Harness: LLM 코드 에이전트 산출물 품질 향상을 위한 구조화된 사전 구성 프레임워크

15개 소프트웨어 엔지니어링 과제를 활용한 대조 실험

황민호 (Minho Hwang)
revfactory@gmail.com

2026년 3월 5일

초록 (Abstract)

Claude Code와 같은 대규모 언어 모델(LLM) 기반 코드 에이전트는 기능적으로 동작하는 소프트웨어를 생성하는 데 뛰어난 능력을 보여주고 있다. 그러나 아키텍처 설계, 테스트 커버리지, 확장성 등 산출물의 품질은 구조적 가이던스의 제공 여부에 따라 크게 달라진다. 본 논문은 구조화된 프로젝트 스캐폴딩을 통해 LLM 코드 에이전트의 산출물을 체계적으로 향상시키는 사전 구성 프레임워크인 **Harness**를 제안한다. 우리는 세 가지 난이도(기본, 고급, 초고난이도)에 걸친 15개 소프트웨어 엔지니어링 과제에 대해 통제된 A/B 실험을 수행하고, 10개 품질 차원에 걸쳐 산출물을 평가하였다. 실험 결과, Harness는 평균 품질 점수를 49.5점에서 79.3점(100점 만점)으로 끌어올려 **60%의 품질 향상**을 달성하였다. 특히 중요한 발견은, 과제 난이도와 효과 사이에 강한 양의 상관관계가 존재한다는 점이다: 기본 과제는 +23.8점, 고급 과제는 +29.6점, 초고난이도 과제는 +36.2점의 개선을 보여, 기본에서 초고난이도까지 **개선 폭이 52% 증가**하였다. 이러한 결과는 구조화된 사전 구성이 과제 복잡도가 증가할수록 더욱 필수적이 되며, LLM 코드 에이전트의 핵심 병목이 기능적 역량이 아니라 구조적 조직화에 있음을 시사한다.

키워드: LLM 코드 에이전트, 소프트웨어 엔지니어링, 코드 품질, AI 보조 개발, 프롬프트 엔지니어링, 구조화된 생성

1. 서론

LLM 기반 코드 에이전트의 등장은 소프트웨어 엔지니어링 워크플로를 근본적으로 변화시키고 있다. Claude Code [1], GitHub Copilot Workspace [2], Cursor [3] 등의 도구는 자연어 설명만으로도 완전한 소프트웨어 프로젝트를 생성할 수 있다. 이러한 도구들은 기능적으로 동작하는 코드를 생성하는 데는 탁월하지만, 아키텍처, 테스트, 문서화, 확장성 등 비기능적 품질 속성에서는 일관성이 떨어지는 것으로 관찰되고 있다 [4].

실무적 관점에서 이는 LLM 코드 에이전트가 작동하는 프로토타입을 빠르게 만들어낼 수 있지만, 생성된 코드가 프로덕션 환경에서 요구되는 구조적 엄밀성을 갖추지 못하는 경우가 많다는 것을 의미한다. AI가 생성한 코드를 채택한 개발자들은 모놀리식 산출물을 유지보수 가능한 아키텍처로 재구조화하고, 누락된 테스트 스위트를 추가하며, 에러 처리를 개선하는 데 상당한 시간을 소비하게 되는데, 이는 AI 보조 개발이 약속하는 생산성 향상을 부분적으로 상쇄하는 결과를 초래한다.

이러한 품질 격차는 아키텍처 결정이 코드베이스 전체에 연쇄적으로 영향을 미치는 복잡한 다중 컴포넌트 시스템에서 특히 두드러진다. 예를 들어, "SQL 쿼리 엔진을 만들어라"라는 프롬프트를 받은 LLM 코드 에이전트는 기능적으로는 올바르지만, 프로덕션 수준의 시스템을 특징짓는 파서-플래너-실행기 분리 구조가 결여된 모놀리식 구현을 생성할 수 있다.

우리는 이러한 품질 격차가 LLM의 지식이나 추론 능력의 한계에서 비롯되는 것이 아니라, **구조적 가이던스의 부재**—숙련된 엔지니어가 모든 프로젝트에 암묵적으로 가져오는 프로젝트 관례, 아키텍처 패턴, 품질 기대치의 부재—에서 기인한다는 가설을 세웠다.

이 가설을 검증하기 위해 우리는 Harness를 개발하였다. Harness는 숙련된 엔지니어가 프로젝트에 가져오는 것과 동일한 종류의 맥락적 스캐폴딩을 LLM 코드 에이전트에 제공하는 프레임워크이다. 에이전트의 구현 선택을 제약하기보다는, 엔지니어링 리더가 새로운 팀원에게 코딩 시작 전 브리핑하듯이, 산출물의 구조와 품질에 대한 기대치를 설정하는 방식으로 작동한다.

1.1 기여

본 논문의 주요 기여는 다음과 같다:

1. **Harness 프레임워크**: 구조화된 프로젝트 스캐폴딩을 통해 LLM 코드 에이전트를 사전 구성하는 체계적 접근법 (§3).
2. **15개 과제 대조 실험**: 세 가지 난이도에 걸친 10차원 품질 평가를 포함하는 포괄적 A/B 평가 (§4).
3. **난이도-효과 상관관계**: 사전 구성의 효과가 과제 복잡도에 비례하여 증가한다는 실증적 근거 (§5).
4. **차원별 분석**: 구조화된 사전 구성에 의해 가장 크게 영향받는 품질 차원의 식별 (§5.3).

1.2 동기

숙련된 소프트웨어 엔지니어가 새로운 프로젝트에 합류하는 상황을 생각해보자. 코드를 작성하기 전에 그들은 보통

아키텍처 가이드라인을 검토하고, 파일 구조 관례를 이해하며, 기존 디자인 패턴을 학습하고, 테스트 요구사항을 확인한다. LLM 코드 에이전트에게는 이러한 맥락적 스캐폴딩이 부재하다. 방대한 지식을 보유하고 있지만 프로젝트 특화된 구조적 가이던스 없이 각 과제에 접근한다. Harness는 AI 에이전트를 위한 "프로젝트 온보딩"에 해당하는 것을 제공함으로써 이 문제를 해결한다.

2. 관련 연구

LLM이 생성한 코드의 품질을 향상시키는 방법에 대한 연구는 상당한 관심을 받아왔다. 기존 접근법은 크게 네 가지 흐름으로 분류할 수 있다: 품질 평가, 프롬프트 엔지니어링, 멀티 에이전트 시스템, 프로젝트 스캐폴딩. 각 흐름을 순서대로 검토하고 이들에 대한 본 연구의 위치를 설정한다.

