관적인 대시보드와 상세 리포트로 제공하여 실험의 신뢰도를 높이고, 과제에 요구되는 과학성과 투명성을 동시에 확보할 수 있다는 장점이 있다.

1.3 제약 사항 분석

기존에 SonarQube를 사용하려고 했으나. C 언어 정적 분석 시 다음과 같은 한계가 있다.

- 유료 라이선스 필요: 기본 오픈소스 버전이나 무료 플랜에서는 C 정적 분석 기능이 제한되며, Developer 에디션과 같은 유료 라이선스가 반드시 필요하다.
- 설치 · 운영 복잡성: 온프레미스 방식으로 서버 구축, 데이터베이스 관리, 플러그인 설치 등 별도의 인프라 관리가 요구된다.
- 업데이트 및 연동 제약: 무료 버전에서는 최신 분석 규칙 업데이트, CI/CD 통합, 다양한 언어 및 규모에 대한 지원이 상대적으로 더디거나 제한적일 수 있다.

이러한 문제점을 해결하기 위해 본 과제에서는 SonarCloud(SonarCloud)를 사용하기로 결정하였다.

2 설계 상세화 및 변경 내역

본 과제에서는 C언어로 작성된 단일 파일과 다중 디렉토리로 구성된 프로젝트 단위 코드를 대상으로 리팩토링을 수행하고, 해당 결과를 SonarCloud 기반의 정적 분석 도구를 활용하여 정량적으로 평가하고자 한다.

기존에는 Minix3, Git, Linux 커널(Linux Kernel)등 오픈소스 프로젝트를 활용할 계획이었으나, 실험의 범위를 다양화하고 리팩토링 품질을 크기별 · 복잡도별로 상세히 비교하기 위해, 다양한 형태의 코드셋을 포함하여 실험 대상으로 선정하였다.

2.1 단일 파일 기반 코드

C언어로 작성된 100줄 미만의 단일 소스코드 파일

- 코드 내에 Security, Reliability, Maintainability 부문에서 코드 스멜이 존재한다.

Mongoose의 amalgamation 프로젝트

- 전체 라이브러리를 하나의 큰 C 파일로 통합한 구조로, 함수 수가 많고 코드 길이가 길다.

2.2 프로젝트 단위 코드

오픈소스 프로젝트 (Minix3 등)

- 여러 개의 디렉토리와 다수의 소스파일로 구성된다.

이와 같이, 다양한 규모와 복잡도 수준의 코드셋을 활용하여 각 LLM 모델(GPT-4.1, Claude 3.7 Sonnet, Gemini 2.5 Pro)의 리팩토링 성능을 비교·분석하고, 그 결과를 기반으로 최종 과제 수행 시 사용할 모델을 선정한다.

3 갱신된 과제 추진 계획

5월

- LLM, 리팩토링 도구 관련 선행연구 조사
- C 언어 리팩토링 필요성과 평가 기준 정리
- GitHub 오픈소스 코드 탐색
- 착수보고서 작성 및 제출

6월

- LLM 프롬프트 실험 설계 및 테스트
- 리팩토링 평가 기준 확정
- 평가 도구 선정

7월

- C코드에 대해 리팩토링 수행
- 기능 테스트 자동화
- 리팩토링 전후 코드 비교 정리
- 간단한 실험: 구조적 개선 여부 확인
- agent 기능 개발

8월

- 리팩토링 대상 코드셋 확장
- 본격적인 리팩토링 실험 수행 및 결과 기록
- 코드 품질 지표 평가 수집
- agent 기능 개발

9월

- 실험 결과 분석 및 통계 정리
- agent 배포 및 수정
- 최종 보고서 작성
- 프로그램 시연 영상 또는 실행 예시

4 구성원별 진척도

김병현

- Maintainability Index가 리팩토링 성능 지표로 타당한지 분석
- 다양한 LLM (GPT, Claude, Gemini) 비교 분석

박준하

- SonarCloud가 리팩토링 성능 지표로 타당한지 분석
- 리팩토링 결과에 대해 정적분석 도구(SonarCloud)로 품질 평가

5 보고 시점까지의 과제 수행 내용 및 중간 결과

5.1 파일 규모에 따른 리팩토링 분석

아래 표는 코드 스멜이 있는 100줄 미만의 간단한 코드에 대한 분석 결과이다.

