creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

석 사 학 위 논 문

소프트웨어 개발 단계별 인공지능 기술 활용 방안 연구

HANSUNG UNIVERSITY

2026년

한성대학교 지식서비스&컨설팅대학원

스마트융합컨설팅학과

스마트융합컨설팅전공

양 병 헌

HANSUNG
UNIVERSITY

석 사 학 위 논 문
지도교수 최동준

소프트웨어 개발 단계별 인공지능 기술 활용 방안 연구

AI Utilization Strategies across the Phases of the Software Development Life Cycle (SDLC)

HANSUNG UNIVERSITY

2026년 6월 일

한성대학교 지식서비스&컨설팅대학원

스마트융합컨설팅학과

스마트융합컨설팅전공

양 병 헌

석 사 학 위 논 문
지도교수 최동준

소프트웨어 개발 단계별 인공지능 기술 활용 방안 연구

AI Utilization Strategies across the Phases of the Software Development Life Cycle (SDLC)

위 논문을 컨설팅학 석사학위 논문으로 제출함

2026년 6월 일

한성대학교 지식서비스&컨설팅대학원

스마트융합컨설팅학과

스마트융합컨설팅전공

양 병 헌

양병헌의 컨설팅학 석사학위 논문을 인준함

2026년 6월 일

심사위원장 주 형 근 (인)

심 사 위 원 정 진 택 (인)

심 사 위 원 최 동 준 (인)

HANSUNG UNIVERSITY

국 문 초 록

소프트웨어 개발 단계별 인공지능 기술 활용 방안 연구

한성대학교 지식서비스&컨설팅대학원
스 마 트 융 합 컨 설 팅 학 과
스 마 트 융 합 컨 설 팅 전 공
양 병 헌

2022년 11월 OpenAI가 대규모 언어 모델(Large Language Model, LLM)을 기반으로 한 ChatGPT를 내놓은 뒤, 생성형 인공지능(Generative AI, 이하 생성형 AI)은 빠르게 업무 현장에 활용되고 있다. 초기에는 학습 데이터의 시점에 대한 한계와 환각(hallucination) 현상 탓에 실무에서 활용되기까지 오랜 시간이 걸릴 것으로 예상되었으나, 새로운 모델이 잇따라 나오면서 사용자층과 활용 범위가 함께 넓어졌다. 소프트웨어 개발 영역도 예외가 아니어서, 소스코드 작성과 문서 작성에 생성형 AI 기술을 쓰는 개발자를 어렵지 않게 볼 수 있다. 그러나 개발 현장에서 이 기술이 단계별로 어떻게 쓰이고 생산성에 어떤 영향을 주는지에 관한 실태 연구는 아직 충분하지 않다.

이 연구는 요구사항 분석에서 배포·유지보수에 이르는 소프트웨어 개발 생명주기(SDLC) 전 단계를 대상으로 생성형 AI 기술의 활용 실태와 생산성 효과를 분석하였다. 먼저 문헌조사로 단계별 활용 수준과 생산성의 관계를 정리하였다. 이어 실제 프로젝트에서 생성형 AI 기술을 사용하는 현업 개발자 200명을 대상으로 설문조사를 실시하여 단계별 활용 수준과 생산성 향상도를 측정·비교하였다. 끝으로 프로젝트 관리자(PM)급 이상의 현장 전문가 3인에게 설문 결과를 검토받아, 정량 분석과 현장 해석을 교차 검증(Triangulation)하였다.

분석 결과, SDLC 모든 단계에서 도구의 활용 수준보다 개발자가 체감하는 생산성 향상도가 높게 나타났다. 다만 그 차이는 단계마다 달랐다. 요구사항 분석

단계에서는 활용 수준이 상대적으로 낮은데도 두 지표의 격차가 0.37점으로 가장 커, 활용만으로도 효과를 더 크게 체감할 수 있었다. 반면 구현과 테스트 단계는 격차가 각각 0.13점, 0.09점으로 가장 좁았는데, 이유는 생성된 코드를 검증하고 수정하는 비용이 효과를 상쇄하기 때문으로 예상된다. 도입을 가로막는 요인으로는 환각 현상에 따른 기술적 불확실성(33.0%)과 투자 대비 효과의 불투명성(27.0%)이 높게 지목되었고, 직무 측면에서는 전문성의 저하와 도구 의존에 대한 우려가 컸다.

이 연구는 SDLC의 단계별로 나누어 생성형 AI 기술의 활용 수준과 생산성 효과를 비교했다는 데 의의가 있다. 이 연구의 결과가 프로젝트를 발주·수행·감리하는 실무 담당자들이 생성형 AI 기반으로 프로젝트를 진행하는 데 참고 자료로 쓰이기를 기대한다.

【주요어】 소프트웨어 개발 생명주기(SDLC), 생성형 AI, 개발 생산성, AI 도입 장애 요인, LLM 코딩 도구

HANSUNG UNIVERSITY

목 차

Ⅰ. 서론 ····· 1
1.1 연구의 필요성 ····· 1
1.2 연구의 목적 ····· 2
1.3 연구 방법 및 논문의 구성 ····· 3
1.3.1 연구 방법 ····· 3
1.3.2 논문의 구성 체계 ····· 4

Ⅱ. 이론적 배경 ····· 5
2.1 소프트웨어 개발 생명주기(SDLC) ····· 5
2.2 소프트웨어 개발을 지원하는 AI 도구 ····· 7
2.3 소프트웨어 개발 생산성 향상과 AI 도구 활용 수준 ····· 12
2.4 선행 연구 검토 요약 ····· 13

Ⅲ. 연구 방법 및 절차 ····· 14
3.1 연구 방법 ····· 14
3.2 연구 절차 ····· 15
3.2.1 SDLC 단계별 AI 도구 활용 연구 ····· 15
3.2.2 SDLC 단계 특성 분석 ····· 15
3.2.3 전문가 인터뷰 기반 설문 결과 리뷰 ····· 22

Ⅳ. SDLC 단계별 AI 활용 연구 ····· 23
4.1 SDLC 단계별 AI 도구 활용 생산성 연구 ····· 23
4.1.1 SDLC 단계별 AI 활용과 대표적인 AI 활용 도구 ····· 23
4.1.2 SDLC AI 도입 효과 및 장애 요인 ····· 29
4.2 SDLC 단계별 AI 이용 설문조사 ····· 31
4.2.1 표본의 선정 및 분석 방법 ····· 31
4.2.2 설문조사 결과 분석 ····· 33

4.3 전문가 그룹 인터뷰 ····· 59
4.3.1 인터뷰 개요 ····· 59
4.3.2 인터뷰 참여자 ····· 60
4.3.3 분석 결과 검토 의견 ····· 60
4.4 소결 ····· 65
4.4.1 SDLC 단계별 활용 수준 및 생산성 향상 효과 ····· 66
4.4.2 AI 도입의 장애 요인 및 직무역량 인식 ····· 66

Ⅴ. 결론 및 제언 ····· 68
5.1 연구 결과 요약 및 시사점 ····· 68
5.1.1 연구 결과 요약 ····· 68
5.1.2 시사점 ····· 69
5.2 정책 제언 및 연구의 한계 ····· 70
5.2.1 정책 제언 ····· 70
5.2.2 연구의 한계와 후속 연구 ····· 71

참 고 문 헌 ····· 72
[부록] 설문지 ····· 78
ABSTRACT ····· 93

표 목차

[표 1-1] 연구 방법 ····· 3
[표 1-2] 논문의 구성 ····· 4
[표 2-1] SDLC 단계 명칭 ····· 5
[표 2-2] 폭포수 SDLC의 일반적인 단계 ····· 6
[표 2-3] AI 개발 지원 도구 예시 ····· 8
[표 2-4] 국외 관련 연구 ····· 9
[표 2-1] 국내 관련 연구 ····· 11
[표 3-1] 연구 방법 상세 ····· 14
[표 3-2] 설문 대상자의 IT 분야 경력 조사 기준 ····· 16
[표 3-3] 설문 대상자의 IT 분야 주요 직무 기준 ····· 16
[표 4-1] 주요 AI 코딩 지원 도구 생산성 효과 ····· 25
[표 4-2] 주요 실증 연구(RCT) 결과 비교 ····· 26
[표 4-3] 소프트웨어 개발 단계별 AI 기술 활용 효과 종합 ····· 29
[표 4-4] AI 장애 요인 및 내용 ····· 30
[표 4-5] AI 도입 장애 요인 극복 방안 ····· 31
[표 4-6] 표본의 일반적 특성 ····· 34
[표 4-7] AI 기술 도입 및 일반 활용 현황 ····· 36
[표 4-8] (SDLC 1단계) 요구사항 분석 단계 조사 ····· 38
[표 4-9] (SDLC 2단계) 시스템 설계 단계 조사 ····· 40
[표 4-10] (SDLC 3단계) 구현/코딩 단계 조사 ····· 41
[표 4-11] (SDLC 4단계) 테스트 및 검증 단계 조사 ····· 43
[표 4-12] (SDLC 5단계) 배포 및 유지보수 단계 조사 ····· 44
[표 4-13] 요구사항 분석단계 활용 수준 ····· 46
[표 4-14] 요구사항 분석단계 생산성 향상도 수준 ····· 47
[표 4-15] 시스템 설계 단계 활용 수준 ····· 48
[표 4-16] 시스템 설계 단계 생산성 향상도 수준 ····· 48
[표 4-17] 구현 단계 활용 수준 ····· 49
[표 4-18] 구현 단계 생산성 향상도 수준 ····· 50

[표 4-19] 테스트 단계 활용 수준 ····· 51
[표 4-20] 테스트 단계 생산성 향상도 수준 ····· 51
[표 4-21] 배포 및 유지보수 단계 활용 수준 ····· 52
[표 4-22] 배포 및 유지보수 단계 생산성 향상도 수준 ····· 53
[표 4-23] SDLC 단계별 활용수준-향상도 평균 점수 ····· 54
[표 4-24] AI 도입 시 가장 크게 우려되는 기술적/조직적 문제(등급별) ····· 58
[표 4-25] AI 기술이 향후 귀하의 직무역량에 미칠 영향(등급별) ····· 59
[표 4-26] 인터뷰 참여자 현황 (20년 이상 관리자급 3명) ····· 60
[표 4-27] (인터뷰 요약1) SDLC 단계별 AI 도구 활용 현상 분석 ····· 61
[표 4-28] (인터뷰 요약2) AI 활용수준과 생산성 향상도 결과 분석 ····· 63
[표 4-29] (인터뷰 요약3) AI 도입의 장애 요인 및 인식 분석 (정성적) ····· 64

그림 목차

[그림 4-1] SDLC 단계별 활용수준-향상도 평균점수 상대 위치 ····· 55
[그림 4-2] SDLC 단계별 활용수준-향상도 단계별 상대 비교 ····· 56

HANSUNG UNIVERSITY

Ⅰ. 서론

1.1 연구의 필요성

21세기 디지털 경제에서 소프트웨어(Software)는 산업과 사회 전반의 시스템을 떠받치는 핵심 인프라다. 처리해야 할 데이터가 증가할수록 소프트웨어의 역할도 함께 커지고 있다. 다양하고 새로운 인터넷 서비스가 생성될 때마다 그에 맞는 소프트웨어가 끊임없이 만들어지고 그 복잡성과 규모도 커지고 있다. 이에 따라 기업들은 새로운 정보시스템 구축 프로젝트를 지속적으로 수행해 왔으며, 이를 체계적으로 진행하기 위한 방법론도 함께 발전해 왔다. 현재의 기업들이 기성 패키지 소프트웨어를 그대로 도입하는 사례는 매우 드물다. 요구사항 파악에서 출발하는 프로젝트형 개발이나, 기업의 업무 프로세스를 분석해 연계·통합하는 패키지 커스터마이징 사업이 주를 이룬다.

이러한 개발은 그동안 소프트웨어 산업 종사자인 개발자가 담당해 왔다. 그런데 여기에 생성형 AI라는 새로운 변수가 등장하였다. 2022년 11월 미국 OpenAI는 파운데이션 모델 기반의 대규모 언어 모델(Large Language Model, LLM)을 적용한 생성형 AI(Generative AI) 서비스인 ChatGPT를 출시하였다. 초기에는 학습 데이터의 시점 한계와 환각(hallucination) 문제 때문에 산업 현장에 적용되기까지 오랜 시간이 걸릴 것으로 예측되었으나, 짧은 기간에 전 세계적인 인기를 끌며 많은 사용자를 확보하였다. 이후 구글과 마이크로소프트 등이 생성형 AI 모델을 잇따라 선보였고, 대규모 언어 모델의 학습 한계를 보완하는 검색증강생성(Retrieval-Augmented Generation, RAG) 기술이 더해지며 신뢰성이 크게 높아져 사용이 계속 늘고 있다.

2026년 현재 생성형 AI를 업무에 활용하고 있는 직장인의 매우 흔하게 볼수 있다. 생성형 AI는 사용자의 프롬프트에 따라 구조화된 문장과 이미지, 영상 등 다양한 콘텐츠를 만들어 내며, 소프트웨어 개발 분야에서도 다양한 생성형 AI가 사용되고 있다. GitHub Copilot, Cursor, Amazon CodeWhisperer와 같은 AI 기반 개발 도구가 실무 현장에서 광범위하게 활용되고 있으며(GitHub, 2024), 가트너는 2028년까지 기업 소프트웨어 개발자의 75%가 AI 코드 어

시스턴트를 사용할 것으로 전망하였다(Gartner, 2024). 마이크로소프트와 GitHub의 공동 연구에서는 특정 코딩 과제를 대상으로 한 실험에서 Copilot을 사용한 개발자가 그렇지 않은 개발자보다 평균 55% 빠르게 완료한 것으로 나타났다(Peng et al., 2023). 생성형 AI가 소프트웨어 개발 생산성에 미치는 영향이 그만큼 크다는 점을 알 수 있다.

그러나 이러한 생산성 효과는 대체로 특정 도구나 특정 과제를 대상으로 측정된 단편적 결과에 가깝다. 요구사항 분석부터 배포·유지보수에 이르는 소프트웨어 개발 생명주기(SDLC) 전 단계를 구분하여, 단계마다 생성형 AI의 활용 수준과 생산성의 관계가 어떻게 다른지 실증적으로 비교한 연구는 아직 충분하지 않다. 국내 IT 컨설팅 역시 업무 프로세스 재설계(BPR)나 정보화 전략계획(ISP)에 치우쳐, 개발 단계별 AI 활용 방안을 정면으로 다룬 사례를 찾기 어렵다. 개발 현장에서 생성형 AI를 어떻게 활용해야 생산성을 높일 수 있는지에 대한 단계별 근거가 마련되어야 하며, 여기에 본 연구의 필요성이 있다.

1.2 연구의 목적

생성형 AI가 소프트웨어 개발 생산성을 높인다는 점은 여러 보고서와 선행 연구에서 확인되지만, 그 효과를 개발 단계별로 나누어 살펴본 연구는 드물다. 본 연구의 목적은 요구사항 분석에서 배포·유지보수에 이르는 소프트웨어 개발 생명주기(SDLC) 전 단계에서 생성형 AI의 활용 수준과 실무자가 체감하는 생산성 향상도의 관계를 단계별로 규명하는 데 있다.

이러한 목적을 달성하기 위해 본 연구는 먼저 SDLC 각 단계에서 생성형 AI의 활용 수준을 조사한다. 이어 단계별 활용 수준과 생산성 향상도의 관계를 분석하고, 단계 간 상대적 위치를 비교하여 어느 단계에서 활용 대비 효과가 크고 작은지를 확인한 뒤 시사점을 도출한다. 분석은 실제 개발 프로젝트에 참여하는 현업 개발자를 대상으로 한 설문과, 그 결과에 대한 현장 전문가의 검토를 함께 활용한다.

이 연구의 결과는 두 측면에서 의미를 가진다. 실무적으로는 프로젝트를 기획·관리하는 현장 감독자에게 단계별로 생성형 AI를 어떻게 배치할지에 대한 근거

를 제공하고, 소프트웨어 개발 방법론과 가이드라인을 마련하는 정책 관계자에게도 참고 자료가 된다. 학술적으로는 그동안 개별 도구나 과제 단위로 흩어져 있던 생산성 논의를 SDLC 단계라는 틀 안에서 활용 수준과 함께 비교했다는 데 의의가 있다.