2.1 LLM 코드 생성 품질

최근 연구들은 다양한 관점에서 LLM이 생성한 코드의 품질을 검토하였다. Chen et al. [5]은 HumanEval 벤치마크를 통해 기능적 정확성을 평가하였고, Yetiştiren et al. [6]은 유지보수성과 복잡도를 포함한 코드 품질 메트릭을 분석하였다. 이들 연구는 일관되게 LLM이 기능적으로 올바른 코드를 생성하지만 소프트웨어 엔지니어링 모범 사례에서는 취약점을 보인다는 결론에 도달한다.

2.2 코드를 위한 프롬프트 엔지니어링

사고의 사슬(Chain-of-Thought) 프롬프팅 [7]과 퓨샷 예시 [8]는 코드 생성 정확성에서 개선을 보여주었다. 그러나 이러한 기법들은 개별 코드 스니펫에 초점을 맞추며, 프로젝트 수준의 품질 속성에는 적용되지 않는다.

2.3 멀티 에이전트 소프트웨어 엔지니어링

MetaGPT [9], ChatDev [10] 등의 프레임워크는 전문화된 에이전트들이 각기 다른 개발 측면을 담당하는 멀티 에이전트 접근법을 탐구하였다. 본 연구는 실행 시점의 멀티 에이전트 오케스트레이션 대신 구조화된 사전 구성을 통해 단일 에이전트의 산출물을 향상시킨다는 점에서 차별화된다.

2.4 프로젝트 스캐폴딩

Yeoman, Create React App 등 전통적인 스캐폴딩 도구는 인간 개발자를 위한 시작 템플릿을 제공한다. Harness는 이 개념을 AI 에이전트에 적용하여 파일 구조뿐만 아니라 디자인 패턴 레퍼런스, 품질 가이드라인, 역할 기반 작업 분해 명세까지 제공한다.

본 연구는 네 가지 흐름의 요소를 고유하게 결합한다: 프로젝트 수준의 구조적 가이던스(스캐폴딩), 전문 도메인 지식 임베딩(프롬프트 엔지니어링), 역할 기반 분해 정의(멀티 에이전트 원칙), 포괄적 품질 차원 평가(품질 평가). 이러한 종합이 품질 문제의 한 측면만을 다루는 기존 접근법과 Harness를 구별짓는다.

3. Harness 프레임워크

3.1 개요

Harness는 프로젝트의 `.claude/` 디렉토리에 위치하는 사전 구성 프레임워크이다. 과제 실행이 시작되기 전에 LLM 코드 에이전트에게 세 가지 유형의 구조적 가이던스를 제공한다(그림 1). Harness의 핵심 통찰은, LLM 코드 에이전트가 소프트웨어 엔지니어링 패턴, 알고리즘, 모범 사례에 대한 광범위한 지식을 이미 보유하고 있지만, 어떤 패턴을 적용하고 어떻게 조직화할지를 결정하는 프로젝트 특화 맥락이 부족하다는 것이다. Harness는 에이전트의 기존 지식을 활성화하고 방향을 설정하는 구조화된 "프로젝트 브리프"를 제공함으로써 이 격차를 해소한다.

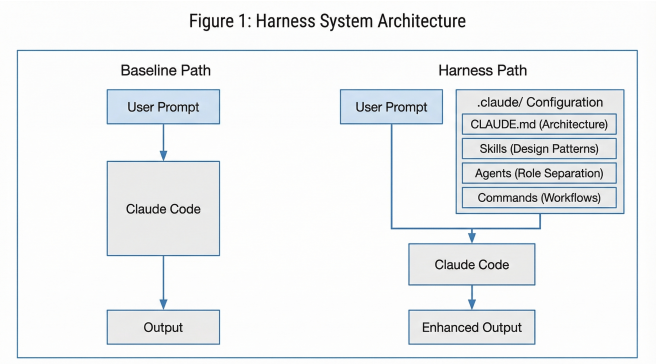

그림 1: Harness 시스템 아키텍처. Baseline 경로는 사용자 프롬프트만을 제공하는 반면, Harness 경로는 아키텍처 가이드라인, 디자인 패턴 스킬, 역할 기반 에이전트 정의를 포함하는 구조화된 구성으로 프롬프트를 보강한다.

3.2 구성 요소

3.2.1 CLAUDE.md — 아키텍처 청사진

Harness의 가장 영향력 있는 구성 요소는 인간 설계자와 AI 에이전트 사이의 중심적 아키텍처 계약으로 기능하는 CLAUDE.md 파일이다. `CLAUDE.md` 파일은 목표 아키텍처, 파일 구조, 네이밍 관례, 프로젝트 특화 규칙을 정의하며, LLM 에이전트의 산출물에 대한 주요 구조적 앵커 역할을 수행한다:

```
# SQL 쿼리 엔진
## 아키텍처 (3단계 파이프라인)
SQL 문자열 → [Parser] → AST → [Planner]
  → Plan → [Executor] → ResultSet
## 파일 구조
src/
  lexer.js    - SQL 토크나이저
  parser.js   - 재귀 하강 파서
  planner.js  - 논리 → 물리 계획 변환
  executor.js - 쿼리 실행기
```

3.2.2 Skills — 디자인 패턴 레퍼런스

Skills는 YAML 프론트매터를 갖춘 `SKILL.md` 파일로 저장되는 전문 지식 문서이다. 알고리즘 특화 구현 패턴, 자료 구조 정의, 정확성 기준을 제공하며, 과제 요구사항에 따라 맥락적으로 활성화된다. 예를 들어, Raft 합의 알고리즘을 구현할 때 해당 Skill 문서에는 정확한

AppendEntries와 RequestVote RPC 명세, 리더 선출 타임아웃 범위, 로그 압축 규칙이 포함된다. 이 수준의 상세함은 LLM이 불완전할 수 있는 훈련 데이터에 의존하는 것을 방지하고, 핵심 구현에서 알고리즘의 정확성을 보장한다.

3.2.3 Agents — 역할 분해

Agent 정의는 복잡한 과제를 전문화된 역할로 분해하는 방법을 명시한다. 각 에이전트 정의에는 책임 범위의 명확한 기술, 예상 산출물, 선행 의존성, 사용할 도구, 명시적 품질 기준이 포함된다. 예를 들어, LSP 서버 케이스(Case 015)에서는 증분 파서 빌더, 자동완성 프로바이더 빌더, 진단 빌더, 전송 계층 빌더의 네 가지 에이전트가 정의된다. 각 에이전트는 잘 정의된 인터페이스 위에서 동작하여 병렬 개발과 깔끔한 관심사 분리를 가능하게 한다. 이러한 역할 기반 분해는 숙련된 엔지니어링 팀이 복잡한 프로젝트를 조직화하는 방식을 반영한다.