	Security	Reliability	Maintainability
Blocker	0	1	0
High	1	0	1
Medium	0	1	0
Low	0	0	1
Info	0	0	0

표 2: 원본 소스 코드의 SonarCloud 분석 결과

GPT 4.1은 2개의 코드 스멜을 제거하였다.

	Security	Reliability	Maintainability
Blocker	0	2	0
High	0	0	0
Medium	0	0	0
Low	0	0	1
Info	0	0	0

표 3: GPT 4.1로 리팩토링한 코드의 SonarCloud 분석 결과

	Security	Reliability	Maintainability
Blocker	0	0	0
High	0	0	0
Medium	0	0	1
Low	0	0	0
Info	0	0	0

표 4: Claude 3.7 Sonnet로 리팩토링한 코드의 SonarCloud 분석 결과

Claude 3.7 Sonnet은 4개의 코드 스멜을 제거하였다.

	Security	Reliability	Maintainability
Blocker	0	0	0
High	0	0	0
Medium	0	0	0
Low	0	0	0
Info	0	0	0

표 5: Gemini 2.5 pro로 리팩토링한 코드의 SonarCloud 분석 결과

Gemini 2.5 pro는 코드 스멜을 모두 제거하였다.

Mongoose는 약 25k 라인을 가진 amalgamation 프로젝트가 존재한다. 아래 표는 LLM을 이용해 리팩토링하고 SonarCloud로 분석한 결과이다.

	Security	Reliability	Maintainability
Blocker	0	1	17
High	0	2	123
Medium	0	2	62
Low	0	1	337
Info	0	0	6

표 6: mongoose 원본 소스 코드의 SonarCloud 분석 결과

	Security	Reliability	Maintainability
Blocker	0	1	3
High	0	2	97
Medium	0	0	45
Low	0	1	244
Info	0	0	6

표 7: Gemini 2.5 pro로 리팩토링한 mongoose의 SonarCloud 분석 결과

Reliability에서 2개 제거, Maintainability에서 150개의 코드 스멜을 제거하였다.

5.2 프로젝트 규모에 따른 리팩토링 분석

아래는 Minix3에서 kernel 디렉토리만 리팩토링하고 분석한 결과이다.

	Security	Reliability	Maintainability
Blocker	0	0	0
High	0	0	8
Medium	0	0	117
Low	0	0	23
Info	0	0	0

표 8: Minix3 kernel/ 원본 소스 코드의 SonarCloud 분석 결과

	Security	Reliability	Maintainability
Blocker	0	0	0
High	0	0	10
Medium	0	0	81
Low	0	0	27
Info	0	0	0

표 9: GPT 4.1로 리팩토링한 Minix3 kernel/의 SonarCloud 분석 결과

GPT 4.1은 Maintainability에서 30개의 코드 스멜을 제거하였다.

	Security	Reliability	Maintainability
Blocker	0	0	0
High	0	0	9
Medium	0	0	113
Low	0	0	23
Info	0	0	0

표 10: Claude 3.7 Sonnet로 리팩토링한 Minix3 kernel/의 SonarCloud 분석 결과

Claude 3.7 Sonnet은 Maintainability에서 3개의 코드 스멜을 제거하였다.

	Security	Reliability	Maintainability
Blocker	0	0	0
High	0	0	8
Medium	0	0	86
Low	0	0	20
Info	0	0	0

표 11: Gemini 2.5 pro로 리팩토링한 Minix3 kernel/의 SonarCloud 분석 결과

Gemini 2.5 pro는 Maintainability에서 34개의 코드 스멜을 제거하였다.

아래는 Minix3의 전체 프로젝트를 리팩토링하고 분석한 결과이다.

	Security	Reliability	Maintainability
Blocker	0	0	12
High	1	10	131
Medium	0	3	775
Low	0	0	387
Info	0	0	0

표 12: Minix3 원본 소스 코드의 SonarCloud 분석 결과

	Security	Reliability	Maintainability
Blocker	0	0	12
High	1	8	137
Medium	0	2	511
Low	0	0	390
Info	0	0	0

표 13: Gemini 2.5 pro로 리팩토링한 Minix3의 SonarCloud 분석 결과

Gemini 2.5 pro는 Reliability에서 3개 제거, Maintainability에서 255개의 코드 스멜을 제거하였다.