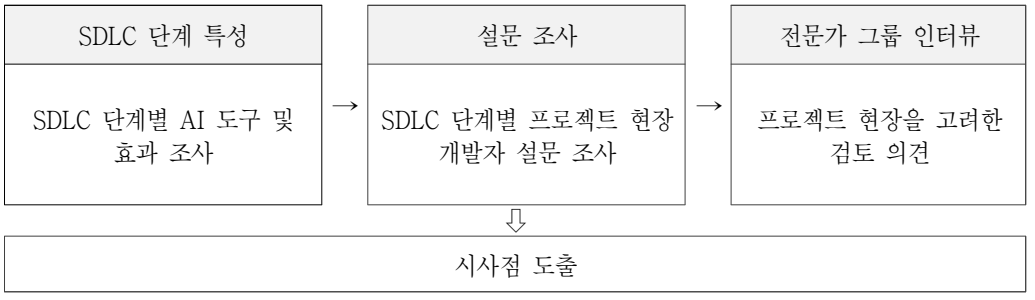
1.3 연구 방법 및 논문의 구성

1.3.1 연구 방법

본 연구는 문헌 검토, 설문조사, 전문가 검토의 세 단계로 진행한다. 먼저 소프트웨어 개발 생명주기(SDLC)의 단계별 특성과 각 단계에서 활용되는 생성형 AI 도구 및 그 효과를 문헌 연구로 정리한다. 이를 바탕으로 실제 개발 프로젝트에 참여하는 현업 개발자를 대상으로 단계별 활용 수준과 생산성 향상도를 묻는 설문조사를 실시한다.

끝으로 현장 경험이 풍부한 전문가 그룹에게 설문 결과를 제시하고 그 해석과 현장 적합성에 대한 의견을 들어, 정량적 설문 결과와 전문가의 정성적 검토를 교차 검증(Triangulation)한 뒤 최종 시사점을 도출한다. 연구의 전체 절차는 [표 1-1]과 같으며, 표본 선정과 측정 도구, 분석 방법 등 구체적인 사항은 제3장에서 기술한다.

[표 1-1] 연구 방법



1.3.2 논문의 구성 체계

본 논문은 총 5장으로 구성된다. 제1장에서는 연구의 배경과 필요성, 목적, 범위와 방법을 기술한다. 제2장에서는 소프트웨어 개발 생명주기와 생성형 AI에 관한 이론적 배경을 검토하고 선행 연구를 고찰한다. 제3장에서는 연구의 분석 틀과 조사·분석 방법을 설계한다. 제4장에서는 SDLC 각 단계의 생성형 AI 활용 방안과 도입 효과 및 장애 요인을 살펴보고, 현업 개발자를 대상으로 한 설문조사 결과와 전문가 그룹 인터뷰를 분석한다. 제5장에서는 연구 결과를 종합하여 시사점과 정책적 제언을 제시하고, 연구의 한계를 정리한다.

[표 1-2] 논문의 구성

1	서론		
연구의 필요성	연구의 목적	연구방법 및 논문의 구성	
⇩			
2	소프트웨어 개발 생명주기와 AI 지원 코딩 이론적 배경		
소프트웨어 개발 생명주기	소프트웨어 개발을 지원하는 AI 도구	소프트웨어 개발 생산성 향상과 AI 도구 활용 수준	
⇩			
3	연구 방법 및 절차		
SDLC 단계 특성	설문 조사	전문가 그룹 인터뷰	시사점 도출
⇩			
4	연구 결과 및 분석		
SDLC 단계별 AI 도구 적용 필요성	SDLC 단계별 AI 도입 효과 및 장애 요인	설문조사 및 결과 분석	전문가 그룹 인터뷰
소결			
⇩			
5	결론 및 제언		
연구 결과 요약 및 시사점			
정책 제언 및 연구의 한계			

Ⅱ. 이론적 배경

2.1 소프트웨어 개발 생명주기(SDLC)

소프트웨어 개발 단계별 AI 기술 활용 방안을 도출하기에 앞서, 관련 용어와 방법론에 대한 개념 정립이 선행되어야 한다. 소프트웨어(Software)는 컴퓨터 시스템 운영과 관련된 프로그램, 절차, 규칙 및 연관 문서 일체를 의미하며, 이를 생성·변경·재사용하는 모든 기술적 활동을 소프트웨어 개발로 정의한다. 전형적인 소프트웨어 개발 프로세스는 사용자 요구사항을 기술적 규격으로 변환하고, 이를 설계와 코드로 구현한 뒤 운영 환경에 맞추어 시험·문서화·검증하는 연속적인 단계들로 구성된다(TTA, 1998).

이러한 전 과정의 절차를 체계화한 것이 생명주기 개념이다. 소프트웨어 생명주기(Software Life Cycle, SLC)가 제품의 기획부터 폐기까지 전체 유효 기간을 포괄한다면, 소프트웨어 개발 생명주기(Software Development Life Cycle, SDLC)는 개발을 결정한 시점부터 제품이 최종 인도되거나 개발자에 의해 보강이 완료될 때까지의 구조화된 방법론적 주기를 의미한다(TTA, 1998; IEEE, 1990). IBM은 이를 개발팀이 효율적인 비용으로 고품질의 시스템 구축, 제공, 유지보수하기 위해 따르는 명확하고 상호 의존적인 단계별 로드맵으로 규정하고 있다(Jackson et al., 2025).

[표 2-1] SDLC 단계 명칭

SWEBOK(IEEE)	단체표준(TTA)	비고
요구사항 분석 (Requirements)	요구사항 분석 (Requirement Phase)	
설계 단계 (Design)	설계 단계 (Design phase)	
구현 단계 (Construction)	구현 단계 (Implementation phase)	
테스트 (Testing)	테스트 (Test Phase)	
배포 및 유지보수 (Deploy & Maintenance Phase)	운영 및 유지보수 단계 (Operation & Maintenance Phase)	

출처: SWEBOK Guide(Washizaki, 2024)와 TTA 단체표준(TTA, 1998)을 바탕으로 연구자가 비교·정리.

소프트웨어 프로젝트 계획 단계에서는 수행 환경에 부합하는 적절한 SDLC 모델의 선택과 테일러링(Tailoring)이 요구된다. 소프트웨어 확장 가이드에 따르면, 프로젝트 수명주기는 예측형(Predictive)에서 적응형(Adaptive)에 이르는 연속체 상에 존재한다(PMI, 2013). 예측형 수명주기는 초기 단계의 철저한 요구사항 명세화와 상세 계획 수립을 통해 위험과 비용을 최소화하며 마일스톤 중심으로 진행된다. 반면, 적응형 수명주기는 짧은 반복 개발 주기를 바탕으로 점진적으로 요구사항을 명세화하고 환경 변화에 유연하게 대응하는 특성을 지닌다(Guimaraes & Nascimento, 2025).

[표 2-2] 폭포수 SDLC의 일반적인 단계

단계	내용	비고
요구사항 분석 (Requirements)	소프트웨어 개발 생명주기의 한 단계로서 시스템의 기능이나 수행능력 같은 소프트웨어 제품을 위한 요구사항을 정의하고 문서화하는 단계	
설계 단계 (Design phase)	소프트웨어 생명주기 중에서 요구사항을 만족하기 위해 소프트웨어의 구조, 소프트웨어의 구성요소, 인터페이스, 자료를 만들고 문서화하고 검증하는 단계	
구현 단계 (Construction) (implementation phase)	소프트웨어 생명주기에서 설계문서로부터 소프트웨어 제품을 만들어내고 오류를 정정하는 단계	
테스트 (Test Phase)	소프트웨어 제품의 구성요소를 평가하고 통합하며 요구가 만족되었는지를 결정하기 위하여 소프트웨어 제품을 평가하는 소프트웨어 생명주기상의 단계	
배포 및 유지보수 (Deploy & Maintenance)	소프트웨어 제품을 운영환경에서 사용하고, 성능의 만족도를 조사하고, 새로운 요구사항에 반응하거나 문제점을 바로잡아 수정하는 동안의 소프트웨어 생명주기상의 단계	

출처: TTA(1998)의 단계별 정의를 바탕으로 연구자가 정리.

전통적인 예측형 모델의 대표적 방법론은 폭포수(Waterfall) 모형으로, 요구사항 분석 → 시스템 설계 → 구현 → 테스트 → 배포/유지보수 단계로 구성되어 있으며, 단계별 순차적 수행과 엄격한 산출물 관리가 특징이다(Royce, 1970). 이에 반해 요구사항 변화에 민첩하게 대응하는 애자일(Agile) 방법론이 등장하였고

(Beck et al., 2001), 최근에는 DevOps 문화의 확산과 더불어 개발 파이프라인 전반에 AI를 통합하는 AI-Native SDLC 방법론까지 제안되고 있다(Hymel, 2024). 본 연구는 소프트웨어 개발 생명주기 전반의 AI 활용 수준과 체감 생산성 향상 효과를 단계별로 명확히 구분하여 정량적으로 비교분석하고자 한다. 따라서 짧은 주기로 단계가 중첩되어 반복되는 애자일 모델보다는, 과업의 경계와 단계 구분이 정형화된 폭포수 모형을 연구의 기준 모델로 채택하였다. SWEBOK 지식 체계와 국내 단체표준에서 정의하는 단계별 명칭 비교 및 폭포수 모델의 일반적인 단계별 정의는 [표 2-1], [표 2-2]와 같다.

2.2 소프트웨어 개발을 지원하는 AI 도구

인공지능(Artificial Intelligence, 이하 AI)은 인간의 지적 능력을 컴퓨터 시스템이 모방하거나 수행하는 기술의 총칭을 의미한다. McCarthy et al.(1956)이 다트머스 회의에서 AI 개념을 정의한 이후, AI는 기계학습(Machine Learning, ML), 딥러닝(Deep Learning, DL), 자연어 처리(Natural Language Processing, NLP), 컴퓨터 비전(Computer Vision) 등 다양한 하위 분야로 발전해 왔다.

소프트웨어 개발과 관련하여 주목받는 AI 기술은 다양하다. GPT-4, Claude, Gemini 등 대형 언어 모델(LLM)은 수십억~수천억 개의 파라미터로 학습된 모델로, 텍스트 이해·생성·변환 능력이 뛰어나 요구사항 분석, 코드 생성, 문서화 등 다양한 SW 개발에 활용된다. 생성형 AI는 AI 기술이 새로운 콘텐츠(코드, 텍스트, 이미지)를 생성하는 기술로, ChatGPT, GitHub Copilot 등 도구가 대표적이다. 코드 특화 모델(Code LLM)은 Open AI의 Codex(Chen et al., 2021), Meta의 Code Llama처럼 코드 데이터로 사전 학습 또는 파인 튜닝된 모델로 코드 자동완성, 생성, 리팩터링 등에 사용할 수 있다. 자동화된 머신러닝(AutoML)은 하이퍼 파라미터 튜닝, 모델 선택 등 ML 파이프라인의 자동화를 지원하며 AI/ML 프로젝트 개발에 활용할 수 있다. AI 에이전트(AI Agent)는 단일 쿼리가 아닌 멀티스텝 태스크를 자율적으로 수행하는 AI로, 복잡한 코딩 작업이나 버그 수정을 자동화하는 데 활용할 수 있다.

개발자의 경험 수준, 조직 환경, 수행하는 태스크의 유형에 따라 AI 도구의

효과가 크게 달라진다는 점이 주요 발견 사항이다. 특히 Becker et al.(2025)의 연구에서 경험 많은 개발자의 경우 AI 도구 사용 시 역설적으로 작업 시간이 증가했다는 발견은 주목할 만하다. 이는 AI 도구가 초급 개발자 또는 루틴 작업에서 높은 효과를 보이지만, 숙련된 개발자의 복잡한 의사결정 과정에서 오히려 방해 될 수 있음을 시사한다. 한편 Butler et al.(2024)의 연구에서는 Microsoft RCT 참여자의 84%가 일상 업무 개선을 경험했다는 보고는 현장 적용성이 높음을 의미하며, 이는 실험실 환경이 아닌 실제 업무 환경에서 AI 도구가 미치는 영향을 보여준다.

[표 2-3] AI 개발 지원 도구 예시

도구유형	대표도구	핵심 기능	적용 SDLC 단계
코드 자동완성	GitHub Copilot	실시간 코드 제안, 함수 자동완성, 오류 교정	구현 단계
채팅 기반 AI	ChatGPT, Claude	복합 태스크(설계, 코드 생성, 리뷰, 문서화)	설계, 구현, 테스트, 문서화
AI-Native IDE	Cursor	멀티파일 편집, 전체 코드베이스 이해, 터미널 통합	구현 단계(통합 환경)
코드 리뷰 AI	Sourcegraph Cody	코드베이스 분석, 리팩터링 제안, 품질 평가	구현 및 배포 전 검토
테스트 자동화	TestGen-LLM	단위 테스트 자동 생성, 커버리지 분석 및 개선	테스트 단계
AIOps	Datadog AI, Dynatrace	지능형 모니터링, 장애 예측, 자동 복구	배포 및 유지보수 단계

자료 : 본 표는 생성형 AI로 초안을 생성하고 연구자가 검토·수정하여 완성

Dohmke et al.(2023)의 연구에서는 대규모 현장 데이터(n=934,533)를 분석하여 GitHub Copilot 제안의 수용률이 약 30%임을 확인하고, 2030년까지 전 세계 GDP에 약 1.5조 달러 규모의 긍정적 영향을 미칠 수 있다는 경제적 함의를 제시하였다. 한편 약 55%의 속도 향상은 통제 실험 기반의 Peng et al.(2023)의 연구에서 확인된 수치이다. 또한 Weisz et al.(2025)의 IBM Watson Code Assistant 연구 사례에서는 코드 이해(43%)가 코드 생성(39%)보다 더 중요한 사

용 사례임을 드러냈는데, 이는 AI 도구의 가치가 단순 코드 생성을 넘어 인지적 부하 감소와 학습 지원에도 있음을 시사한다. Bakal et al.(2025)의 ZoomInfo 연구 사례에서는 33% 수용률, 72% 만족도, 약 20%의 시간 절감은 현실적인 도입 효과를 보여준다. 또한 Guglielmi et al.(2025)의 연구에서 Copilot 제안이 좌절감을 감소시키는 효과가 경험 수준에 따라 상이하다는 발견은 향후 개발자 지원 도구 설계에 중요한 시사점을 제공한다. 표에서 제시한 선행 연구들은 AI 기술이 소프트웨어 개발에 미치는 효과가 매우 다양함을 보여준다.

[표 2-4] 국외 관련 연구

연구자(연도)	연구 내용	주요 결과	한계점
Peng et al. (2023)	무작위 대조 실험을 통한 GitHub Copilot의 개발자 생산성 영향 분석	Copilot 사용 그룹이 55.8% 빠른 과제 완료; 비개발자 진입 장벽 완화 가능성 확인	단일 과제 기반 측정으로 실무 일반화 한계
Coutinho et al. (2024)	소프트웨어 개발 생산성에서 생성형 AI 도구의 역할 파일럿 사례 연구	AI 도구 활용 현황·효과·도전 요인 분석; 반복 작업 감소 및 코드 품질 향상 확인	소규모 조직 대상 파일럿 스터디의 일반화 한계
Weber et al. (2024)	LLM 기반 프로그래밍을 통한 생산성 향상 사용자 연구 (N=24)	GitHub Copilot·GPT-3 활용 시 유의미한 생산성 향상; 도구 유형별 상이한 사용 패턴 관찰	실험실 환경으로 현장 적용 가능성 제한적
Paradis et al. (2024)	Google 내 96명 개발자 대상 기업 수준 무작위 대조 실험	복잡한 기업 과제에서 AI 도구의 효과는 제한적이며 개발자 경험 및 맥락에 따라 상이	단일 기업(Google) 환경으로 다양성 부족
Borg et al. (2024)	AI 어시스턴트 공동 개발이 코드 유지보수성에 미치는 영향 분석	AI 보조 코드의 단기 생산성 향상 확인; 장기 유지보수성 영향은 미결론	장기적 코드 유지보수 영향 측정 어려움
Perry et al. (2023)	AI 코드 어시스턴트 사용 시 보안 취약점 작성 여부 사용자 연구	AI 어시스턴트 사용 그룹이 통계적으로 더 많은 보안 취약점 코드 작성; 과신 경향 관찰	보안 특화 과제 환경으로 일반 개발 적용 시 한계
Dohmke et al. (2023)	GitHub Copilot 현장 효과 실증 분석 (n=934,533)	55% 코딩 속도 향상; 30% 제안 수용율; GDP $1.5T 잠재성 (2030년 예상)	대규모 샘플로 외부타당도 높음
Weisz et al. (2025)	IBM Watson Code Assistant 사용자 연구 (N=669)	코드 이해가 43%로 최상위 사용사례; 코드 생성 39%; 긍정적 생산성 증감 분포	사용자 경험 편차로 보편적 효과 제한적

연구자(연도)	연구 내용	주요 결과	한계점
Butler et al. (2024)	Microsoft 내 무작위 대조 시험 및 일지 연구 (n=200+)	84% 일상업무 개선 보고; 66% 일감에 대한 심리변화; 특히 주니어 개발자 효과 높음	단일 기업 환경으로 다양성 제한
Bakal et al. (2025)	ZoomInfo에서 GitHub Copilot 도입 효과 분석	33% 수용율; 72% 만족도; 약 20% 시간 절감; 실무 도입 사례	비기술적 요인(조직문화, 교육) 고려 부족
Becker et al. (2025)	경험 많은 오픈소스 개발자 RCT (n=16, 246개 과제)	역설적으로 AI 도구 사용 시 경험 많은 개발자의 작업 시간 증가; 경험의 이질성 발견	소규모 샘플(n=16)로 통계 신뢰도 제한; 맥락 의존성 높음
Guglielmi et al. (2025)	Copilot 제안이 개발자 좌절감 및 생산성에 미치는 영향	제안이 좌절감 감소에 기여; 다만 경험 수준에 따라 효과의 편차 관찰	감정적 효과 측정의 주관성; 장기 효과 미실증

자료 : 각 문헌의 내용을 바탕으로 연구자가 요약·정리

국내의 연구 동향을 보면 방미영 외(2026)의 연구에서는 국내 IT 관리자와 개발자 총 924명을 대상으로 상호지향성(Co-orientation) 모형을 적용하여 생성형 AI 코딩 도입에 관한 인식 격차를 분석하였다. 분석 결과 IT관리자가 생산성 향상·산출물 품질·조직지원에 더 긍정적이나, 개발자는 법적·윤리적 이슈에 더 민감하게 반응하는 인식 격차가 실증되었다. 이러한 관리자와 실무자 간의 인식 차이는 AI 도구 도입 시 조직적 변화 관리와 양방향 소통이 필수임을 시사한다. 한편 최종명(2023)의 연구에서는 ChatGPT를 요구 공학 도구로 활용하는 방법론을 제안하여, 자연어 입력만으로 초기 유스케이스 도출 및 요구사항 불일치 탐지를 자동화할 수 있는 가능성을 보였다. 이는 SDLC의 첫 단계인 요구사항 분석에 LLM을 적용하는 국내 초기 연구로서의 의의를 갖는다.