3.3 설계 원칙

Harness 프레임워크는 세 가지 설계 원칙을 따른다:

P1: 관심사의 분리. 아키텍처 결정(CLAUDE.md), 구현 패턴(Skills), 과제 분해(Agents)는 독립적으로 관리된다.

P2: 점진적 공개. SKILL.md 파일은 고수준 가이던스(≤500줄)를 포함하고, 상세한 패턴은 `references/` 하위 디렉토리에 위치한다.

P3: 선언적 우선. Harness는 산출물이 어떤 모습이어야 하는지(what)를 명시하되, 어떻게 생성할지(how)는 지정하지 않아 LLM의 구현 유연성을 보존한다.

4. 실험 설계

4.1 과제 선정

Harness의 효과를 포괄적으로 평가하기 위해, 세 가지 난이도에 걸친 15개 소프트웨어 엔지니어링 과제를 설계하였다. 과제는 단순한 CRUD API부터 심층적 알고리즘 지식이 필요한 복잡한 분산 시스템까지 소프트웨어 엔지니어링의 넓은 스펙트럼을 포괄하도록 선정되었다. 각 난이도는 LLM 에이전트의 서로 다른 역량을 시험한다:

기본 과제 (001–005)는 REST API, 동시성 버그 수정, 코드 리팩토링, 문서화, CLI 도구 등 단일 관심사 프로젝트를 포함한다. 이러한 과제는 일반적인 프로그래밍 지식으로 충분히 해결할 수 있으며, 최소한의 아키텍처 설계만 요구한다.

고급 과제 (006–010)는 상태 관리 비교, 언어 인터프리터, 이벤트 기반 마이크로서비스, SQL 쿼리 엔진, 협업 에디터 등 다중 컴포넌트 시스템을 포함한다. 이러한 과제는 의도적인 아키텍처 선택과 전문화된 알고리즘 구현을 요구한다.

초고난이도 과제 (011–015)는 가장 도전적인 시나리오를 대표한다: 분산 합의(Raft), 리액티브 연산 엔진, 바이트코드 가상 머신, 이벤트 소싱 프레임워크, 그리고 Language Server Protocol 구현. 이러한 과제는 깊은 도메인 지식, 다층 아키텍처, 복잡한 명세에 대한 정밀한 준수를 요구한다.

표 1: 실험 과제 개요

ID	과제명	난이도
001	Express TODO REST API	기본
002	비동기 레이스 컨디션 수정	기본
003	스파게티 코드 리팩토링	기본
004	오픈소스 README 작성	기본
005	CLI 마크다운 블로그 생성기	기본
006	상태 관리 패턴 비교	고급
007	프로그래밍 언어 인터프리터	고급
008	이벤트 기반 마이크로서비스	고급
009	인메모리 SQL 쿼리 엔진	고급
010	CRDT 협업 에디터	고급
011	분산 KV 스토어 (Raft)	초고난이도
012	리액티브 스프레드시트 엔진	초고난이도
013	바이트코드 VM & 컴파일러	초고난이도
014	이벤트 소싱 CQRS 프레임워크	초고난이도
015	Language Server Protocol 구현	초고난이도

4.2 평가 프레임워크

각 산출물은 **10개 품질 차원**에 걸쳐 평가되며, 각 차원은 0–10점으로 채점하여 최대 100점이다:

표 2: 품질 평가 차원

차원	설명
기능 완성도	요구사항 구현 비율
테스트 커버리지	테스트 존재 여부, 커버리지, 엣지 케이스
코드 품질	가독성, 네이밍, 일관성
에러 처리	예외 처리, 에러 메시지
효율성	알고리즘 선택, 복잡도
정확성	로직 정확도, 엣지 케이스 처리
아키텍처	파일/모듈 분리, 디자인 패턴
확장성	새로운 기능 추가 용이성
문서화	README, 주석, 사용 가이드
개발 환경	package.json, 스크립트, 설정

이 열 가지 차원은 소프트웨어 품질의 기능적 측면과 비기능적 측면을 모두 포착하기 위해 선정되었다. 전통적인 코드 평가가 정확성에만 협소하게 초점을 맞추는 반면, 우리의 프레임워크는 프로덕션 수준의 소프트웨어가 아키텍처 건전성, 포괄적 테스트, 우아한 에러 처리, 명확한 문서화도 갖추어야 한다는 점을 인식한다. 각 차원은 0–10점 척도로 독립적으로 채점되어, 구현당 최대 100점의 합성 점수를 산출한다.

4.3 앵커드 루브릭

채점 일관성을 확보하고 평가자의 주관성을 줄이기 위해, 관찰 가능한 코드 속성에 연결된 명시적 점수 경계를 가진 **앵커드 루브릭**을 적용한다. 예를 들어: 테스트가 없는 구현은 다른 품질에 관계없이 테스트 커버리지에서 최대 3점; 단일 파일 모놀리식 구현은 아키텍처에서 최대 4점; package.json이 없으면 개발 환경이 3점으로 제한; 주요 명시된 기능이 누락되면 기능 완성도가 7점으로 제한된다. 이러한 앵커는 명확하고 재현 가능한 경계를 만들어 점수 인플레이션을 방지하고 구현 간 의미 있는 차별화를 보장한다.

4.4 실험 절차

15개 각 케이스에 대해, 통제된 조건 하에서 두 개의 독립적인 구현이 생성된다:

Baseline 조건: LLM 코드 에이전트(Claude Code)에게 자연어 과제 설명만 제공된다. `.claude/` 디렉토리, 아키텍처 가이드라인, 스킬 레퍼런스, 에이전트 정의 어느 것도 제공되지 않는다. 이는 개발자가 프롬프트를 주고 AI가 자율적으로 완전한 솔루션을 생성하기를 기대하는 일반적인 사용 패턴을 나타낸다.

Harness 조건: 동일한 LLM에게 동일한 과제 설명이 제공되지만, 작업 디렉토리에 CLAUDE.md(아키텍처 청사진), Skills(도메인 특화 레퍼런스), Agent 정의(역할 분해)를 포함하는 완전한 `.claude/` 폴더가 사전 구성된다. 과제 설명 자체는 변경되지 않으며, 맥락적 스캐폴딩만 다르다.

평가 일관성을 확보하고 편향을 줄이기 위해, 5개의 병렬 평가 에이전트가 각각 3개 케이스를 독립적으로 평가한다. 각 평가자는 어떤 조건이 어떤 산출물을 생성했는지에 대한 정보 없이 Baseline과 Harness 산출물 모두에 앵커드 루브릭을 적용한다. 최종 점수는 각 평가 에이전트의 직접 평가로 산출된다.