최승윤 외(2023)의 연구에서는 LLM을 활용한 대화형 코드 리뷰 시스템을 개발하고 12인 사용자 평가를 통해 피드백의 적절성·구체성·명확성·유용성 모두에서 긍정적 평가를 받음으로써, LLM 기반 코드 리뷰 자동화의 교육적·실용적 가능성을 입증하였다.

황혜민·이장원(2025)의 연구에서는 ChatGPT를 파트너로 활용하는 AI 기반 짝 프로그래밍 수업모델을 설계·운영하여 학습자의 알고리즘적 사고, 문제 해결 능력, 프로그래밍 자기효능감 향상 효과를 확인하였다. 한편 황진석 외(2025)의

연구에서는 AI와 Vibe Coding을 결합한 Spec-Driven Development(SDD) 방법론을 정립하여 명세→검증→생성→운영의 루프를 체계화하고, 기존 SDLC에서 AI 주도 개발 패러다임으로의 전환 로드맵을 제안하였다.

이 연구들은 SDLC 전 단계에 걸쳐 AI 기술 적용이 확산되고 있음을 보여주며, 특히 국내 환경에 맞는 도입 전략과 조직적 변화 관리의 중요성을 공통으로 강조한다.

[표 2-1] 국내 관련 연구

연구자(연도)	연구 내용	주요 결과	한계점
방미영 외 (2026)	국내 IT 관리자(462명) 개발자(462명) 총 924명 대상 생성형 AI 코딩 도입 관련 인식 격차 탐색 (상호지향성 모형 적용)	IT관리자가 생산성 향상·품질·조직지원에 더 긍정적; 개발자는 법적·윤리적 이슈에 민감; 상호 인식 격차가 도입 저해 요인으로 확인	설문 자기보고로 실제 도입 효과와의 연계 분석 부족
최종명 (2023)	ChatGPT를 소프트웨어 요구공학 도구로 활용하여 요구사항 도출·분석·유스케이스 문서화 자동화 방법론 제안	자연어 입력만으로 초기 유스케이스 도출 및 요구사항 불일치 탐지 자동화 가능성 확인; ChatGPT 기반 요구공학 프로세스 정립	실증 실험 없이 개념적 제안 수준; 실무 환경 적용 시 검증 필요
최승윤 외 (2023)	LLM 기반 대화형 코드 리뷰 학습 시스템 설계 및 개발; 12인 사용자 평가로 피드백 적절성·구체성·명확성·유용성 측정	LLM 기반 코드 리뷰 시스템이 학습자의 코드 오류 수정 능력 향상에 효과적; 4개 평가 기준 모두 긍정적 평가 수렴	교육 환경 대상 실험으로 실제 산업 개발 현장 적용 검증 부재
황혜민, 이장원 (2025)	ChatGPT를 파트너로 활용한 AI 기반 짝 프로그래밍 수업모델 설계·운영; 컴퓨터교육과 학생 19명 사전·사후 설문 및 심층 인터뷰	알고리즘적 사고, 문제해결능력, 프로그래밍 자기효능감에서 유의미한 향상 확인; AI 보조 짝 프로그래밍의 교육적 효과 검증	소규모 표본(n=19) 및 단일 대학 대상으로 외부타당도 제한
황진석 외 (2025)	AI와 Vibe Coding 결합의 Spec-Driven Development(SDD) 방법론 정립 및 전통적 SDLC·워터폴·애자일에서 SDD로의 전환 로드맵 제안	SDD 핵심 구성요소 (명세→검증→생성→운영 루프) 체계화; AI 주도 개발에서 인간 역할이 검토·최종판단으로 전환되는 패러다임 정립	이론·방법론적 제안으로 실증 검증 미완료; 도입 비용·저항 요인 분석 부재

자료 : 각 문헌의 내용을 바탕으로 연구자가 요약·정리함

2.3 소프트웨어 개발 생산성 향상과 AI 도구 활용 수준

국제전기전자공학회(IEEE)에서 발간한 소프트웨어 공학 분야의 핵심 지식 체계(Software Engineering Body of Knowledge, SWEBOK)에서 생산성은 경제적 관점에서 산출량과 투입량의 비율로 설명한다. 산출량은 창출된 가치를 의미하며, 투입량은 산출물을 만들어 내는데 사용된 모든 자원(예: 노력)을 포함한다. 이때 생산성은 가치 지향적 관점에서 효율성과 효과성을 결합한 개념이다. 그러므로 생산성 극대화란 최소한의 자원을 투입 또는 소비하여 최대한의 가치를 창출하는 것을 의미한다.

소프트웨어 개발 생산성(Software Development Productivity)은 소프트웨어를 생산하고 유지보수하기 위해 조직, 기술, 기법 및 자동화 도구 등으로 구성되는 상대적인 능력을 의미한다(TTA, 1998). 소프트웨어 개발 단계에서 AI 기술의 이용은 최근 5년간 급속히 확대되었다. Stack Overflow社에서 진행한 2024년 개발자 설문조사에 따르면 응답자의 76%가 이미 AI 도구를 개발에 활용하고 있거나 활용할 계획이라고 응답하였으며, 이 중 코드 자동완성 기능 사용이 51%로 가장 높은 비율을 차지하였다(Stack Overflow, 2024).

국내에서도 AI 기술 활용에 대한 관심이 높아지고 있다. 「인공지능산업 실태조사」에 따르면 2024년 국내 AI 산업 규모는 6조 3천억 원에 달하며, AI 응용 소프트웨어 분야가 주력 사업의 46.8%를 차지하는 것으로 조사되었다(정보통신기획평가원, 2024). 또한 국내 현황을 살펴보면, SK플래닛이 2024년 수행한 내부 설문조사에서 응답 개발자의 약 87%가 GitHub Copilot에 긍정적인 반응을 보였으며, 사용하지 않을 경우 생산성이 하락할 것이라는 응답이 85.7%에 달했다(SK플래닛, 2024). 그러나 국내 AI 산업의 전체 인력 부족률이 7.4%에 달하고, 특히 핵심 공급 주체인 AI 개발자 직군의 부족률이 6.5%를 기록하는 등 인력난이 심각한 현실이다(정보통신기획평가원, 2024). 또한 체계적인 AI 활용 방법론이 미흡한 현실에서, 소프트웨어 개발 현장의 AI 기술 도입과 활용은 표준화된 프로세스 없이 개별 조직이나 개발자의 경험적 판단과 임기응변에 의존하는 수준에 머물러 있다.

2.4 선행 연구 검토 요약

AI와 소프트웨어 개발의 융합에 관한 선행 연구는 크게 세 가지 흐름으로 구분할 수 있다. 첫 번째 흐름은 코드 생성 및 자동완성에 관한 연구이다. Peng et al.(2023)의 연구에서는 무작위 대조 실험을 통해 GitHub Copilot 사용 그룹이 동일 과제를 55.8% 빠르게 완료함을 확인하였다. Weber et al.(2024)의 연구에서는 LMU 뮌헨에서 GitHub Copilot 및 GPT-3 활용 시 통계적으로 유의미한 생산성 향상이 나타남을 확인하였다. 두 번째 흐름은 AI 기반 테스트 자동화 연구이다. Alshahwan et al.(2024)의 연구에서는 LLM을 활용한 단위 테스트 자동 생성 방법론을 제안하였으며, Meta에서 수행한 TestGen-LLM 프로젝트에서는 75%의 테스트가 정상 빌드되고 57%가 안정적으로 통과하는 결과를 확인하였다 (Alshahwan et al., 2024). Wang et al.(2025)의 연구에서는 Context-Based RAG를 활용한 AI Copilot for Testing 시스템을 제안하여 코드 생성에서 소프트웨어 테스팅까지 통합 파이프라인을 구현하였다. 세 번째 흐름은 AI-Native SDLC 및 방법론적 연구이다. Hymel(2024)의 연구에서는 AI를 SDLC 전반에 통합한 AI-Native 소프트웨어 개발 생명주기 방법론을 이론적으로 정립하고 V-Bounce 모델을 제안하였다.

Paradis et al.(2024)의 연구에서는 Google 내 96명의 full-time 소프트웨어 엔지니어를 대상으로 한 RCT에서 기업 수준의 복잡한 태스크에서 AI 도구의 영향을 측정하였다. 한편 Borg et al.(2024)의 연구에서는 AI 어시스턴트와의 공동 개발이 코드 유지보수성에 미치는 영향을 분석하였고, Perry et al.(2023)의 연구에서는 AI 코드 어시스턴트 사용 시 보안 취약점 발생 가능성을 실증 연구를 통해 규명하였다. 이러한 선행 연구들은 AI 기술이 소프트웨어 개발의 특정 단계에서 유의미한 효과를 보임을 입증하고 있다. 그러나 대부분의 연구가 구현(코딩) 단계에 집중되어 있으며, SDLC 전 단계에 걸친 통합적 활용 방안과 국내 개발 환경에 맞는 구체적 지침을 제시하는 연구는 부족한 실정이다. 이 연구는 이러한 연구 공백을 메우고자 한다.

Ⅲ. 연구 방법 및 절차

3.1 연구 방법

이 연구는 소프트웨어 개발 생명주기(SDLC) 5단계 전반에 걸쳐 실무에서 활용되는 AI 도구의 유형과 적용 방식을 파악하고, 이에 따른 정량적 생산성 변화와 실질적 도입 효과를 분석한다.

이를 위해 먼저 검증된 국내외 학술 문헌 및 사례 분석을 진행하여 개발 단계별 특성과 주요 AI 도구의 활용 실태, 선행 연구에 나타난 한계점을 이론적으로 고찰한다. 도출된 이론적 배경을 바탕으로 통계적 중립성과 객관성을 확보한 설문지를 설계하며, 산업 현장에서 AI 도구를 도입하여 프로젝트를 수행 중인 소프트웨어 개발자를 대상으로 정량적 데이터를 수집한다.

이어서 설문 분석 결과의 실무적 타당성을 검증하기 위해, 개발 프로세스와 현장 상황에 대한 높은 이해도를 갖춘 프로젝트 관리자(PM)급 이상의 전문가를 대상으로 심층 검토 및 교차 검증(Triangulation)을 수행한다. 최종적으로 정량적 설문 결과와 전문가의 정성적 견해를 종합하여 단계별 AI 기술 활용의 시사점을 도출한다. 구체적인 연구 방법은 [표 3-1]과 같다.

[표 3-1] 연구 방법 상세

SDLC 단계 특성			→	설문 조사			→	전문가 검토	
SDLC 단계별 AI 도구 활용 생산성 연구				SDLC 단계별 AI 이용 설문조사				프로젝트 현장을 고려한 검토 의견	
단계별 AI 활용	대표적인 AI 도구	도입 효과 및 장애 요인		표본 선정	설문설계 및 설문조사	설문조사 결과		인터뷰	결과 정리
⇩									
시사점 도출									

3.2 연구 절차

3.2.1 SDLC 단계별 AI 도구 활용 연구

문헌조사와 사례연구는 소프트웨어 개발 생명주기에 적용되는 AI 기술로 한정하며, 하드웨어 수준의 AI 칩 설계나 AI 모델 자체의 개발은 연구 범위에서 제외한다. 또한 소프트웨어 개발을 의뢰한 발주처의 기획 단계의 활용도 제외한다.

AI 기술의 등장을 고려하여 2020년부터 2025년까지의 최근 5년간 선행 연구 및 산업 현황을 중심으로 조사하고, 필요에 따라 관련된 이전의 연구 내용도 포함한다.

문헌 고찰에서는 IEEE, ACM 등 국제 학술 데이터베이스 및 RISS, DBpia 등 국내 학술 데이터베이스에서 관련 논문을 수집·분석한다. 사례 분석에서는 GitHub, Microsoft, AWS, Meta 등 선도기업의 AI 도구 도입 사례와 측정된 효과를 검토한다. 또한 Gartner, McKinsey, Forrester 등 리서치 기관의 보고서를 통해 산업 현황 및 미래 전망 데이터를 이용하여 보완한다.

3.2.2 SDLC 단계 특성 분석

3.2.2.1 SDLC 단계별 AI 도구 활용 표본 설계

1) 응답자 기본 정보 (Demographics)

일반적인 산업군의 설문조사에서는 직급(사원~부장등)을 기준으로 경력을 분류하지만, 본 연구는 소프트웨어 산업의 특성을 반영하여 현장에서 통용되는 '소프트웨어 기술자 등급'을 사용한다.

그러나 종전의 단순 연차 중심(투입공수 방식) 기술 등급은 고도화된 현재의 개발 환경에서는 실무자의 실제 역량과 업무 난이도를 객관적으로 반영하는 데 한계가 있다(한국인공지능·소프트웨어산업협회, 2025). 이에 본 연구는 현행 한국소프트웨어산업협회의 'IT분야 직무역량체계(ITSQF)'가 38개 직무를 세분화하고 현장 직무역량을 강조하는 최근의 개편 방향성을 반영하여 설문 구간을 재구성하였다.

구체적으로 실무 주도 계층을 초급(3년 미만), 중급(3년~7년), 고급(7년~12년)으로 분류하였다. 특히, 12년 차 이상의 기술자를 단순 통합 할 경우 발생하는 통계적 오류를 방지하기 위해, 실질적 아키텍트 역할을 수행하는 특급기술자(12년~17년) 구간과, ITSQF의 최상위 레벨에 해당하는 프로젝트 총괄 및 의사결정권자 수준의 관리자/전문가(18년 이상) 구간을 분리하였다.

[표 3-2] 설문 대상자의 IT 분야 경력 조사 기준

구분	경력기간	비고
초급기술자	3년 미만	학사 학위/기사 자격증 취득자 기준
중급기술자	3년 이상 ~ 7년 미만	
고급기술자	7년 이상 ~ 12년 미만	
특급기술자	12년 이상 ~ 18년 미만	
관리자/전문가 수준	18년 이상	소프트웨어 기술자 등급에 없는 기준

출처 : 한국인공지능·소프트웨어산업협회(2025)의 SW기술자 등급 기준을 바탕으로 연구자가 재구성.

2) 소프트웨어 개발 분야 주요 직무

소프트웨어 기술자의 직무는 IT기획자, IT컨설턴트, 업무분석가, 데이터 분석가, IT PM, IT아키텍트, UI/UX 기획 개발자, UI/UX 디자이너, 응용SW개발자, 시스템SW개발자, 정보시스템운용자, 정보보안전문가 등 17개로 구분할 수 있다.

[표 3-3] 설문 대상자의 IT 분야 주요 직무 기준

구분	내용	비고
요구사항 분석 (프로젝트 관리·기획)	요구사항 분석, 일정 관리, 거버넌스 수립 주도	PM, PMO, 기획(사업관리 등)
시스템 설계	아키텍처 설계, DB 모델링, 파트 리딩 및 기술 결정	TA, AA, Architect, PL
어플리케이션 개발	코어 로직 구현, API 개발 및 UI/UX 구현	Backend, Frontend Developer
품질 관리 및 테스트	테스트 케이스 작성, 검증 자동화 및 품질 분석	QA, Tester, SDET
인프라 운영	배포 파이프라인 구축, 인프라 설계 및 유지보수	DevOps, SRE, Cloud Engineer

출처: 한국인공지능·소프트웨어산업협회(2025)의 SW기술자 직무 분류를 SDLC 단계 기준으로 연구자가 재구성.

그러나 이 연구에서는 SDLC 단계를 기준으로 진행하고 있으므로 프로젝트 관리 및 기획, 시스템 설계 및 리딩, 어플리케이션 개발, 품질 관리 및 테스트, 인프라 운영 총 5개 직무로 구분하여 진행한다.

3.2.2.2 SDLC 단계별 AI 도구 활용 설문 설계

1) 요구사항 분석 단계의 활용 도구와 주요 효과

SDLC의 요구사항 분석 단계에서 주로 활용되는 AI 도구는 네 가지 유형으로 구분할 수 있다. 첫 번째, 범용 대화형 LLM 도구이다. ChatGPT, Claude, Gemini 등이 이에 해당하며, 주로 요구사항의 초안을 작성하거나 브레인스토밍 및 아이디어 도출 과정에 활용할 수 있다. 두 번째, 회의 및 음성 분석 AI 도구이다. 클로바노트(Clova Note), Otter.ai, MS Copilot 등이 대표적이며, 고객 및 이해관계자와의 미팅 녹취 데이터를 기반으로 핵심 요구사항을 추출하고 요약하는 데 사용된다. 세 번째, AI 연동형 문서 및 이슈 관리 플랫폼이다. Notion AI, Atlassian Intelligence(JIRA, Confluence) 등 기존 협업 도구에 내장된 AI를 뜻하며, 요구사항 정의서 작성, 이슈 트래킹, 명세서 정형화 업무의 효율성을 높인다. 네 번째, 다이어그램 및 시각화 AI 도구이다. Eraser.io, Mermaid AI, Whimsical 등이 해당하며 프로세스 매핑, 화면 설계, 유스케이스 다이어그램 및 워크플로우 시각화에 적용할 수 있다.

SDLC 요구사항 분석 단계에서 AI를 이용한 주요 효과는 다섯 가지로 정리할 수 있다. 첫 번째는 요구사항 구체화 및 초안 작성 기간 단축이다. 아이디어 브레인 스토밍에서부터 요구사항 목록 초안 생성까지 물리적 시간을 절감할 수 있다. 두 번째는 요구사항 간 논리적 모순 및 누락 식별이다. 텍스트 간의 불일치를 탐지하거나 정의되지 않은 예외 케이스를 발견하여 분석 품질을 향상 시킬 수 있다. 세 번째는 표준화된 산출물 자동 생성이다. 요구사항 명세서, 사용자 스토리, 유스케이스 명세서 등을 정해진 템플릿에 맞춰 자동 구조화할 수 있다. 네 번째는 비즈니스 또는 의사소통 영역이다. 모호한 비즈니스 요구사항을 개발자가 이해 가능한 기술적 상세 규격이나 데이터 모델로 변환할 수 있다. 다섯 번째는 이해관계자 간 의사결정 및 합의 지원이다. 유스케이스 시각화나 프로토타입 시나리오 생성으로 고객과 PM, 개발자 간 명확한 의도 파악 및 소통을 강화할 수 있다.

2) 시스템 설계의 활용 도구와 주요 효과

SDLC 시스템 설계 단계에서 주로 활용하는 AI 도구는 네 가지 유형으로 구분할 수 있다. 첫 번째는 범용 대화형 LLM이다. 아키텍처 패턴 비교, API 스펙(OpenAPI, Swagger) 및 스키마 구조 설계 등에 사용할 수 있다. 대표적인 AI 도구는 ChatGPT, Claude, Gemini 등이 해당한다. 두 번째는 AI 기반 다이어그램/시각화 도구이다. 텍스트/코드 기반 시스템 구성도, 시퀀스/클래스 다이어그램 등을 자동 생성할 수 있다. 대표적인 AI 도구는 Eraser.io, Mermaid AI, Draw.io AI 등이 해당한다. 세 번째는 클라우드 및 인프라 설계 특화 AI 이다. 클라우드 아키텍처 추천 및 프로비저닝 스크립트 설계에 사용할 수 있다. 대표적인 AI 도구로는 Amazon Q, Google Cloud Gemini, 인프라스트럭처 애즈 코드(IaC) 보조 AI 등이 해당한다. 네 번째는 데이터베이스/ERD 특화 AI 도구이다. DB 정규화 검토, 스키마 모델링 및 인덱스 설계 보조 등을 할 수 있다. 대표적인 AI 도구로는 AI 지원 ERD 툴, SQLDBM, DataGrip AI 등이 해당한다.

SDLC 시스템 설계 단계에서 AI를 이용한 주요 효과는 다섯 가지로 정리할 수 있다. 첫 번째는 초기 설계 및 산출물 작성 시간 단축이다. API 명세서, DB 스키마 보일러플레이트(기본 구조) 생성 및 설계 문서화 속도 향상을 기대할 수 있다. 두 번째는 최적의 아키텍처 및 기술 대안 비교이다. 모놀리식 또는 MSA, RDBMS 또는 NoSQL 등 요구사항에 적합한 디자인 패턴 및 기술 스택 결정을 지원할 수 있다. 세 번째는 다이어그램 등 시각적 구조화 자동화이다. 복잡한 비즈니스 로직이나 시스템 컴포넌트 간의 상호작용을 다이어그램으로 즉각 변환하여 가시성을 확보할 수 있다. 네 번째는 구조적 결함 및 보안 병목 사전 식별이다. 설계 단계에서 단일 장애점(Single Point Of Failure, SPOF), 트랜잭션 충돌 가능성, 데이터 무결성 위협 등을 사전 검토할 수 있다. 다섯 번째는 설계 표준화 및 일관성 유지이다. API 네이밍 컨벤션, 데이터 타입, 인터페이스 규격 등을 팀 또는 업계 표준에 맞게 통일할 수 있다.

3) 소프트웨어 구현, 코딩의 활용 도구와 주요 효과

SDLC 소프트웨어 구현, 코딩 단계에서 주로 활용하는 AI 도구는 네 가지 유형으로 구분할 수 있다. 첫 번째는 범용 대화형 LLM이다. 외부 웹브라우저에서

로직 질의응답 및 코드 스니펫 복사/붙여 넣기 등에 활용할 수 있다. 대표적인 AI 도구는 ChatGPT, Claude, Gemini 등이 해당한다. 두 번째는 IDE 확장 플러그인이다. 기존 에디터 내에서 실시간으로 소스코드나 함수 등을 자동 완성할 수 있다. 대표적인 AI 도구는 GitHub Copilot, JetBrains AI, Amazon Q 등이 해당한다. 세 번째는 AI 네이티브 에디터이다. 프로젝트 전체 맥락을 이해하고 다중 파일 단위로 코드를 수정, 생성할 수 있다. 대표적인 AI 도구는 Cursor, Windsurf 등이 해당한다. 네 번째는 특화 생성기 및 자율형 에이전트이다. 프롬프트만으로 UI 컴포넌트나 전체 모듈을 자율적으로 구현할 수 있다. 대표적인 AI 도구는 v0.dev, bolt.new, Aider, Devin 등이 해당한다.

SDLC 소프트웨어 구현, 코딩 단계에서 AI를 이용한 주요 효과는 다섯 가지로 정리할 수 있다. 첫 번째는 상용구 및 반복 코드 자동 생성이다. 단순 타이핑을 줄여 물리적인 코딩(구현) 시간을 단축할 수 있다. 두 번째는 복잡한 로직 및 알고리즘 구현을 보조하는 것이다. 정규표현식, 복잡한 쿼리문, 특정 자료구조 알고리즘 등 난도 높은 코드 작성을 지원한다. 세 번째는 레거시 코드 해석 및 주석 자동화이다. 타인이 작성했거나 오래된 기존 코드의 의도를 파악하고 문서화, 주석 작성 시간을 절감할 수 있게 한다. 네 번째는 코드 가독성 향상 및 리팩터링이다. 클린코드 원칙이나 디자인 패턴에 적용하여 기존 코드 구조를 최적화할 수 있다. 다섯 번째는 신규 기술 및 프레임워크 학습 곡선의 단축이다. 익숙하지 않은 언어나 API 사용 시 실시간 예제와 가이드를 제공받아 학습 및 적용 시간을 단축할 수 있다.

4) 소프트웨어 테스트, 검증의 활용 도구와 주요 효과

SDLC 소프트웨어 테스트, 검증 단계에서 주로 활용하는 AI 도구는 네 가지 유형으로 구분할 수 있다. 첫 번째는 범용 대화형 LLM이다. 테스트 시나리오 초안 작성 및 테스트 데이터 생성에 활용할 수 있다. 대표적인 AI 도구는 ChatGPT, Claude, Gemini 등이 해당한다. 두 번째는 유닛 테스트 자동 생성 도구이다. 기존 코드 분석을 통한 단위 테스트 코드 및 보일러플레이트 자동 생성에 활용할 수 있다. 대표적인 AI 도구는 Diffblue Cover, TestGen-LLM, GitHub Copilot 등이 있다. 세 번째는 자율형 E2E 및 QA 에이전트이다. 사용

자 시나리오 기반의 종단(E2E) 간 테스트 자동화 및 UI 변경에 대응하여 활용할 수 있다. 대표적인 AI 도구는 mabl, Sauce Labs AI, Blinq.io 등이 해당한다. 네 번째는 AI 기반 정적 분석 및 버그 탐지 도구이다. PR 단계에서의 자동 코드 리뷰, 보안 취약점 및 런타임 버그 사전 탐지 등에 사용할 수 있다. 대표적인 AI 도구는 CodeRabbit, Qodo Merge, Snyk, SonarQube AI 등이 해당한다.

SDLC 소프트웨어 테스트, 검증 단계에서 AI를 이용한 주요 효과는 다섯 가지로 정리할 수 있다. 첫 번째는 테스트 커버리지 확대 및 엣지 케이스 발견이다. 수동으로 확인하기 어려운 예외 상황을 탐지하여 미커버 영역을 최소화할 수 있다. 두 번째는 테스트 스크립트 작성 및 유지보수 기간 단축이다. 테스트 코드 작성 속도 향상 및 UI 변경 시 테스트가 자동 수정되는 자기 치유 기능으로 관리를 효율화할 수 있다. 세 번째는 결함 탐지 정확도 향상 및 근본적 원인 분석 지원이다. 단순 버그 발견뿐만이 아니라 로그 분석을 통해 에러 발생 원인과 수정 방안을 제시할 수 있다. 네 번째는 테스트 시나리오의 다양성 및 현실성 확보이다. 실제 사용자 행동 패턴을 모사한 다양한 테스트 시나리오와 대량의 합성 데이터를 생성할 수 있다. 다섯 번째는 개발-QA 협업 효율성 및 배포 신뢰성 향상이다. 코드 리뷰 자동화와 신속한 피드백 루프로 결함 수정 시간 단축과 안정적인 배포를 지원할 수 있다.

5) 소프트웨어 배포, 유지보수의 활용 도구와 주요 효과

SDLC 소프트웨어 배포·유지보수 단계에서 주로 활용하는 AI 도구는 네 가지 유형으로 구분할 수 있다. 첫 번째는 지능형 통합 모니터링/AIOps 플랫폼이다. 실시간 지표 분석 및 이상 징후 탐지에 사용할 수 있다. 대표적인 AI 도구는 Datadog Bits, Dynatrace Davis, New Relic AI 등이 해당한다. 두 번째는 로그 분석 및 패턴 인식 특화 AI이다. 대량의 비정형 로그 데이터 필터링 및 에러 패턴 식별에 사용할 수 있다. 대표적인 AI 도구는 Splunk AI, Elastic AI, Sumo Logic 등이 해당한다. 세 번째는 인시던트(Incident) 대응 및 협업 보조 AI이다. 장애 상황 요약, 담당자 할당 및 유사 장애 이력 검색 등에 사용할 수 있다. 대표적인 AI 도구는 PagerDuty AI, Slack AI, MS Teams Copilot 등이 해당한다. 네 번째는 릴리스 및 운영 문서화 AI이다. 코드 변경점 기반의 릴리스 노트, 기

술 매뉴얼 및 장애 보고서 작성 등에 활용할 수 있다. 대표적인 AI 도구는 ChatGPT/Claude, Jasper, 전용 릴리스 노트 생성기 등이 해당한다.

SDLC 소프트웨어 배포·유지보수 단계에서 AI를 이용한 주요 효과는 다섯 가지로 정리할 수 있다. 첫 번째는 장애 복구 시간 단축 및 가용성 향상이다. 인시던트 발생 시 실시간 알림과 빠른 원인 분석으로 서비스 중단 시간을 최소화할 수 있다. 두 번째는 근본 원인 분석(RCA)의 자동화 및 정확도 향상이다. 복잡한 마이크로 서비스 아키텍처(MSA) 환경에서 장애의 시작점과 연쇄 파급 효과를 지능적으로 추적할 수 있다. 세 번째는 운영 문서화 및 보고 업무의 효율화이다. 릴리스 노트, 운영 가이드, 고객 대응 FAQ 등 반복적인 문서 작성 시간을 절감할 수 있다. 네 번째는 선제적 모니터링을 통한 장애 예방이다. 과거 데이터를 기반으로 시스템 임계치 초과나 성능 저하 징후를 사전에 탐지하여 대응할 수 있다. 다섯 번째는 운영 데이터 기반의 의사결정 지원이다. 로그 및 성능 데이터를 분석하여 인프라 증설, 비용 최적화, 코드 수정 우선순위 등을 도출할 수 있다.

3.2.2.3 AI 도입의 장애 요인 및 인식

1) AI 도입 시 우려되는 기술적 조직적 문제

이 연구는 SDLC 단계별 AI 활용을 다루고 있으나, 기술적·조직적 문제로 인해 AI 도입이 저해될 수 있다고 판단하였다. 이에 프로젝트 관리자, 경영진, 발주처의 변화 관리를 지원하기 위한 기초 자료로서 관련 문제점에 대한 설문 문항을 구성하였다.

연구자가 도출한 기술적, 조직적 문제는 총 5가지이다. 첫 번째는 보안 및 지식재산권 유출 문제이다. 소스코드가 외부로 유출되거나 데이터 보안 정책과 맞지 않아서 사용할 수 없는 경우이다. 두 번째는 기술적 불확실성 및 환각 현상 문제이다. 결과물의 정확성을 신뢰할 수 없으면 검증 시간이나 비용이 증가하는 경우이다. 세 번째는 투자 대비 효율(ROI)의 불투명성이다. 도구 도입 비용 및 유지보수 대비 실질적 성과 측정이 어려운 문제를 의미한다. 네 번째는 사내 AI 숙련도 부족 및 교육 부재이다. 구성원의 AI 리터러시 격차로 인한 도입 속도 저하 문제를 의미한다. 다섯 번째는 기존 소프트웨어 개발 프로세스(SDLC) 및

조직 문화와의 부조화이다. 도구 도입에 따른 업무 흐름의 변화 및 조직적 저항 문제를 의미한다.

2) AI 기술이 향후 소프트웨어 개발 직무에 미칠 영향

이 연구는 SDLC 단계의 AI 활용에 대해 다루고 있으나, 현장 개발자가 AI 기술로 인한 직무 역량 변화를 어떻게 인식하는지 확인하는 설문 문항을 추가하였다.

연구자가 도출한 직무 역량의 변화는 총 5가지이다. 첫 번째는 업무 생산성 극대화이다. 단순 반복 업무 자동화로 설계, 아키텍처 등 고부가가치 업무에 시간과 역량을 집중할 수 있게 되는 것을 의미한다. 두 번째는 직무 역할의 전환이다. 가장 시간이 많이 걸리는 코드 작성자에서 AI를 관리·조정하는 오케스트레이션으로의 변화를 의미한다. 세 번째는 전문성 저하와 도구 의존이다. AI 의존도 심화로 인한 문제 해결 능력 및 기초 기술 역량 약화를 의미한다. 네 번째는 지속적 학습에 대한 심리적 압박이다. 급변하는 AI 기술의 습득을 위한 인지적 과부하 및 번 아웃을 우려하는 것을 의미한다. 다섯 번째는 직무 대체 가능성 및 고용 불안이다. AI 수행 범위 확대로 인한 인간 개발자의 역할 축소 및 경쟁 심화를 의미한다.