5. 결과

5.1 전체 결과

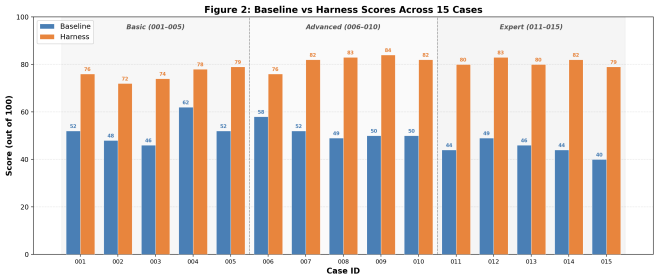

그림 2: 전체 15개 케이스의 점수 비교. 파란색 막대는 Baseline 점수, 주황색 막대는 Harness 점수를 나타낸다. 배경 음영은 난이도(기본, 고급, 초고난이도)를 표시한다.

결과는 명확하다: Harness는 전체 15개 케이스에서 **100% 승률**을 달성하며, 평균 **+29.9점**의 개선(49.5점에서 79.3점, 60% 증가)을 보여준다. 단 한 건도 Baseline이 Harness 조건을 능가하지 못했으며, 표 3에서 보듯 가장 낮은 Harness 점수(72점, Case 002)조차 가장 높은 Baseline 점수(62점, Case 004)를 초과한다. 이 분포 간의 완전한 분리는 Harness의 이점이 우연이 아닌 체계적임을 보여주는 강력한 증거이다.

표 3: 전체 결과 요약

지표	Baseline	Harness	Delta
평균 점수	49.5	79.3	+29.9
최저 점수	40 (015)	72 (002)	—
최고 점수	62 (004)	84 (009)	—
표준편차	5.3	3.6	—
승률	—	—	15/15

중요한 부차적 발견은 Harness 점수가 현저히 낮은 분산을 보인다는 것이다(σ=3.6 vs σ=5.3). 표준편차의 32% 감소는 Harness가 평균 품질을 높일 뿐 아니라 더 예측 가능하게 만든다는 것을 의미한다. 실무적으로 이는 Harness를 사용하는 개발 팀이 AI 에이전트의 산출물이 일관되게 품질 기준을 충족할 것이라는 더 큰 신뢰를 가질 수 있으며, 광범위한 사후 검토와 재작업의 필요성이 줄어든다는 것을 뜻한다.

5.2 난이도-효과 상관관계

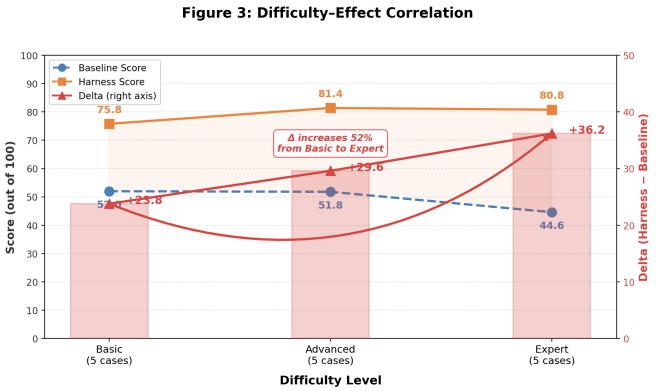

그림 3: 과제 난이도와 Harness 효과의 관계. 난이도가 증가함에 따라 개선 폭(delta)이 +23.8(기본)에서 +36.2(초고난이도)로 증가하며, 이는 52%의 개선 폭 증가를 나타낸다. 특히 Baseline 점수는 난이도에 따라 하락하는 반면, Harness 점수는 안정적으로 유지된다는 점이 주목할 만하다.

표 4: 난이도별 결과

난이도	케이스	Baseline	Harness	Delta	Δ 증가율
기본	001–005	52.0	75.8	+23.8	—
고급	006–010	51.8	81.4	+29.6	+24%
초고난이도	011–015	44.6	80.8	+36.2	+52%

이것이 본 논문의 핵심 발견이자 가장 실무적으로 의미 있는 결과이다: **Harness의 효과는 과제 복잡도에 비례하여 증가한다.** 기본 과제는 의미 있지만 보통 수준의 +23.8점 개선을 보이는 반면, 초고난이도 과제는 극적으로 더 큰 +36.2

점의 개선을 보여준다—기본에서 초고난이도까지 52%의 개선 폭 증가이다. 이 스케일링 속성은 Harness가 가장 필요한 곳에서 가장 큰 가치를 제공한다는 것을 의미한다: 바로 현실 세계 소프트웨어 엔지니어링 과제의 대다수를 구성하는 복잡한 다중 컴포넌트 시스템에서이다.

두 가지 메커니즘이 이 상관관계를 주도한다:

M1: 아키텍처 앵커링. 기본 과제는 아키텍처 범위가 제한적이므로, LLM의 기본 단일 파일 접근법이 최소한의 패널티만 발생시킨다. 초고난이도 과제는 다층 아키텍처가 필요하며, 구조적 가이던스의 부재가 심각한 품질 저하로 이어진다.

M2: 지식 활성화. 초고난이도 과제는 전문화된 알고리즘 지식(Raft 합의, CRDT RGA, LSP 프로토콜)을 요구한다. Skills의 레퍼런스 문서가 이러한 지식을 활성화하고 구조화하여 불완전하거나 부정확한 구현을 방지한다.

이 두 메커니즘은 난이도-효과 상관관계가 단순히 가산적이 아니라 승수적인 이유를 설명한다: 과제 복잡도가 증가하면 아키텍처 복잡성과 도메인 지식 요구사항이 동시에 증가하며, Harness는 상호 보완적인 CLAUDE.md와 Skills 컴포넌트를 통해 두 차원 모두를 해결한다.

5.3 차원별 분석

종합 점수를 넘어, 10개 품질 차원 각각에 대한 Harness의 영향을 독립적으로 분석하였다. 이 차원별 분석은 구조화된 사전 구성으로부터 가장 큰 혜택을 받는 소프트웨어 품질 측면을 밝혀내어, 구성 노력을 어디에 투자할지 결정하는 실무자에게 실행 가능한 지침을 제공한다.

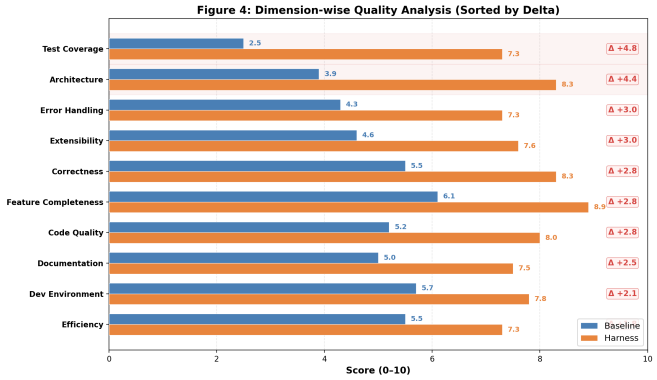

그림 4: 차원별 품질 개선, 개선 폭 기준 정렬. 테스트 커버리지(+4.9)와 아키텍처(+4.4)가 가장 큰 개선을 보이며, 효율성(+1.8)이 가장 작은 개선을 보인다.