3.2.3 전문가 인터뷰 기반 설문 결과 리뷰

프로젝트 현장에서 소프트웨어 개발 단계에 따라 업무를 수행 중인 전문가를 대상으로 설문조사 결과를 제시하고 현상에 대한 해석과 현장 활용 가능성에 대해 인터뷰하고 그 결과를 요약한다.

Ⅳ. SDLC 단계별 AI 활용 연구

4.1 SDLC 단계별 AI 도구 활용 생산성 연구

4.1.1 SDLC 단계별 AI 활용과 대표적인 AI 활용 도구

4.1.1.1 소프트웨어 요구사항 분석 단계

소프트웨어 요구사항 분석 단계는 이해 관계자의 비즈니스 요구사항을 수집하고, 소프트웨어가 수행해야 할 기능적 및 비기능적 요구사항을 명세화하는 과정이다. 이 단계는 SDLC에서 가장 중요하면서도 오류가 발생하기 쉬운 단계로, 요구사항 오류로 인한 재작업 비용은 이후 단계에서 발견될수록 기하급수적으로 증가한다(Boehm & Basili, 2001). 전통적으로 요구사항 분석은 발주처의 담당자와 소프트웨어 개발 공급사 간 직접 의사소통하면서 소프트웨어 개발 요건을 정의하는 인간 중심의 활동이었다. 그러나 자연어 처리(NLP)와 LLM의 이용으로 이 단계에서도 AI의 활용이 가능해졌다. Guimaraes & Nascimento(2025)의 연구에 따르면 LLM은 요구사항 추출, 모호성 탐지, 요구사항 추적성 분석 등에서 유의미한 성과를 보였다. 요구사항 문서화에는 LLM 사용한다. ChatGPT, Claude, Gemini와 같은 LLM 도구를 활용하여 이해관계자 인터뷰 내용이나 사용자 스토리를 자동으로 구조화된 요구사항 명세서로 변환한다. Cannizzaro et al.(2025)의 연구에 따르면 MITRE社의 NLP 기반 RAAM(Requirements Analysis using AI for Mapping) 시스템이 요구사항 생성, 품질 개선, 추적성 분석을 자동화하고 있다. LLM으로 문서화한 요구사항은 AI 기술을 활용하여 자동으로 검증할 수도 있다. AI가 요구사항의 모호성·불완전성·충돌을 자동 탐지하여 오류를 찾아내고 구체화하여 품질을 향상시킨다.

Guimaraes & Nascimento(2025)의 패널 연구에서 LLM이 요구사항 검토에서 인간 검토자와 유사한 수준의 모호성 탐지 능력을 보임을 확인하였다. 요구사항 분석 단계에서 AI 도입의 핵심 가치는 분석 속도 향상과 일관성 확보에 있다. 그러나 AI가 생성한 요구사항은 반드시 도메인 전문가의 검토를 거쳐야 하며, AI의 환각(Hallucination) 현상으로 인한 오류에 주의할 필요가 있다.

4.1.1.2 소프트웨어 설계 단계

소프트웨어 설계 단계는 요구사항 명세서나 정의서를 바탕으로 시스템 아키텍처, 컴포넌트 구조, 데이터베이스 스키마, API 인터페이스 등을 정의하는 과정이다. 이 단계의 설계 결정은 이후 모든 개발 활동과 결과물에 영향을 미치는 고도의 전문성과 경험을 요구한다.

AI는 소프트웨어 설계 단계에서 아키텍처 패턴 추천, UML 다이어그램 자동 생성, API 명세 자동화 등에 활용될 수 있다. Hymel(2024)의 AI-Native SDLC 연구에서는 AI가 비즈니스 의도를 아키텍처 설계로 자동 변환하는 "Mob Construction" 접근법을 제안하였으며, AWS(2024)에서는 Amazon Bedrock을 활용한 AI 기반 아키텍처 설계 지원 서비스를 제공하고 있다.

AI 기반 아키텍처 추천 서비스는 ChatGPT, Claude 등이 제공하고 있다. LLM에 요구사항과 제약 조건을 입력하면 마이크로서비스, 레이어드 아키텍처 등 적합한 아키텍처 패턴과 구성 요소를 추천받을 수 있다. 설계 단계에는 표나 그림 등 다이어그램의 작성이 필요하다. 이때 Mermaid AI, PlantUML + LLM 조합을 활용하여 자연어 설명으로부터 UML 클래스 다이어그램, 시퀀스 다이어그램, ER 다이어그램을 자동으로 생성할 수 있다.

최근에는 외부 데이터를 연결하여 이용하는 정보시스템 구축 프로젝트가 많은데 이때 외부 API 이용 시 설계를 자동화할 수 있다. AI를 활용하여 OpenAPI(Swagger) 명세를 자동 생성하고, API 일관성 검증 및 문서화를 자동화할 수 있다. 또한 Sourcegraph Cody, GitHub Copilot Chat 등을 활용하여 기존 코드베이스의 아키텍처를 분석하고 개선 방향을 제안받을 수 있다.

4.1.1.3 소프트웨어 구현 단계

소프트웨어 구현 단계는 SDLC에서 AI 기술 활용이 가장 활발하게 이루어 질 수 있는 단계이다. 많은 인적 자원과 시간이 투입되는 단계이다. 이 단계에서는 코드 자동완성, 코드 생성, 리팩터링, 버그 수정 등 다양한 코딩 태스크에 AI가 통합되고 있으며, 그 효과 또한 실증 연구를 통해 검증되고 있다. 가트너는 2028년까지 전문 개발자의 75%가 AI 기반 코딩 도구를 활용할 것으로 전망하

며, 이를 "AI-Augmented Development"의 핵심 영역으로 규정하고 있다(Gartner, 2024). CIO Korea(2024)에 따르면 2024년 기준 AI 코딩 도구로 개발자 생산성을 향상 시키는 기업의 비율이 빠르게 증가하고 있다.

[표 4-1] 주요 AI 코딩 지원 도구 생산성 효과

도구	주요 기능	생산성 향상	출처
GitHub Copilot	코드 자동완성 생성	55% 속도 향상	Dohmke et al.(2023)
ChatGPT / Claude	코드 생성/리팩터링	35-45% / 20-30% 향상	McKinsey(2023)
Amazon Q Dev	보안 점검, 코드 생성	AWS 환경 최적화	AWS(2024)
Cursor IDE	AI-native 편집환경	멀티파일 편집 자동화	Cursor(2024)

주: 본 표는 생성형 AI로 초안을 생성하고 연구자가 검토·수정하여 완성함.

소프트웨어 개발 단계에서 사용되는 소프트웨어를 살펴보면 생산성 향상에 관한 효과를 알 수 있다. GitHub Copilot은 OpenAI의 Codex 모델 기반으로 코드 자동완성, 함수 생성, 주석에서 코드 생성 등을 지원한다.

Microsoft·GitHub 실증 연구(Peng et al., 2023)에서 Copilot 사용 개발자가 동일 과제를 평균 55% 빠르게 완료하였으며, 2024년 조사에서 사용자의 88%가 코딩 속도 향상을 경험하고 75% 이상이 업무 만족도 향상을 보고하였다. Cursor와 Cursor IDE는 GPT-4 기반의 AI 코딩 환경으로 전체 코드베이스를 이해하고 멀티파일 편집, 터미널 명령 제안 등 고급 기능을 제공하는 AI-native 에디터이다. Amazon CodeWhisperer와 Amazon Q Developer는 AWS 환경에 최적화된 코드 생성 도구로 AWS SDK 사용 코드를 자동 생성하고 보안 취약점을 실시간으로 탐지한다. 채팅 방식의 이용을 지원하는 ChatGPT와 Claude는 복잡한 알고리즘 설계, 전체 모듈 코드 생성, 코드 리뷰 및 설명, 디버깅 지원에 활용한다. McKinsey(2023)의 연구에서 코드 문서화 45~50%, 코드 생성 35~45%, 리팩터링 20~30% 생산성 향상이 확인되었다. 구현 단계에서의 AI 효과를 정량적으로 검증한 무작위 대조 실험(Randomized Controlled Trial, RCT) 연구의 결과를 비교하면 주요 결과와 시사점을 볼 수 있다. 이 연구는 AI 도구의 객관적 효과를 측정하는 데 가장 높은 신뢰도를 제공한다.

[표 4-2] 주요 실증 연구(RCT) 결과 비교

연구자	연구방법	참여자 수	AI 도구	주요 결과	비고
Peng et al. (2023)	RCT	n=95	Copilot	Copilot 사용 그룹이 동일 과제를 더 빠르게 완료	55.8% 속도 향상
Paradis et al. (2024)	RCT	n=96	Google AI 기능	복잡한 기업 과제에서는 AI 도구의 영향 제한적	맥락 의존적
Becker et al. (2025)	RCT	n=16	Cursor+Claude	경험 많은 개발자에게는 역설적으로 작업 시간 증가	경험 수준 따라 상이
Butler et al. (2024)	RCT + 일지 연구	n=200+	Copilot(MS)	84% 일상업무 개선 보고; 66% 심리 변화 경험	높음 (실무환경)

주: Peng et al.(2023), Butler et al.(2024), Paradis et al.(2024), Becker et al.(2025)를 바탕으로 연구자가 작성함.

위 RCT 실증 연구는 AI 도구의 효과가 맥락에 매우 의존적임을 보여준다. Peng et al.(2023)이 보고한 55.8% 속도 향상은 통제된 실험실 환경에서의 결과이며, Paradis et al.(2024)의 연구는 실제 기업 환경의 복잡한 과제에서는 효과가 제한적일 수 있음을 시사한다. 특히 Becker et al.(2025)의 역설적 발견은 경험 수준에 따른 효과의 이질성을 강조한다. 반면 Butler et al.(2024)의 Microsoft Copilot을 이용한 연구에서 200명 이상의 현업 개발자를 대상으로 84%가 일상 업무 개선을 경험했다는 결과는 대규모 현장 적용의 가능성을 시사한다.

AI 생성 코드의 장점에도 불구하고, 보안 및 품질 측면의 우려가 제기되고 있다. Perry et al.(2023)의 Stanford 연구에서는 AI 어시스턴트를 사용하는 개발자가 통계적으로 더 많은 보안 취약점 코드를 작성하는 경향을 보였으며, 이는 개발자의 과신 때문일 수 있다고 분석했다. Klemmer et al.(2024)의 연구도 개발자들이 AI 보조 코드의 보안 검증에 대해 우려를 갖고 있으며, 책임 귀속(responsibility attribution)의 모호함이 문제가 될 수 있음을 지적했다. Borg et al.(2024)의 연구는 AI 어시스턴트와의 공동 개발이 단기적인 생산성 향상은 가

져올 수 있으나, 장기적인 코드 유지보수성과 가독성에 미치는 영향은 아직 명확하지 않음을 보였다. 따라서 AI 생성 코드의 도입 시 강화된 코드 리뷰 프로세스, 보안 정적 분석 도구 의무 적용, 개발자 교육 및 가이드라인 수립, AI 생성 코드의 출처 및 라이선스 추적 등 기준이 필요하다.

4.1.1.4 소프트웨어 테스트 단계

테스트 단계는 소프트웨어의 결함을 발견하고 품질을 보증하는 과정으로, 단위 테스트(Unit Test), 통합 테스트(Integration Test), 시스템 테스트(System Test), 인수 테스트(Acceptance Test) 등으로 구분할 수 있다. 전통적으로 테스트는 많은 시간과 인력이 요구되는 단계였으나, AI 기술의 적용으로 자동화 범위가 크게 확대되고 있다.

Gartner(2024)의 보고서에 따르면 3년 내 기업의 80%가 AI 증강 테스트 도구를 소프트웨어 엔지니어링 툴체인에 통합할 것으로 전망되며, 이는 2023년 기준 15%에서 급격히 증가한 수치이다. CloudQA(2025)의 보고서에 따르면 LLM은 요구사항을 이해하여 테스트 케이스를 자동으로 생성하고, 테스트 데이터 생성, 결과 분석까지 자동화한다.

소프트웨어 테스트 분야의 AI 도구는 많이 알려져 있다. TestGen-LLM은 Meta에서 개발한 LLM 기반 단위 테스트 자동 개선 시스템으로, Instagram·Facebook 테스트 대상에 적용 시 75%의 테스트가 정상 빌드되고, 57%가 안정적으로 통과하였으며, 적용 클래스의 11.5%에서 테스트 커버리지가 향상되었다. 73%의 권장 사항이 프로덕션에 채택되었다(Schäfer et al., 2024). Cover-Agent(CodiumAI)는 코드 커버리지를 자동으로 분석하고 미커버된 코드 경로에 대한 테스트 케이스를 자동 생성하는 오픈소스 AI 도구이다. Diffblue Cover는 자바 코드를 위한 AI 기반 단위 테스트 자동 생성 도구로, 기존 코드를 분석하여 최대 80%의 코드 커버리지를 달성하는 테스트를 자동으로 작성한다. Selenium + LLM을 조합하여 UI 테스트 스크립트를 자연어로 작성하면 LLM이 Selenium 코드로 변환하는 방식으로, 비 기술 테스터도 자동화 테스트를 작성할 수 있게 한다.

AI Copilot for Testing은 Context-Based RAG를 활용하여 코드베이스 업데

이트와 버그 탐지를 동기화하는 자동화 테스트 시스템으로, 코드 생성에서 소프트웨어 테스팅까지 통합 파이프라인을 제공한다(Wang et al., 2025). Wang et al.(2025) 연구의 Context-Based RAG 접근법은 기존 코드 생성 AI의 한계를 극복하기 위해 제안된 것으로, 소스코드 저장소에 대한 동적 컨텍스트를 활용하여 테스트 케이스 생성의 정확도를 크게 향상시킨다. 한편 Dong et al.(2025)의 LLM 기반 코드 생성 에이전트에 관한 설문 연구에서는 AI 에이전트가 단순 코드 생성을 넘어 멀티스텝 테스트 작업(테스트 케이스 설계, 테스트 데이터 생성, 결과 검증)까지 자동으로 수행할 수 있음을 보였다. 특히 AI 에이전트는 개발 단계의 버그를 테스트 단계에서 발견하고 피드백하는 폐쇄 루프를 형성할 수 있어, CI/CD 파이프라인의 자동화 수준을 크게 높일 수 있다.

4.1.1.5 소프트웨어 배포 및 유지보수 단계

소프트웨어 배포 및 유지보수 단계는 소프트웨어를 운영 서버 등 환경에 릴리스하고 지속적으로 관리하는 과정이다. DevOps 및 CI/CD 파이프라인의 확산으로 배포 주기가 단축되면서, AI를 활용한 지능형 모니터링과 자동화된 운영 관리의 중요성이 높아지고 있다.

AIOps(AI for IT Operations)는 ML 알고리즘을 활용하여 시스템 로그, 메트릭, 이벤트를 자동으로 분석하고 장애를 예측하거나 자동으로 복구하는 접근법이다. CircleCI(2024)의 연구에 따르면 AI 기반 SDLC에서 배포 및 유지보수 단계의 실세계 인사이트가 실시간으로 계획·설계·테스트 단계에 피드백되어 개발 사이클 전체가 지속적으로 개선되는 구조가 구현되고 있다.

소프트웨어는 개발 후 계속해서 수정하고 배포하는 작업을 반복한다. 이때 AI 기반 CI/CD 업무를 최적화할 수 있다. GitHub Actions와 AI 분석을 통해 빌드 실패 패턴을 학습하고 파이프라인 실행 시간을 최적화한다. AI가 코드 변경 영향도를 분석하여 필요한 테스트만 선택적으로 실행하는 스마트 테스트 선택을 지원할 수 있다. 유지보수 운영 단계에서는 Datadog AI, Dynatrace Davis AI 등이 로그 데이터와 시스템 메트릭을 AI로 분석하여 장애를 사전에 예측하고 근본 원인을 자동으로 분석한다. 즉, AI 기반 버그 예측이 가능하다. 소스코드 변경 이력과 결함 데이터를 학습하여 높은 결함 위험도를 가진 코드 영역을 사전

에 식별하고 테스트 집중 영역을 안내한다. 유지보수 단계에서 발주처 또는 협업 담당부서에 제공할 문서도 자동화할 수 있다. AI가 코드 변경 사항을 분석하여 릴리스 노트, API 문서, 변경 이력을 자동으로 생성하여 유지보수 문서화 부담을 줄인다.

4.1.2 SDLC AI 도입 효과 및 장애 요인

4.1.2.1 통합적 효과 분석

소프트웨어 개발 전 단계에 AI 기술을 이용하여 시스템을 통합했을 때의 효과를 종합하면 다음과 같다. EPAM Systems(2024)은 2028년까지 SDLC 전반에 AI를 통합한 팀이 코드 생성 도구만 사용하는 팀(약 10% 향상)보다 높은 25~30%의 생산성 향상을 달성할 것으로 전망하였다. 이는 AI 도구를 특정 단계에 국한하여 사용하는 것보다 전체 파이프라인에 통합했을 때 시너지 효과가 발생함을 의미한다.