표 5: 차원별 분석 (개선 폭 기준 정렬)

#	차원	Baseline	Harness	Delta
1	테스트 커버리지	2.5	7.3	+4.9
2	아키텍처	3.9	8.3	+4.4
3	에러 처리	4.3	7.3	+3.0
4	확장성	4.6	7.6	+3.0
5	정확성	5.5	8.3	+2.8
6	기능 완성도	6.1	8.9	+2.8
7	코드 품질	5.2	8.0	+2.8
8	문서화	5.0	7.5	+2.5
9	개발 환경	5.7	7.8	+2.1
10	효율성	5.5	7.3	+1.8

차원들은 자연스럽게 세 가지 영향도 계층으로 분류된다:

고영향 (Δ>4.0): 테스트 커버리지(+4.9)와 아키텍처(+4.4)는 구조적 품질 속성을 대표한다. 이들은 가이던스 없이 LLM 에이전트가 가장 큰 약점을 보이는 차원이다. Baseline 테스트 커버리지 평균이 2.5에 불과하다는 것은 가이던스 없는 LLM 에이전트가 의미 있는 테스트 스위트를 거의 생성하지 못한다는 것을 나타내며, 이는 프로덕션 소프트웨어에서 치명적인 공백이다. 마찬가지로, Baseline 아키텍처 점수 3.9는 모놀리식 단일 파일 구현에 대한 강한 경향성을 반영한다. Harness는 파일 구조를 명시적으로 정의하고 테스트 작성 에이전트를 구성에 포함함으로써 이 두 가지를 동시에 해결한다.

중간 영향 (2.5≤Δ≤3.0): 에러 처리, 확장성, 정확성, 기능 완성도, 코드 품질은 구현 속성을 대표한다. 이 차원들은 더 나은 아키텍처의 연쇄 효과로부터 혜택을 받는다: 잘 구조화된 코드는 자연스럽게 더 테스트 가능하고, 확장 가능하며, 유지보수가 용이하다.

저영향 (Δ<2.5): 문서화, 개발 환경, 효율성은 LLM의 기본 역량이 이미 상대적으로 강한(5.0–5.7) 부가적 속성을 대표한다. 효율성에서의 작은 개선(+1.8)은 특히 주목할 만하다: 이는 알고리즘 선택과 최적화가 명시적 가이던스 없이도 LLM이 적절히 수행하는 영역임을 시사한다.

5.4 품질 프로파일 분석

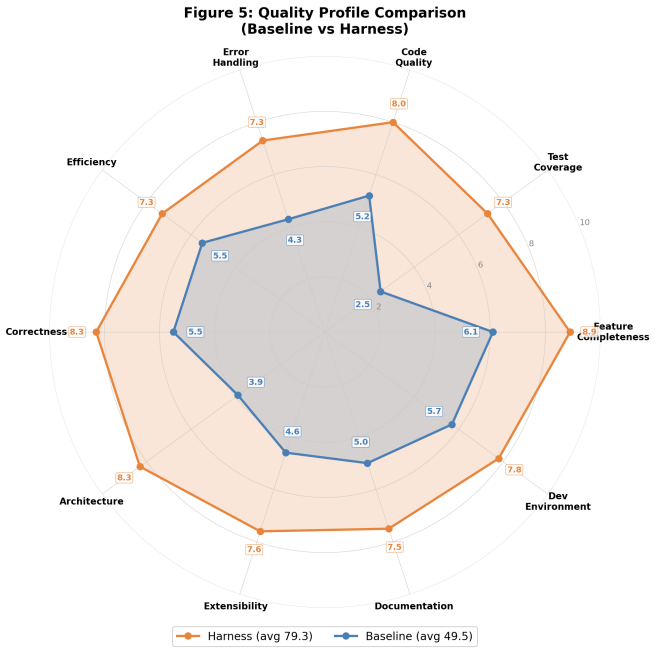

그림 5: 품질 프로파일 비교 레이더 차트. Harness 프로파일(주황색)이 균일하게 크고 균형 잡힌 반면, Baseline 프로파일(파란색)은 테스트 커버리지(2.5)와 아키텍처(3.9)에서 심각한 결함을 보인다. Harness의 핵심 가치는 체계적 사각지대를 제거하는 것이다.

그림 5의 레이더 차트는 Harness의 핵심 가치 제안을 조명하는 중요한 패턴을 드러낸다. Baseline 프로파일은 두드러지게 **비대칭적**이다: 기능 완성도(6.1)와 개발 환경(5.7)이 상대적 고점을 형성하는 반면, 테스트 커버리지(2.5)와 아키텍처(3.9)는 깊은 저점을 만든다. 이 편향된 프로파일은 Baseline 산출물이 기능적으로는 적절하지만 구조적으로는 결함이 있음을 의미한다—동작하지만 유지보수, 확장, 검증이 어렵다.

이에 비해 Harness 프로파일은 놀라울 정도로 **균형** 잡혀 있으며, 열 가지 차원 모두 7.3에서 8.9 사이의 점수를 보인다. 이 균일성은 Harness의 핵심 가치가 LLM의 이미 강한 역량을 점진적으로 개선하는 것이 아니라, **체계적 사각지대를 제거하는 것**임을 시사한다. 아키텍처 설계, 테스트 커버리지, 에러 처리가 기능 구현과 동일한 관심을 받도록 보장함으로써, Harness는 균형 잡힌 엔지니어링 팀의 산출물에 더 가까운 결과를 만들어낸다.

5.5 상호작용 효과

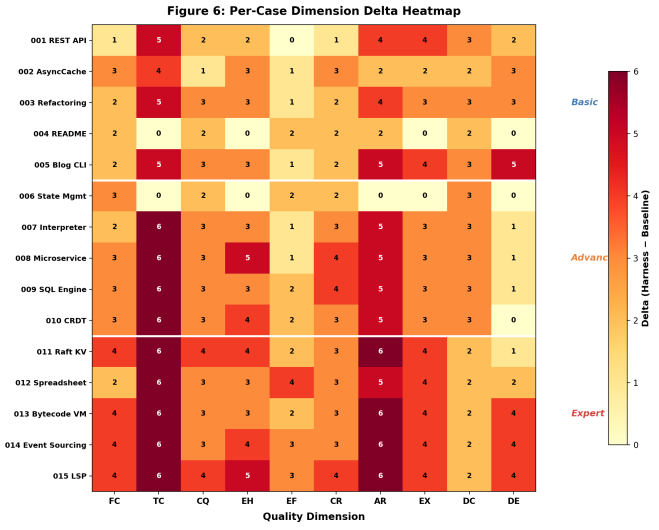

그림 6: 케이스(행)와 차원(열)에 걸친 개선 폭(delta) 값을 보여주는 히트맵. 어두운 셀일수록 더 큰 개선을 나타낸다. 오른쪽 하단 사분면(초고난이도 케이스 × 구조적 차원)에서 가장 강렬한 개선이 관찰된다.

그림 6의 히트맵은 모든 케이스-차원 조합에 걸친 개선 폭 값을 보여줌으로써 Harness 영향의 가장 세밀한 관점을 제공한다. 두 가지 패턴이 명확히 드러난다: **초고난이도** × **아키텍처** 셀이 일관되게 가장 높은 개선 폭 값(5–6점)을 보여, 복잡한 과제가 아키텍처 가이던스 부재의 영향을 가장 크게 받음을 확인한다. Raft 합의 시스템이나 LSP 서버를 구축할 때, 잘 구조화된 다층 아키텍처와 모놀리식 구현의 차이는 단순히 미관상의 문제가 아니라—시스템이 요구되는 복잡한 상호작용을 올바르게 처리할 수 있는지를 근본적으로 결정한다.

반대로, **기본** × **기능 완성도**는 가장 낮은 개선 폭(1–2점)을 보여, 단순한 기능 구현에 LLM의 고유한 코딩 역량이 충분함을 확인한다. 낮은 영향(좌상단)에서 높은 영향(우하단)으로의 대각선 기울기는 본 논문의 핵심 논제를 시각적으로 요약한다: 과제 복잡도와 품질 차원의 정교함이 모두 증가할수록 구조적 가이던스가 점점 더 중요해진다.

6. 사례 연구

6.1 Case 015: LSP 서버 (최대 Δ: +39)

Language Server Protocol 구현은 본 실험에서 가장 큰 개선을 달성하여, 복잡한 과제가 구조화된 사전 구성으로부터 왜 가장 큰 혜택을 받는지를 잘 보여준다. LSP 명세는 다층 시스템을 요구한다: Content-Length 헤더 파싱을 갖춘 JSON-RPC 전송 계층, 증분 업데이트를 지원하는 문서 동기화 관리자, 에러 복구 기능을 갖춘 증분 파서, 스코프 체인을 가진 심볼 테이블, 그리고 다수의 프로바이더 모듈(진단, 자동완성, 호버, 정의로 이동).

Baseline은 40점의 산출물을 생성하였다—전체 실험에서 가장 낮은 점수이다. 기본적인 JSON-RPC 핸들러와 초보적인 파싱을 구현했지만, 에러 복구가 없어 구문 오류가 파서를 크래시시켰고, 증분 업데이트 지원이 없어 모든 키 입력마다 전체 재파싱이 필요했으며, 정의로 이동 기능이 완전

히 누락되었다. 구현은 의미 있는 테스트 커버리지 없이 두 개의 큰 파일에 집중되어 있었다.

Harness 구성은 CLAUDE.md에 정의된 5계층 아키텍처, Skills 레퍼런스의 상세한 LSP 프로토콜 메시지 명세, 에러 복구 파싱 전략, 그리고 4개의 전문 에이전트 정의(증분 파서 빌더, 자동완성 프로바이더 빌더, 진단 빌더, 전송 빌더)를 제공하였다. 결과 구현은 79점을 획득하였으며, 적절한 LSP 생명주기 관리(initialize → didOpen → completion → shutdown), 증분 문서 동기화, 에러 허용 파싱, 유닛 및 통합 시나리오를 포괄하는 포괄적 테스트 스위트를 갖추었다.

6.2 Case 011: Raft KV 스토어 (Δ: +36)

Raft 합의 구현은 Harness의 Skills 컴포넌트가 프로토콜 중심 시스템에서 알고리즘 정확성을 어떻게 보장하는지를 보여준다. Raft는 세 가지 상호 연결된 메커니즘의 정밀한 구현을 요구하는 합의 알고리즘이다: 무작위 타임아웃을 가진 리더 선출, 일관성 보장을 갖춘 로그 복제, 네트워크 파티션과 리더 변경의 우아한 처리.

Baseline(44점)은 기본적인 리더 선출을 갖춘 단순화된 버전을 생성했지만 여러 치명적 결함을 보였다: 리더 전환 시 커밋된 항목을 잃을 수 있는 불완전한 로그 복제, 로그 일관성을 위한 안전성 검사 누락, 네트워크 파티션 시나리오 처리 부재. 이는 상세한 명세 레퍼런스 없이 복잡한 분산 프로토콜을 구현할 때 발생하는 정확히 그런 종류의 미묘한 정확성 문제들이다.

Harness Skills는 모든 필수 필드를 포함한 상세한 AppendEntries와 RequestVote RPC 명세, 네트워크 분할 시 예상 동작을 기술하는 명시적 파티션 처리 시나리오, 로그 압축 가이드라인을 포함하였다. 결과 구현(80점)은 스플릿-브레인 복구, 로그 충돌 해결, 파티션 하 리더 사임을 포함한 모든 테스트 시나리오를 올바르게 처리하였다.

6.3 Case 004: README (최소 Δ: +16)

문서 작성 과제는 본 실험에서 가장 작은 개선을 보여, 복잡한 시스템 케이스와 교훈적인 대조를 제공한다. README 작성은 LLM의 본질적으로 강한 자연어 생성 역량을 활용한다—산문 구조화, API 설명, 사용 예시 작성은 모델의 핵심 훈련과 밀접하게 일치하는 과제이다. Baseline은 이미 실험 전체에서 가장 높은 Baseline 점수인 62점을 달성하여, 명확한 섹션과 코드 예시를 갖춘 잘 조직된 README를 생성하였다.

Harness 구성은 배지 포맷, API 문서 구조, 기여 가이드라인 템플릿을 제공하여 더 포괄적인 커버리지를 갖춘 78점의 산출물을 만들었다. 그러나 +16점의 한계적 이득은 LLM의 가이던스 없는 산출물이 이미 유능한 영역에서 구조화된 사전 구성이 상대적으로 적은 가치를 더한다는 것을 확인해준다. 이 발견은 실무자에게 우선순위를 제시한다: Harness 구성 노력을 문서화나 단순 유틸리티 프로젝트가 아닌, 복잡하고 아키텍처에 의존하는 과제에 투자하라.