[표 4-3] 소프트웨어 개발 단계별 AI 기술 활용 효과 종합

단계	주요 AI 도구	측정된 효과	데이터 출처
요구사항 분석	LLM(ChatGPT, Claude)	분석시간 약 40-60% 단축	Guimaraes & Nascimento(2025)
설계	AI 아키텍처 추천	설계 오류 약 30% 감소	AWS (2024)
구현	GitHub Copilot	코딩 속도 약 55% 향상	Peng et al.(2023)
테스트	TestGen-LLM	커버리지 약 25% 향상	Alshahwan et al.(2024)
배포·유지보수	AIOps 도구	MTTR 약 50% 단축	Gartner (2024)

주: 각 출처의 공개 자료를 바탕으로 연구자가 종합

4.1.2.2 AI 도입 장애 요인

AI 기술의 실질적 도입에는 다양한 장애 요인이 존재한다. 이를 조직적, 기술적, 인적 측면으로 구분하여 살펴볼 수 있다. 조직적 장애는 AI 기술을 이용한 지적재산권 관련 이슈가 주를 이룬다. 소스코드 또는 특허 기술 등을 AI 서비스를 제공하는 회사가 이용할 것이 우려되는 것이다. 기술적 장애 요인은 여전히 발생하고 있는 환각 현상이다. AI는 어떠한 질문에도 답변을 생성하는 구조이므로, 부정확한 답변을 개발자가 그대로 사용할 경우 오류가 발생할 수 있다는 한계와 취약점이 존재한다. 인적 장애 요인은 개발자의 역량과 관련된 부문이다.

[표 4-4] AI 장애 요인 및 내용

구분	내용	비고
조직적 장애 요인	- AI 도구 도입에 따른 보안 및 IP 유출 우려 - AI 도구 라이선스 비용 및 ROI 불확실성 - 경영진의 AI 도입에 대한 의사결정 지연 - 기존 개발 프로세스와의 통합 복잡성	
기술적 장애 요인	- AI의 환각(Hallucination) 현상으로 인한 코드 오류 위험 - 기존 레거시 시스템과의 통합 어려움 - 프라이빗 코드베이스에 대한 AI 컨텍스트 이해 한계 - AI 생성 코드의 보안 취약점 가능성	
인적 장애 요인	- 개발자의 AI 도구 학습 및 활용 역량 부족 - AI 도구에 대한 과도한 의존 및 비판적 사고 약화 - 일부 개발자의 AI 도입에 대한 심리적 저항 - AI 활용에 관한 팀 내 역량 격차 확대	

자료: 본 표는 생성형 AI로 초안을 생성하고 연구자가 검토·수정하여 완성함.

4.1.2.3 장애 요인 극복 방안

AI 도입과 관련된 장애 극복은 탐색, 확산, 통합 등 단계적으로 진행해야 한다. 첫 번째는 단계적 접근이다. 전사적 전면 도입보다는 파일럿 프로젝트를 수행하여 검증 후 단계적 확산하는 것이 필요하다. 두 번째는 AI 리터러시 교육 강화이다. 개발자 대상 AI 도구 활용 교육 및 프롬프트 엔지니어링 훈련을 체계화하면 더 효과적으로 AI 도구를 사용할 수 있다. 세 번째는 코드 리뷰 프로세스 강화이다. AI 생성 코드에 대한 필수 코드 리뷰 정책 수립 및 품질 검증 자

동화 정책을 수립하는 품질관리 절차 정의가 필요하다. 네 번째는 보안 가이드라인 수립이다. AI 도구 사용 시 기밀 코드 보호를 위한 사내 보안 정책 및 가이드라인 정비가 필요하다. 다섯 번째는 성과 측정 체계 구축이다. AI 도입 ROI 효과를 정량적으로 측정하기 위한 KPI 및 측정 방법론을 수립하고 이를 지속적으로 모니터링하고 관리하면 개선 또는 개선되지 않는 단계를 파악하고 대안을 검토할 수 있게 된다.

[표 4-5] AI 도입 장애 요인 극복 방안

단계	도입 기간	주요 활동	성과 지표
1단계 탐색	0~3개월	AI 코딩 도구 파일럿(GitHub Copilot 등) 도입; 개발자 AI 리터러시 교육; AI 거버넌스 정책 수립	파일럿 참여자 생산성 변화율; 교육 이수율
2단계 확산	3~12개월	구현 단계 AI 도구 전사 확산; AI 테스트 자동화 도입; AI 기반 코드 리뷰 체계 구축	전사 코딩 속도 향상률; 테스트 커버리지 변화; 결함 발견율
3단계 통합	12~24개월	SDLC 전 단계 AI 통합; 요구사항 분석·설계 단계 AI 지원; AIOps 기반 운영 자동화	전체 개발 사이클 타임 단축; AI 활용 ROI; 개발자 만족도
4단계 AI-Native	24개월 이후	AI-Native SDLC 방법론 전환; AI 에이전트 기반 자율 개발 파이프라인 구현; 지속적 AI 개선 문화 정착	AI 기여 코드 비율; 자율화 수준; 조직 AI 성숙도 지수

자료: 본 표는 생성형 AI로 초안을 생성하고 연구자가 검토·수정하여 완성함.

4.2 SDLC 단계별 AI 이용 설문조사

4.2.1 표본의 선정 및 분석 방법

4.2.1.1 표본의 선정 및 자료 수집

이 연구는 소프트웨어 개발 단계별 AI 기술 활용 수준 및 생산성 향상 효과를 분석하기 위해, 실제 산업 현장에서 AI 도구를 활용해 본 경험이 있는 '소프트웨어 개발자'만을 대상으로 한정하여 표본을 선정하였다. 표본 추출 과정에서는 연구의 목적에 부합하지 않는 비기술 직군(일반 영업, 인사, 총무, 재무, 비기

술 마케팅, 단순 관리 등)의 응답이 통계에 혼입되는 것을 원천적으로 차단하기 위해, 설문 도입부에 소프트웨어 개발 생명주기(SDLC) 직접 참여 여부를 묻는 핵심 필터(Filter) 질문을 배치하였다. 본 설문은 2026년 상반기에 약 2주간 전문 리서치 기관에 의뢰하여, 해당 기관이 보유한 IT·소프트웨어 개발자 패널을 대상으로 온라인으로 실시하였다. 설문 도입부의 핵심 필터 질문을 통과한 SDLC 직접 참여 직군만을 유효 표본으로 집계하여 총 200부를 확보하였다. 수집된 자료는 연구 목적(소프트웨어 개발 단계별 인공지능 기술 활용 방안 연구)에 따른 통계분석용으로만 활용하였다.

4.2.1.2 측정 도구의 구성

이 연구의 측정 도구로 사용된 설문지는 문헌 고찰 및 선행 연구를 바탕으로 연구자가 직접 구성하였다. 전체 설문은 총 4개의 파트로 구성하였다. 첫 번째 파트(1)는 AI 기술 도입 및 일반 활용 현황 (사용 빈도, 주요 도구, 숙련도, 도입 계기 등) 등을 조사하였다. 두 번째 파트(2)는 조사의 정확도를 높이기 위해 조사 전 각 단계에 대한 경험의 유·무를 확인하였으며, 경험한 단계(모든 단계)에 대해서만 SDLC 5단계(요구사항 분석, 설계, 구현/코딩, 테스트 및 검증, 배포 및 유지보수) 별 AI 도구 활용 현황 및 정량적 효과 등을 조사하였다. 세 번째 파트(3)는 AI 도입의 장애 요인 및 직무역량 인식을 조사하였으며, 네 번째 파트(4)는 응답자의 인구통계학적 기본 정보 (경력, 조직 규모, 주요 직무)를 조사하였다. 특히 본 설문은 연구의 통계적 중립성을 확보할 수 있도록 편향된 설계를 배제하고자 노력하였다. 또한 'AI가 무조건적으로 생산성을 향상시킨다'는 오류를 방지하기 위해 정량적 성과 측정 문항에 '단축 효과 없음 또는 오히려 증가(0% 이하)'와 같은 보기 항목을 포함하여 효과율을 5단계 척도(0%이하, 20% 미만, 20~40%, 40~60%, 60% 이상)로 세분화하여 구성하였다. 또한, 각 SDLC 단계에서의 단계별 활용 수준 및 생산성 향상도는 1점부터 10점까지의 10점 만점 척도를 적용하여 응답의 분산 및 정밀도를 높였다.

4.2.1.3 자료 분석 방법

자료 분석 방법은 빈도분석, 다중 응답 분석, 기술 통계 분석 3가지 방법을

이용하였다. 빈도분석(Frequency Analysis)은 조사 대상자의 인구통계학적 특성, 소속 조직의 규모, 업무 경력 및 기본적인 AI 도구 사용 빈도 등을 파악하기 위해 실시하였다. 다중 응답 분석(Multiple Response Analysis)은 SDLC 각 단계에서 응답자가 중복으로 선택한 '주요 활용 도구' 및 '주요 효과'의 분포와 우선순위를 도출하기 위해 수행하였다. 기술 통계 분석(Descriptive Statistics)은 단계별 AI 기술 활용 수준 및 생산성 향상도(1~10점 척도)의 평균, 표준편차 등을 산출하여 전반적인 경향성을 정량적으로 확인하였다.

4.2.2 설문조사 결과 분석

4.2.2.1 표본의 일반적 특성

다음의 [표 4-6]은 설문조사에 참여한 표본(총 200명)의 인구통계학적 및 직무적 특성을 분석한 결과이다. 응답자의 소프트웨어 개발 관련 총 업무 경력을 비중별로 살펴보면, '중급(3년 이상 – 7년 미만)'이 91명(45.5%)으로 가장 높은 비중을 차지하였다. 이어서 '고급(7년 이상 – 12년 미만)' 58명(29.0%), '특급(12년 이상 – 18년 미만)' 24명(12.0%), '초급(3년 미만)' 15명(7.5%), '관리자/전문가 수준(18년 이상)' 12명(6.0%) 순으로 나타났다. 이는 이 연구의 표본이 SDLC의 여러 단계의 현장 실무 경험이 충분한 중·고급 이상의 개발자 중심으로 구성되어, AI 도입 효과에 대한 평가의 신뢰성이 높음을 보여주고 있다.

소속 조직의 규모는 '100인 – 500인'이 76명(38.0%)으로 가장 많았고, '10인 – 100인' 56명(28.0%), '500인 – 1,000인' 38명(19.0%), '1,000인 이상' 25명(12.5%), '10인 미만' 5명(2.5%) 순으로 조사되어, 중견 및 중소 규모 기업부터 대기업까지 다양한 조직 및 규모의 개발 환경의 응답자가 포함되어 환경에 따른 편향이 없도록 표본이 구성되어 있다. 주요 직무 분포에서는 '시스템 설계 및 리딩'이 60명(30.0%)으로 가장 높게 나타났으며, '어플리케이션 개발'과 '품질 관리 및 테스트'가 각각 45명(22.5%)으로 동일한 비중을 차지하였다. 다음으로 '프로젝트 관리 및 기획' 40명(20.0%), '인프라 및 운영' 10명(5.0%) 순으로 집계되었다. 이는 SDLC 전 단계(분석-설계-구현-테스트-배포/운영)의 실무 담당자들이 고르게 분포되어 있어, 특정 개발 단계에 편중되지 않은 객관적인 분석이

가능함을 시사한다. 다만 이 직무 분포는 현재의 담당 업무에 대한 직무 분포이며, 경험한 모든 직무 결과를 의미하지는 않는다.

[표 4-6] 표본의 일반적 특성

		빈도	백분율
전체		200	100.0
총 업무 경력 (IT 분야)	초급(3년 미만)	15	7.5%
	중급(3년 이상 ~ 7년 미만)	91	45.5%
	고급(7년 이상 ~ 12년 미만)	58	29.0%
	특급(12년 이상 ~ 18년 미만)	24	12.0%
	관리자/전문가 수준(18년 이상)	12	6.0%
소속 조직	10인 미만	5	2.5%
	10인 ~100인	56	28.0%
	100인 ~ 500인	76	38.0%
	500인 ~ 1,000인	38	19.0%
	1,000인 이상	25	12.5%
주요 직무	프로젝트 관리 및 기획	40	20.0%
	시스템 설계 및 리딩	60	30.0%
	어플리케이션 개발	45	22.5%
	품질 관리 및 테스트	45	22.5%
	인프라 및 운영	10	5.0%

4.2.2.2 AI 기술 도입 및 일반 활용 현황

다음의 [표 4-7]은 소프트웨어 개발 생명주기(SDLC) 단계별 심층 분석에 앞서, 응답자 전반의 AI 기술 도입 및 일반적 활용 현황을 파악하기 위해 조사한 결과이다. 먼저 실무에서의 AI 도구 사용 빈도를 살펴보면, '매일' 사용한다는 응답이 77명(38.5%)으로 가장 많았고, '주 3-4회' 73명(36.5%), '주 1-2회' 46명(23.0%), '월 1-2회' 4명(2.0%) 순으로 나타났다. 반면 '거의 사용안함'은 0명(0.0%)으로 조사되어, 전체 응답자가 현업에서 정기적으로 AI 도구를 활용하고 있음이 확인되었다. 업무 전반에 걸쳐 사용하는 주요 AI 도구에 대한 복수 응답 결과(총 342건), '범용 대화형 LLM'이 120건(35.1%)으로 가장 높은 비중을 차지하였다. 이어서 'IDE 확장형 코딩 어시스턴트' 84건(24.6%), 'AI 네이티브 코드 에디터' 53건(15.5%), '분석 및 품질/보안 특화 도구' 62건(18.1%), '자율형 에이전트 및 자동화 도구' 23건(6.7%) 순으로 나타났다. 이는 개발자들이 단일 도구에 국한되지 않고 업무 목적과 태스크의 성격에 맞추어 다양한 AI 도구를 교차 활용하고 있음을 보여준다. 응답자 스스로 평가한 AI 도구 활용 숙련도는 '중급'이 103명(51.5%)으로 과반을 차지했으며, '상급' 44명(22.0%), '기초' 38명(19.0%), '전문가' 9명(4.5%), '입문' 6명(3.0%) 순으로 집계되었다. 상기의 대답은 응답자가 스스로 중급 또는 상급으로 분류하고 있으나 이에 대한 객관적인 기준이 없는 각자의 주관적인 판단이다. 소속 조직의 AI 도구 사용 환경을 묻는 질문에는 '유료 계정 지원'이 98명(49.0%)으로 가장 높게 나타났다.

그 외 '제한적 허용' 40명(20.0%), '전용 인프라 구축' 37명(18.5%), '강력한 권장/필수' 13명(6.5%), '개인적 활용' 12명(6.0%) 순으로 조사되어, 이미 다수의 기업이 적극적으로 비용 및 인프라 제공을 통해 AI 활용을 장려하고 있음이 확인된다. 마지막으로 AI 도구를 실무에 도입하게 된 주된 동기 및 목적에 대해서는 '업무 과부하 해결'이 79명(39.5%), '기술적 한계 극복'이 70명(35.0%)으로 주축을 이루었다. 반면 '조직의 권고/지시'는 36명(18.0%), '동료/업계 트렌드' 10명(5.0%), '개인적 호기심' 5명(2.5%)에 그쳤다. AI 도구 사용환경과 동기 및 목적에 대한 조사를 보면 AI 기술의 수용이 조직의 권고와 인프라 제공이라는 환경도 있지만 업무 효율성 극대화 및 기술적 한계 극복이라는 개인적·실용적인 목적하에 이뤄지고 있음을 보여준다.

[표 4-7] AI 기술 도입 및 일반 활용 현황

		빈도	백분율
전체		200	100.0
AI 도구 사용 빈도	매일	77	38.5%
	주 3-4회	73	36.5%
	주 1-2회	46	23.0%
	월1-2회	4	2.0%
	거의 사용안함	0	0.0%
업무 전반 사용 AI 도구 (복수선택) 합 342	범용 대화형 LLM	120	35.1%
	IDE 확장형 코딩 어시스턴트	84	24.6%
	AI 네이티브 코드 에디터	53	15.5%
	분석 및 품질/보안 특화 도구	62	18.1%
	자율형 에이전트 및 자동화 도구	23	6.7%
AI 도구 활용 숙련도	입문	6	3.0%
	기초	38	19.0%
	중급	103	51.5%
	상급	44	22.0%
	전문가	9	4.5%
조직 내 AI 도구 사용 환경	개인적 활용	12	6.0%
	제한적 허용	40	20.0%
	유료 계정 지원	98	49.0%
	전용 인프라 구축	37	18.5%
	강력한 권장/필수	13	6.5%
AI 도구 도입 동기/목적	개인적 호기심	5	2.5%
	업무 과부하 해결	79	39.5%
	기술적 한계 극복	70	35.0%
	조직의 권고/지시	36	18.0%
	동료/업계 트렌드	10	5.0%

HANSUNG UNIVERSITY

4.2.2.3 SDLC 단계별 AI 도구 활용

1) 소프트웨어 요구사항 분석 단계

[표 4-8]은 SDLC의 첫 단계인 요구사항 분석 단계에서 현장 실무자들이 AI를 어떻게 활용하고 있으며, 그로 인해 어떤 실질적 효익을 얻고 있는지를 분석한 결과이다. 요구사항 분석은 고객의 모호한 요구를 구체적인 시스템 규격으로 번역하는 고도의 인지적 작업이다. 따라서 특정 도구를 이용한 단순한 물리적 시간 단축 여부만으로는 AI의 진정한 가치를 측정할 수 없다고 판단하여, 본 연구에서는 다중 응답 방식을 통해 실무적 효용을 입체적으로 추적했다.