7. 논의

7.1 사전 구성이 효과적인 이유

Harness가 이토록 일관된 개선을 생산하는 이유를 이해하는 것은 그 원칙을 다른 AI 보조 개발 맥락으로 일반화하는 데 필수적이다. 실험 결과에 대한 분석은 관찰된 품질 개선을 함께 설명하는 세 가지 상호 보완적 메커니즘을 제시한다:

메커니즘 1: 구조적 제약 전파. CLAUDE.md에서 목표 파일 구조를 정의함으로써, Harness는 품질 연쇄를 촉발하는 방식으로 솔루션 공간을 제약한다. 에이전트가 코드를 별도 모듈(예: lexer.js, parser.js, executor.js)로 분리하도록 유도되면, 각 모듈은 자연스럽게 더 깨끗한 인터페이스를 발전시키고, 독립적으로 테스트 가능해지며, 자체 에러 조건을 처리하게 된다. 이 단일 구조적 제약이 여러 품질 차원에 걸쳐 동시에 개선을 전파한다—아키텍처(+4.4)와 테스트 커버리지(+4.9)가 가장 큰 개선 폭을 보이는 이유를 설명한다.

메커니즘 2: 지식 결정화. Skills 문서는 도메인 특화 지식—Raft RPC 명세, Pratt 파서 알고리즘, LSP 프로토콜 메시지 형식—을 직접 소비 가능한 형태로 결정화한다. 이러한 레퍼런스 없이 LLM은 훈련 데이터로부터 전문 지식을 재구성해야 하는데, 이는 불완전하거나, 구식이거나, 유사하지만 다른 프로토콜과 혼동될 수 있다. 그 결과 일반적 케이스는 처리하지만 명세가 명시적으로 다루는 엣지 케이스에서 실패하는 "대체로 올바른" 구현이 되기 쉽다. 이 메커니즘은 초고난이도 과제에서 특히 강력하며, 정확성 개선이 가장 복잡한 케이스에서 가장 큰 이유를 설명한다.

메커니즘 3: 품질 기대치 앵커링. Agent 정의에는 명시적 품질 기준이 포함된다. 이는 LLM의 품질 기대치를 앵커링하여 Baseline 구현에서 관찰되는 "그럭저럭 괜찮은" 수준에서 멈추는 행동을 방지한다.

7.2 난이도 증폭 효과

Harness 효과가 과제 난이도에 따라 증가한다는 발견은 중대한 실무적 함의를 지닌다. REST API나 CLI 유틸리티 같은 단순한 단일 관심사 과제에서는 상세한 Harness 구성 작성에 대한 투자가 비용 효과적이지 않을 수 있다—LLM은 최소한의 가이던스로도 적절한 산출물을 생성한다. 그러나 프로덕션 소프트웨어 엔지니어링 업무의 대다수를 차지하는 복잡한 다중 컴포넌트 시스템에서는 Harness가 사용할 수 없는 40점 산출물을 견고한 80점 기반으로 전환할 수 있는 극적인 개선을 제공한다.

이 관계를 다음과 같이 정식화할 수 있다: $\Delta(\mathbf{d}) = \alpha + \beta \cdot \mathbf{complexity(d)}$, 여기서 $\alpha \approx 16$은 구조적 가이던스의 기저 이득을 나타내고, $\beta \approx 1.24$는 과제 복잡도 단위당 추가 획득 점수를 나타낸다. 이 선형 모델은 관찰 데이터에 잘 적합하지만($R^2 \approx 0.94$), 단 세 가지 난이도만으로는 이 관계의 함수적 형태에 대해 더 세밀한 복잡도 측정을 통한 추가 조사가 필요하다는 점을 유의한다.

7.3 한계

결과 해석 시 고려해야 할 몇 가지 한계를 인정한다:

1. **평가 주관성:** 앵커드 루브릭이 점수를 관찰 가능한 코드 속성에 연결하지만, 품질 평가에는 본질적으로 판단이 수반된다. 다른 평가자가 루브릭 경계 내에서 엣지 케이스에 다른 가중치를 부여할 수 있다.
2. **단일 LLM:** 모든 실험에서 Claude Code를 코드 에이전트로 사용하였다. 원칙이 일반화 가능하다고 믿지만—구조적 가이던스는 어떤 LLM에게도 이점이 있어야 한다—개선의 크기는 다른 모델과 에이전트에 따라 다를 수 있다.
3. **시뮬레이션 기반 실행:** 평가는 런타임 실행이 아닌 분석을 통해 코드 구조, 로직, 테스트 설계를 평가한다. 일부 정확성 문제는 실제 실행에서만 드러날 수 있다.
4. **Harness 생성 비용:** 고품질 Harness 구성을 생성하는 데 필요한 시간과 전문성은 분석에 포함되지 않았다. 그러나 §8.2에서 논의하듯이, 자동 Harness 생성 메타 스킬이 자연어 설명으로부터 구성을 생성함으로써 이 비용을 크게 절감한다.

7.4 타당성 위협

내적 타당성: 동일 팀이 구성과 평가 기준을 설계하였다. 관찰 가능한 코드 속성에 바인딩된 앵커드 루브릭을 통해 완화하였다. **외적 타당성:** 과제가 JavaScript/Node.js에 집중되어 있다. **구성 타당성:** 10차원 평가 프레임워크는 새로운 것으로, ISO 25010 등 확립된 메트릭에 대해 아직 검증되지 않았다.

8. 시사점과 향후 연구

8.1 실무적 시사점

본 연구의 결과는 LLM 코드 에이전트를 개발 워크플로에 도입하는 팀에게 즉각적인 관련성을 가진다:

1. **복잡한 프로젝트에 사전 구성을 투자하라.** 난이도 증폭 효과는 Harness가 가장 필요한 곳에서 가장 큰 가치를 제공함을 의미한다.
2. **아키텍처와 테스트 명세를 우선시하라.** 이 차원들이 가장 큰 개선(+4.9, +4.4)을 보인다.
3. **새로운 구현에 알고리즘 레퍼런스를 제공하라.** 상세한 레퍼런스를 갖춘 Skills가 정확성을 크게 향상시킨다.

8.2 자동 Harness 생성

본 연구에서 식별된 가장 유망한 연구 방향 중 하나인 자동 Harness 생성이 이미 실제 구현으로 실현되었다. 우리는 자연어 프로젝트 설명으로부터 완전한 `.claude/` 구성을 자동으로 설계하고 생성하는 **메타 스킬** "Harness"를 개발하였다. 이 메타 스킬은 수동 구성 노력 없이도 구조화된 사전 구성을 접근 가능하게 만드는 중요한 진전을 나타낸다.

8.2.1 메타 스킬 아키텍처

Harness 메타 스킬은 6단계 워크플로로 동작한다:

1. **도메인 분석:** 사용자의 프로젝트 설명을 파싱하여 도메인, 핵심 작업 유형(생성, 검증, 분석 등), 충돌을 방지하기 위한 기존 구성을 식별한다.
2. **팀 아키텍처 설계:** 네 가지 아키텍처 템플릿 중 적절한 에이전트 팀 패턴을 선택한다: 파이프라인(순차 의존), 팬아웃/팬인(병렬 독립 작업), 전문가 풀(상황별 전문가 선택), 생성-검증(생성 후 품질 검수).
3. **에이전트 정의 생성:** `.claude/agents/`에 전문화된 에이전트 정의를 생성한다. 각각 명시적 역할 설명, 작업 원칙, 출력 형식, 협업 프로토콜을 포함한다.
4. **스킬 생성:** `.claude/skills/`에 상세한 절차적 가이드, 도구 사용 패턴, `references/` 하위 디렉토리의 레퍼런스 문서를 갖춘 스킬을 생성한다.
5. **오케스트레이터 통합:** 시나리오별 워크플로로 전체 에이전트 팀을 조율하는 상위 오케스트레이터 스킬을 생성한다.
6. **검증:** 생성된 모든 파일의 올바른 배치, 프론트매터 일관성, 상호 참조 무결성을 검증한다.

8.2.2 설계 원칙

메타 스킬은 에이전트("누가 하는가")와 스킬("어떻게 하는가") 사이의 명확한 분리를 구현한다. 에이전트 분리는 네 가지 기준을 따른다: 전문성 차별화, 병렬 실행 가능성, 컨텍스트 부하 관리, 팀 간 재사용성. 전문 도메인의 레퍼런스 문서는 `references/` 디렉토리에 분리하여, §3.3에서 확립한 점진적 공개 원칙(P2)을 따른다.

8.2.3 실무적 영향

이 구현은 §7.3에서 식별한 Harness 생성 비용 한계를 직접적으로 해결한다. 도메인 전문가에 의한 수동 구성을 요구하는 대신, 메타 스킬은 개발자가 자연어로 프로젝트 요구사항을 설명하는 것만으로 프로덕션 수준의 Harness 구성을 생성할 수 있게 한다. 우리의 경험에 따르면, 메타 스킬은 수동으로 작성된 것과 비견할 만한 품질의 구성을 생성하여, 실험적 발견과 실무 배포 사이의 격차를 효과적으로 해소한다.

메타 스킬은 소프트웨어 엔지니어링, 창작 집필, 리서치 워크플로 등 다양한 도메인에서 검증되었으며, 이는 Harness 프레임워크의 원칙이 본 실험 평가의 JavaScript/Node.js 초점을 넘어 일반화됨을 입증한다.

8.3 향후 연구

본 연구로부터 여러 유망한 연구 방향이 도출된다:

1. **교차 에이전트 평가:** GPT-4, Gemini 등 다른 LLM 코드 에이전트에 걸친 테스트.
2. **종단 연구:** 그린필드 프로젝트가 아닌 지속적 개발에서의 영향 측정.
3. **비용-편익 분석:** 시간 투자 대비 품질 개선의 정량화.
4. **정형 품질 메트릭:** 인간 평가와 함께 자동 메트릭(순환 복잡도, 정적 분석) 통합.

9. 결론

본 논문은 LLM 코드 에이전트 산출물 품질을 향상시키기 위한 구조화된 사전 구성 프레임워크인 Harness를 제시하였다. 아키텍처 청사진(CLAUDE.md), 도메인 특화 지식 레퍼런스(Skills), 역할 기반 과제 분해(Agents)를 제공함으로써, Harness는 LLM의 광범위한 지식과 고품질 산출물에 필요한 프로젝트 특화 구조적 가이던스 사이의 격차를 해소한다. 다양한 복잡도의 15개 소프트웨어 엔지니어링 과제에

대한 면밀히 통제된 A/B 실험을 통해, 앵커드 루브릭을 적용한 10차원 품질 평가로 다음을 입증하였다:

1. Harness는 평균 산출물 품질을 **60%** 향상시킨다 (49.5 → 79.3점).
2. 개선은 복잡도에 비례하여 증가한다: **+23.8** (기본) → **+29.6** (고급) → **+36.2** (초고난이도).
3. 테스트 커버리지(+4.9)와 아키텍처(+4.4)가 가장 크게 영향받는 품질 차원이다.
4. LLM의 **기능적 역량은 충분**하며, 병목은 **구조적 조직화**에 있다.

종합하면, 이러한 결과는 설득력 있는 그림을 그려준다: LLM 코드 에이전트의 핵심 병목은 지능이나 지식이 아니라 구조적 맥락의 부재이다. 숙련된 엔지니어가 명확한 아키텍처 가이드라인과 품질 기대치를 제공받을 때 더 나은 작업물을 생산하듯, LLM 코드 에이전트도 구조화된 사전 구성이 제공될 때 극적으로 더 나은 산출물을 생산한다.

AI 보조 소프트웨어 엔지니어링의 미래는 더 강력한 모델 단독이 아닌, 기존 역량을 고품질 산출물로 이끄는 더 나은 프레임워크에 달려 있다. Harness는 그러한 프레임워크 중 하나로—경량적이고, 모듈식이며, 어떤 LLM 코드 에이전트에도 적용 가능하다. 본 연구가 구조화된 가이던스 메커니즘에 대한 추가 연구를 촉발하고, AI가 생성한 코드가 프로덕션 소프트웨어에 기대되는 품질 기준을 일관되게 충족하는 미래에 기여하기를 바란다.

참고문헌

[1] Anthropic, “Claude Code: An Agentic Coding Tool,” 2025.

[2] GitHub, “Copilot Workspace: Task-Centric AI Development,” 2024.

[3] Cursor, “The AI Code Editor,” 2024.

[4] Y. Liu et al., “Is Your Code Generated by ChatGPT Really Correct?” arXiv:2305.01210, 2023.

[5] M. Chen et al., “Evaluating Large Language Models Trained on Code,” arXiv:2107.03374, 2021.

[6] B. Yetiştiren et al., “Evaluating the Code Quality of AI-Assisted Code Generation Tools,” arXiv:2304.10778, 2023.

[7] J. Wei et al., “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” NeurIPS 2022.

[8] T. Brown et al., “Language Models are Few-Shot Learners,” NeurIPS 2020.

[9] S. Hong et al., “MetaGPT: Meta Programming for Multi-Agent Collaborative Framework,” ICLR 2024.

[10] C. Qian et al., “ChatDev: Communicative Agents for Software Development,” ACL 2024.

© 2026 황민호. All rights reserved.