분석 결과에서 가장 주목할 만한 점은 AI의 주된 도입 효과가 단순한 '속도전'을 넘어 '품질 검증'과 '표준화'에 집중되어 있다는 사실이다. 총 111건의 응답 중 ‘요구사항 간 논리적 모순 및 누락 식별’과 ‘표준화된 산출물(문서) 자동 생성’이 각각 54건(27.0%)으로 가장 높은 비중을 차지했다.

이는 프로젝트 현장에서 가장 치명적인 리스크인 '요구사항 누락으로 인한 후반부 대규모 재작업(Rework)'을 개발자들이 AI를 통해 선제적으로 방어하고 있음을 의미한다. 즉, 현장에서는 AI를 단순한 초안 작성기(25.5%)를 넘어, 인간의 실수를 교차 검증하고 산출물의 일관성을 강제하는 '보조 분석가'로 적극 활용하고 있다는 것이 연구자의 해석이다. 반면, ‘이해관계자 간 의사결정 및 합의 지원’은 단 7건(3.5%)에 불과했다. 이는 현재 기술 수준에서 AI가 데이터 구조화나 문서 양식 통일 등 기계적이고 정형화된 작업에는 탁월한 성능을 보이나, 발주처와 수행사 간의 미묘한 입장차이를 조율하거나 전략적 합의를 이끌어 내는 '인간 고유의 정치적·정서적 소통 영역'에는 개입하지 못하고 있다는 명확한 한계를 보여준다.

[표 4-8] (SDLC 1단계) 요구사항 분석 단계 조사

		빈도	백분율
전체		111	100.0
주요 활용 도구	범용 대화형 LLM (ChatGPT, Claude, Gemini 등)	46	41.4%
	회의/음성 분석 AI (클로바노트, Otter.ai, MS Copilot 등)	19	17.1%
	문서 및 이슈 관리 내장 AI (Notion AI, Atlassian Intelligence(Jira/Confluence) 등)	40	36.0%
	다이어그램 및 시각화 AI (Eraser.io, Mermaid AI, Whimsical 등)	6	5.4%
	기타	0	0.0%
	AI 도구를 활용하지 않음	0	0.0%
주요 효과 (복수선택: 최대2개) 합: 200	요구사항 구체화 및 초안 작성 시간 단축	51	25.5%
	요구사항 간 논리적 모순 및 누락 식별	54	27.0%
	표준화된 산출물(문서) 자동 생성	54	27.0%
	비즈니스 연계성 확보(모호한 요구사항을 데이터 모델로 변환)	34	17.0%
	이해관계자 간 의사결정 및 합의 지원	7	3.5%
정량적 효과	단축 효과 없음	0	0.0%
	20% 미만	23	20.7%
	20% ~ 40%	63	56.8%
	40% ~ 60%	17	15.3%
	60% 이상	8	7.2%

2) 소프트웨어 시스템 설계 단계

[표 4-9]는 SDLC의 두 번째 단계인 시스템 설계 단계에서 AI 기술이 추상적인 요구사항을 시스템 아키텍처와 데이터베이스 스키마로 구체화하는데 어떻게 개입하고 있는지 분석한 결과이다. 설계 단계는 단 한 번의 구조적 결함이 향후 구현 및 테스트 단계에서 막대한 기술 부채(Technical Debt)로 전이되는 핵심 구간이므로, 현장에서 AI 도구를 어떤 목적으로 활용하는지는 매우 중요한 시사점을 제공한다.

조사 결과, 가장 많이 활용된 도구는 'AI 기반 다이어그램/시각화 도구(35.2%)'와 '클라우드 및 인프라 설계 특화 AI(29.5%)'로 나타났다. 이는 현장 실무자들이 복잡한 비즈니스 로직과 컴포넌트 간의 상호작용을 시각적으로 구조화하고, 클라우드 네이티브 환경에 맞는 최적의 아키텍처를 구성하는 데 AI의 패턴 인식 능력을 적극 차용하고 있음을 의미한다.

AI 도입으로 인한 주요 효과 역시 단순한 설계 도면 산출을 넘어 '설계의 시각적 명확성'과 '표준화'에 방점이 찍혀 있다. 응답자 다수가 '다이어그램 등 시각적 구조화 자동화(예: 33.6%)'와 '설계 표준화 및 일관성 유지(예: 32.0%)'를 핵심 성과로 지목했다. 프로젝트 수행 과정에서 가장 빈번하게 발생하는 개발자 간의 인터페이스 규격 오해나 데이터 타입 불일치 등의 휴먼에러를 AI가 강제적으로 통일시켜, 팀 내 기술적 의사소통 비용을 획기적으로 낮추고 있다는 것이 연구자의 분석이다. 반면, 복잡한 '구조적 결함 및 보안 병목 사전 식별(예: 2.5%)'과 같은 심층적인 보안 아키텍처 검증 영역에서의 효용성은 아직 수치가 미미하다. 이는 고도의 도메인 지식과 얽혀있는 아키텍처 결함을 AI가 독자적으로 판단하기에는 현재 기술적 한계가 있음을 이야기한다.

정량적 시간 단축 지표는 설계 단계에서 AI의 실무적 개입이 얼마나 성공적으로 안착했는지를 통계적으로 보여준다. 전체 응답자의 48.4%(59명)가 20%~40%의 시간 단축 효과를 경험했으며, 40% 이상의 고효율을 경험한 비율도 27.1%(33명)에 달했다. 특히 주목할 점은 '단축 효과가 없다'는 응답이 단 1명(0.8%)에 불과하다는 사실이다. 이는 인간의 고도의 창의성과 직관이 지배한다고 여겨졌던 시스템 아키텍처 설계 영역조차도, AI의 패턴 추천과 보일러 플레이트 생성 기능이 필수적인 생산성 도구로 자리매김했음을 보여준다.

[표 4-9] (SDLC 2단계) 시스템 설계 단계 조사

		빈도	백분율
	전체	122	100.0
주요 활용 도구	범용 대화형 LLM (ChatGPT, Claude, Gemini 등)	31	25.4%
	AI 기반 다이어그램/시각화 도구 (Eraser.io, Mermaid AI, Draw.io AI 등)	43	35.2%
	클라우드 및 인프라 설계 특화 AI (Amazon Q, Google Cloud Gemini, 인프라스트럭처 애즈 코드(IaC) 보조 AI 등)	36	29.5%
	데이터베이스/ERD 특화 AI (AI 지원 ERD 툴, SQLDBM, DataGrip AI 등)	12	9.8%
	기타	0	0.0%
	AI 도구를 활용하지 않음	0	0.0%
주요 효과 (단일응답)	초기 설계 및 산출물 작성 시간 단축	18	14.8%
	최적의 아키텍처 및 기술 대안 비교	41	33.6%
	다이어그램 등 시각적 구조화 자동화	39	32.0%
	구조적 결함 및 보안 병목 사전 식별	21	17.2%
	설계 표준화 및 일관성 유지	3	2.5%
정량적 효과	단축 효과 없음	1	0.8%
	20% 미만	29	23.8%
	20% ~ 40%	59	48.4%
	40% ~ 60%	24	19.7%
	60% 이상	9	7.4%

3) 소프트웨어 구현 단계

[표 4-10]은 SDLC의 핵심이자 물리적 자원이 가장 많이 투입되는 구현 및 코딩 단계에서 현장 개발자들이 AI를 어떻게 활용하고 있는지 분석한 결과이다. 흔히 코딩 단계에서 AI의 역할이 단순한 코드 자동완성에 머물 것이라 예상하지만, 본 조사 결과는 현장의 활용 수준이 이미 그 단계를 훌쩍 넘어섰음을 보여준다.

[표 4-10] (SDLC 3단계) 구현/코딩 단계 조사

		빈도	백분율
전체		131	100.0
주요 활용 도구	범용 대화형 LLM (ChatGPT, Claude, Gemini 등)	21	16.0%
	IDE 확장 플러그인 (GitHub Copilot, JetBrains AI, Amazon Q 등)	37	28.2%
	AI 네이티브 에디터 (Cursor, Windsurf 등)	50	38.2%
	특화 생성기 및 자율형 에이전트 (v0.dev, bolt.new, Aider, Devin 등)	23	17.6%
	기타	0	0.0%
	AI 도구를 활용하지 않음	0	0.0%
주요 효과 (단일응답)	상용구(Boilerplate) 및 반복 코드 자동 생성	13	9.9%
	복잡한 로직 및 알고리즘 구현 보조	39	29.8%
	레거시 코드 해석 및 주석 자동화	37	28.2%
	코드 가독성 향상 및 리팩터링	34	26.0%
	신규 기술 및 프레임워크 학습 곡선 (Learning Curve) 단축	8	6.1%
정량적 효과	단축 효과 없음	0	0.0%
	20% 미만	12	9.2%
	20% ~ 40%	67	51.1%
	40% ~ 60%	37	28.2%
	60% 이상	15	11.5%

조사 결과, 범용 대화형 LLM(16.0%)보다 'AI 네이티브 에디터(38.2%)'와 'IDE 확장 플러그인(28.2%)'의 선호도가 압도적으로 높게 나타났다. 이는 실무자들이 웹브라우저를 오가는 번거로움을 기피하고, 프로젝트 전체의 맥락(Context)을 이해하며 개발 환경 내에서 즉각적으로 상호작용하는 통합된 전용 도구를 필수재로 인식하고 있음을 의미한다. 주요 효과 부문에서도 이러한 심화

된 활용 양상이 여실히 드러난다. 단순한 ‘상용구(Boilerplate) 및 반복 코드 자동 생성(9.9%)’보다 ‘복잡한 로직 및 알고리즘 구현 보조(29.8%)’, ‘레거시 코드 해석 및 주석 자동화(28.2%)’, ‘코드 가독성 향상 및 리팩터링(26.0%)’에 응답이 집중되었다. 즉, 현장 개발자들은 AI를 단순 타이핑을 대신하는 '기계적 타자수'가 아니라, 얽혀있는 기존 코드의 의도를 파악하고 난도 높은 로직을 함께 고민하는 '페어 프로그래머(Pair Programmer)'이자 '코드 리뷰어'로 활용하여 인지적 과부하를 방어하고 있다는 것이 연구자의 해석이다.

생산성 향상 효과를 살펴보면 응답자의 51.1%가 20%~40%의 단축 효과를, 28.2%가 40%~60%의 단축 효과를 경험했다고 답했다. '단축 효과 없음'이 0%라는 사실은 코딩 단계에서의 AI 도입이 더 이상 선택의 영역이 아니라, 개발 리드타임 단축과 경쟁력 확보를 위한 상수(Constant)로 자리 잡았음을 객관적으로 보여준다.

4) 소프트웨어 테스트 및 검증 단계

[표 4-11]은 시스템의 무결성을 담보하는 테스트 및 검증 단계에서 AI 기술이 어떻게 품질 보증(QA) 패러다임을 전환하고 있는지를 짚어낸 결과이다. 과거 테스트 단계는 막대한 인력이 투입되는 반복적이고 소모적인 구간으로 인식되었으나, 최근 AI의 개입으로 가장 혁신적인 지능화가 이루어지고 있는 영역이기도 하다.

활용 도구의 분포를 보면 ‘AI 기반 정적 분석 및 버그 탐지 도구(29.9%)’와 ‘유닛 테스트 자동 생성 도구(28.7%)’가 근소한 차이로 상위권을 형성했다. 이는 코드 병합(PR) 과정에서 보안 취약점과 런타임 버그를 사전에 차단하고, 수동으로 작성하기 번거로운 단위 테스트를 자동화하여 품질 검증의 속도와 정확성을 동시에 끌어올리려는 실무진의 강한 의지가 반영된 결과다.

특히 체감하는 도입 효과에 대한 데이터는 매우 흥미로운 시사점을 제공한다. 테스트 커버리지 확대(6.9%)나 스크립트 작성 시간 단축(11.5%)과 같은 1차원적인 생산성 지표보다, ‘테스트 시나리오의 다양성 및 현실성 확보(37.9%)’와 ‘결함 탐지 정확도 향상 및 근본 원인 분석(RCA) 지원(31.0%)’에 응답이 집중되었다. 이는 AI가 사람이 미처 생각하지 못한 다양한 사용자 엣지 케이스(Edge

Case)를 모사하여 테스트의 질적 수준을 높일 뿐 아니라, 단순한 버그 발견을 넘어 원인 추적과 해결책 까지 제시하는 '지능형 QA 분석가'로 기능하고 있음을 보여준다.

[표 4-11] (SDLC 4단계) 테스트 및 검증 단계 조사

		빈도	백분율
전체		87	100.0
주요 활용 도구	범용 대화형 LLM (ChatGPT, Claude, Gemini 등)	15	17.2%
	유닛 테스트 자동 생성 도구 (Diffblue Cover, TestGen-LLM, GitHub Copilot 등)	25	28.7%
	자율형 E2E 및 QA 에이전트 (mabl, Sauce Labs AI, Blinq.io 등)	21	24.1%
	AI 기반 정적 분석 및 버그 탐지 도구 (CodeRabbit, Qodo Merge, Snyk, SonarQube AI 등)	26	29.9%
	기타	0	0.0%
	AI 도구를 활용하지 않음	0	0.0%
주요 효과 (단일응답)	테스트 커버리지 확대 및 엣지 케이스 발견	6	6.9%
	테스트 스크립트 작성 및 유지보수 시간 단축	10	11.5%
	결함 탐지 정확도 향상 및 근본 원인 분석(RCA) 지원	27	31.0%
	테스트 시나리오의 다양성 및 현실성 확보	33	37.9%
	개발-QA 협업 효율성 및 배포 신뢰도 향상	11	12.6%
정량적 효과	단축 효과 없음	1	1.1%
	20% 미만	8	9.2%
	20% ~ 40%	48	55.2%
	40% ~ 60%	21	24.1%
	60% 이상	9	10.3%

정량적 단축 효과 측면에서도 응답자의 55.2%가 20%~40%, 24.1%가 40%~60%의 뚜렷한 시간 절감을 보고했다. 이는 AI 도구가 복잡한 결함 추적과 수동 테스트의 물리적 한계를 타파하여, 제한된 납기 내 개발과 QA 조직의 협업 효율성을 극대화하는 강력한 매개체로 작동하고 있음을 뒷받침하는 통계적 근거이다.

5) 소프트웨어 배포 및 유지보수 단계

[표 4-12]는 SDLC의 마지막 단계이자 시스템의 생명 주기가 끝날 때까지 지속되는 배포 및 유지보수 단계의 AI 활용 실태를 분석한 결과이다. 흔히 운영 단계에서의 AI 도입을 논할 때 AIOps를 통한 완벽한 자율 장애 복구 등을 연상하기 쉽지만, 본 통계는 운영 환경의 제약과 리스크 관리의 보수적 특성을 보여준다.

[표 4-12] (SDLC 5단계) 배포 및 유지보수 단계 조사

		빈도	백분율
전체		30	100.0
주요 활용 도구	지능형 통합 모니터링/AIOps 플랫폼 (Datadog Bits, Dynatrace Davis, New Relic AI 등)	5	16.7%
	로그 분석 및 패턴 인식 특화 AI (Splunk AI, Elastic AI, Sumo Logic 등)	4	13.3%
	인시던트 대응 및 협업 보조 AI (PagerDuty AI, Slack AI, MS Teams Copilot 등)	8	26.7%
	릴리스 및 운영 문서화 AI (ChatGPT/Claude, Jasper, 전용 릴리스 노트 생성기 등)	13	43.3%
	기타	0	0.0%
	AI 도구를 활용하지 않음	0	0.0%
주요 효과 (단일응답)	장애 복구 시간(MTTR) 단축 및 가용성 향상	1	3.3%
	근본 원인 분석(RCA)의 자동화 및 정확도 향상	4	13.3%
	운영 문서화 및 보고 업무의 효율화	12	40.0%
	선제적 모니터링을 통한 장애 예방	7	23.3%
	운영 데이터 기반의 의사결정 지원	6	20.0%
정량적 효과	단축 효과 없음	0	0.0%
	20% 미만	5	16.7%
	20% ~ 40%	12	40.0%
	40% ~ 60%	8	26.7%
	60% 이상	5	16.7%

조사 결과, 이 단계에서 압도적으로 많이 쓰이는 도구는 '릴리스 및 운영 문서화 AI(43.3%)'였으며, 주요 효과 역시 '운영 문서화 및 보고 업무의 효율화(40.0%)'가 1위를 차지했다. 반면 궁극적인 목표라 할 수 있는 '장애 복구 시간

(MTTR) 단축 및 가용성 향상'은 단 3.3%에 그쳤다. 이 간극은 실무 환경의 구조적 특성에서 기인한다. 시스템 장애와 직결되는 운영 환경에서는 책임 소재의 엄중함과 내부 전용망 규제 등 인프라적 제약 때문에 AI에게 시스템 제어권이나 자율적 복구 권한을 완전히 위임하기 어렵다. 따라서 현장에서는 무리한 자동화보다는 리스크가 낮은 릴리스 노트 작성이나 고객 대응 가이드 생성 등 고질적인 문서화 업무 부담을 경감하는 수준에 머물러 있는 것이 현재 상황이다. 그럼에도 불구하고 '선제적 모니터링을 통한 장애 예방(23.3%)'과 '운영 데이터 기반의 의사결정 지원(20.0%)'의 수치가 유의미하게 나타난 점은 긍정적이다. 이는 AI가 단순 요약기를 넘어, 대규모 로그 데이터 속에서 인간이 놓치기 쉬운 이상 징후를 식별하는 이상 탐지(anomaly detection) 수단으로 기능하고 있음을 의미한다. 생산성 향상 지표를 살펴보면 응답자의 83.3%가 20% 이상의 뚜렷한 시간 단축을 경험했으며(20~40% 구간 40.0%, 40~60% 구간 26.7% 등), '단축 효과 없음'은 0%로 나타났다. 비록 완전한 지능형 자동 복구에는 도달하지 못했을지라도, 운영 단계에서의 AI는 번거로운 문서화와 데이터 필터링 시간을 획기적으로 줄여주어 엔지니어가 핵심 장애 원인 파악에 집중할 수 있도록 돕는 실질적인 생산성 향상의 실질적인 수단으로 작용하고 있음을 통계가 보여주고 있다.

4.2.2.4 활용 수준과 생산성 향상도 결과 분석

본 절에서 비교하는 활용 수준과 생산성 향상도 간의 격차는 기술 통계(평균) 기준의 상대적 경향이며, 두 지표는 앵커가 다른 별개 척도이고 격차의 통계적 유의성은 검정하지 않았다. 따라서 이하의 해석은 확정적 결론이 아니라 경향성 수준으로 제한하여 읽어야 한다.

1) 소프트웨어 요구사항 분석 단계

[표 4-13]과 [표 4-14]는 SDLC의 첫 단계인 요구사항 분석 단계에서 실무자들이 AI를 어느 수준으로 '활용'하고 있으며, 그 결과로 어느 정도의 '생산성 향상'을 체감하는지 10점 척도로 비교 분석한 결과이다.

먼저 [표 4-13]의 AI 활용 수준을 살펴보면, 7점 이상의 '높음' 구간에

64.8%(72명)가 집중되어 있다. 이는 프로젝트 기획 및 분석 단계에서 AI가 이미 실험적인 도입 단계를 지나 실무의 핵심 보조 도구로 안착했음을 의미한다. 하지만 활용 수준의 전체 평균 점수는 6.94점으로, 이후 이어지는 설계나 구현 단계에 비해 상대적으로 낮게 형성되어 있다. 이는 요구사항 분석이라는 업무가 지닌 본질적 특성에서 기인한다. 발주처의 숨은 의도를 파악하고 모호한 비즈니스 도메인을 해석하는 고도의 인간 중심적 소통 과정 전체를 AI에게 '전면 도입(10점)'하기에는 아직 기술적, 환경적 제약이 뚜렷하게 존재한다는 것이 평가 결과이다.

[표 4-13] 요구사항 분석단계 활용 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	2	0	5	8	24	33	25	9	5	
백분율	0.0%	1.8%	0.0%	4.5%	7.2%	21.6%	29.7%	22.5%	8.1%	4.5%	
구간합계	낮음 7명, 6.3%				보통 32명, 28.8%		높음 72명, 64.8%				

반면, [표 4-14]에 나타난 생산성 향상도 수준은 활용 수준과 매우 극적인 대조를 이룬다. AI 도입으로 인한 생산성이 '높음(7~10점)'이라고 응답한 비율은 72.9%(81명)에 달하며, 평균 점수 역시 7.31점으로 활용 수준(6.94점)을 명확하게 상회한다. 반대로 4점 이하의 '낮음' 구간은 4.5%(5명)에 불과했다. 이 두 지표 간의 긍정적인 불균형(활용 6.94점 vs 생산성 7.31점)은 매우 중요한 실무적 시사점을 던진다. 현장 인력들이 요구사항 분석 단계에서 AI를 코딩 단계처럼 항시적이고 맹목적으로 다루지 않더라도, 회의록 요약이나 요구사항 명세서 초안 작성과 같은 부분적인 적용만으로도 높은 기저효과(Base effect, 낮은 투입 대비 높은 체감 효율)를 얻고 있음을 확인할 수 있다. 즉, 최소한의 도구 개입만으로도 인간의 문서화 피로도를 획기적으로 낮추고 작업의 속도를 끌어올릴 수 있는 '투입 대비 최고 효율'의 구간이 바로 요구사항 분석 단계라는 결론을 내릴 수 있다.

[표 4-14] 요구사항 분석단계 생산성 향상도 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	0	3	2	9	16	27	28	20	6	
백분율	0.0%	0.0%	2.7%	1.8%	8.1 %	14.4 %	24.3 %	25.2 %	18.0 %	5.4%	
구간합계	낮음 5명, 4.5%				보통 25명, 22.5%		높음 81명, 72.9%				

2) 소프트웨어 시스템 설계 단계

[표 4-15]와 [표 4-16]은 소프트웨어 개발 생명주기(SDLC)의 두 번째 단계인 시스템 설계 과정에서 AI 기술의 실제 활용 수준과 그에 따른 생산성 향상 인식을 10점 만점 척도로 조사하여 비교한 결과이다. 시스템 설계는 추상적인 요구사항을 구체적인 아키텍처와 데이터 규격으로 변환하는 과정으로, 이 단계의 정합성 여부가 프로젝트 후반부의 기술 부채 규모를 결정하는 핵심 변수이다.

먼저 [표 4-15]의 AI 활용 수준을 보면, 7점 이상의 상위 구간(높음)에 전체의 68.0%(83명)가 집중되어 있으며 평균 점수는 7.14점으로 집계되었다. 이는 아키텍처 패턴 추천, DB 스키마 설계, 다이어그램 자동 생성 등 높은 전문성이 요구되는 설계 업무 영역에서도 AI가 실무자들의 일상적인 업무 보조 도구로 안착했음을 정량적으로 보여준다. 다만, 5~6점의 '보통' 구간이 25.5%(31명)로 적지 않은 비중을 차지하는 현상은 시스템 설계가 지닌 본질적 특성이 반영된 결과로 풀이된다. 정형화된 아키텍처 패턴이나 API 명세의 보일러 플레이트 생성에는 AI의 기여도가 높지만, 프로젝트마다 고유하게 존재하는 비즈니스 도메인의 특수성과 복합적인 연계 제약 조건까지 기술에 온전히 위임하기에는 아직 실무적인 한계가 따르기 때문이다.

[표 4-15] 시스템 설계 단계 활용 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	0	5	3	8	23	25	36	17	5	
백분율	0.0%	0.0%	4.1%	2.5%	6.6%	18.9 %	20.5 %	29.5 %	13.9 %	4.1%	
구간합계	낮음 8명, 6.6%				보통 31명, 25.5%		높음 83명, 68.0%				

이러한 활용 양상은 [표 4-16]의 생산성 향상도 지표와 연계하여 분석할 때 보다 입체적인 시사점을 도출한다. AI 활용을 통한 생산성 향상도가 '높음(7~10점)' 수준이라고 평가한 응답자는 73.1%(89명)로 활용 수준의 비중을 상회하고 있으며, 평균 점수 역시 7.29점으로 가로축의 활용도(7.14점)를 명확하게 넘어서고 있다.

이처럼 활용도에 비해 체감 생산성이 더 높게 나타나는 지표의 추이는 설계 단계에서 AI 기술이 발휘하는 기저효과의 구조적 성격을 규명한다. 설계 업무의 특성상 코딩 단계처럼 도구를 상시적으로 붙잡고 작업하지 않더라도, 복잡한 인프라 구성안을 도출하거나 시퀀스 다이어그램을 시각화하는 특정 과업에 한정하여 적용하는 것만으로도 실무자가 체감하는 작업시간 단축과 품질 개선의 폭이 매우 크다는 사실을 뒷받침한다. 결과적으로 초기 요구사항을 시스템 규격으로 구체화하는 과정에서 AI의 패턴 추천과 표준화 기능을 부분적으로 차용하는 행위 자체가 인터페이스 불일치와 같은 휴먼에러를 선제적으로 제어하고, 산출물의 일관성을 견인하는 핵심 생산성 향상 요인으로 작용하고 있다.

[표 4-16] 시스템 설계 단계 생산성 향상도 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	0	2	6	6	19	28	34	23	4	
백분율	0.0%	0.0%	1.6%	4.9%	4.9%	15.6 %	23.0 %	27.9 %	18.9 %	3.3%	
구간합계	낮음 8명, 6.5%				보통 25명, 20.5%		높음 89명, 73.1%				

3) 소프트웨어 구현 단계

[표 4-17]과 [표 4-18]은 소프트웨어 개발 생명주기(SDLC)에서 가장 많은 물리적 시간과 인적 자원이 투입되는 구현(코딩) 단계의 AI 활용 수준과 생산성 향상도를 측정한 결과이다. 대중적으로 AI의 개입 효과가 가장 극적으로 나타날 것으로 기대되는 구간인 만큼, 이 두 지표 간의 상관관계는 실제 개발 현장에서 AI가 지니는 효용성과 한계를 동시에 짚어보는 중요한 기준이 된다.

먼저 [표 4-17]에 나타난 구현 단계의 AI 활용 수준을 살펴보면, 7점 이상의 상위 구간(높음)에 67.9%(89명)가 분포하며 평균 7.11점을 기록했다. 이는 응답자의 절대다수가 AI를 단순한 보조 수단을 넘어 코드 자동완성, 레거시 코드 해석, 리팩터링 등 실무의 핵심 파이프라인에 적극적으로 연동해 사용하고 있음을 보여준다. 다만 5~6점의 '보통' 구간 역시 30.6%(40명)로 적지 않은 비중을 차지하고 있는데, 이는 개별 도구의 성능 문제라기보다는 사내 보안망(망분리) 제약, 프라이빗 코드베이스와의 호환성 등 외부 환경적 통제 요인이 반영된 현상으로 해석된다.

[표 4-17] 구현 단계 활용 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	1	0	1	15	25	39	29	14	7	
백분율	0.0%	0.8%	0.0%	0.8%	11.5%	19.1%	29.8%	22.1%	10.7%	5.3%	
구간합계	낮음 2명, 1.6%				보통 40명, 30.6%		높음 89명, 67.9%				

이어서 [표 4-18]의 생산성 향상도 수준을 확인해 보면, 상위 구간(높음)이 68.7%(90명)로 집계되었으며 평균 점수는 7.24점으로 나타났다. 여기서 주목할 대목은 앞선 요구사항 분석이나 시스템 설계 단계와 달리, 코딩 단계에서는 활용 수준(7.11점)과 생산성 향상도(7.24점) 간의 격차가 0.13점으로 상대적으로 좁게 나타난다는 점이다. 기획 단계에서는 제한적인 도구 활용만으로도 큰 작업량 감소를 체감했던 반면, 코딩 단계에서는 도구 투입량에 비례하는 선형적 양상에 가깝게 나타난다. 이러한 지표의 수렴 경향은 소프트웨어 구현 업무가 직면한 구조적 병목(Bottleneck)을 시사하는 것으로 해석된다. 다만 0.13점의 격차

는 기술 통계상의 값으로 유의성이 검증된 것은 아니므로, 단정적 결론보다는 경향으로 이해할 필요가 있다. AI 코딩 어시스턴트가 물리적인 상용구(Boilerplate) 작성 속도를 획기적으로 높여주더라도, 생성된 코드의 로직 정합성을 확인하고 환각(Hallucination) 현상이나 잠재적 보안 취약점을 걸러내는 '인지적 검증 작업'은 결국 인간 개발자에게 고스란히 전가된다. 즉, 코딩 단계의 최종 생산성은 AI가 코드를 얼마나 빠르게 쏟아내느냐가 아니라, 인간이 그 결과물을 얼마나 빠르고 정확하게 디버깅하여 책임질 수 있느냐에 따라 제한받고 있음을 보여준다.

[표 4-18] 구현 단계 생산성 향상도 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	0	0	3	14	24	30	33	21	6	
백분율	0.0%	0.0%	0.0%	2.3%	10.7%	18.3%	22.9%	25.2%	16.0%	4.6%	
구간합계	낮음 3명, 2.3%				보통 38명, 29.0%		높음 90명, 68.7%				

4) 소프트웨어 테스트 및 검증 단계

[표 4-19]와 [표 4-20]은 소프트웨어 개발 생명주기(SDLC)의 네 번째 단계인 테스트 및 검증 과정에서 AI 기술의 실제 활용 수준과 그에 따른 생산성 향상 정도를 10점 만점 척도로 조사하여 분석한 결과이다. 테스트 단계는 구축된 시스템의 결함을 발견하고 품질을 보증하는 최종 단계로, 이 단계에서의 지능형 자동화는 전체 프로젝트의 안정성과 납기 준수를 좌우하는 핵심 요인이다.

먼저 [표 4-19]의 AI 활용 수준을 살펴보면, 7점 이상의 상위 구간(높음)에 무려 80.4%(70명)가 집중되어 있으며 평균 점수는 7.47점으로 집계되었다. 이는 앞서 분석한 요구사항 분석(6.94점), 시스템 설계(7.14점), 구현(7.11점) 단계와 비교했을 때 전 단계 중 가장 높은 활용도를 나타내는 수치이다. 이러한 결과는 현장 실무자들이 단위 테스트 코드 자동 생성이나 정적 분석을 통한 버그 탐지 등 반복적이면서도 정밀함이 요구되는 QA 업무에 AI 도구를 매우 밀접하게 통합하여 상시적으로 활용하고 있음을 보여준다.

[표 4-19] 테스트 단계 활용 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	0	0	3	2	12	27	24	15	4	
백분율	0.0%	0.0%	0.0%	3.4%	2.3%	13.8 %	31.0 %	27.6 %	17.2 %	4.6%	
구간합계	낮음 3명, 3.4%				보통 14명, 16.1%		높음 70명, 80.4%				

이러한 적극적인 활용 양상은 [표 4-20]의 생산성 향상도 지표와 결합할 때 한층 더 정교한 상관관계를 나타낸다. AI 활용을 통한 생산성 향상도가 '높음(7~10점)' 수준이라고 응답한 비중은 73.5%(64명)이며, 평균 점수는 7.56점으로 조사되었다. 가로축인 활용도 평균(7.47점)과 세로축인 생산성 향상도 평균(7.56점) 간의 격차는 0.09점으로, SDLC 전체 단계 중 지표 간 간극이 상대적으로 가장 좁게 나타난다. 이와 같은 수렴 경향은 테스트 단계에서 AI 기술의 효용이 상대적으로 '예측 가능'하고 투입에 '비례'하는 성격을 시사하는 것으로 해석할 수 있다. 다만 0.09점은 매우 작은 차이로, 측정 오차와 구분되는 유의한 격차인지는 본 연구의 기술 통계 범위에서 단정하기 어렵다. 기획이나 설계 단계처럼 제한적인 도구 사용으로 폭발적인 기저효과를 누리는 것과 달리, 테스트 단계에서는 도구를 실무에 정착시키고 투입하는 양에 정확히 비례하여 안정적이고 정직한 생산성 개선 효과가 도출된다는 의미로 풀이된다.

[표 4-20] 테스트 단계 생산성 향상도 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	0	0	2	3	18	15	23	22	4	
백분율	0.0%	0.0%	0.0%	2.3%	3.4%	20.7 %	17.2 %	26.4 %	25.3 %	4.6%	
구간합계	낮음 2명, 2.30%				보통 21명, 24.1%		높음 64명, 73.5%				

동시에 이 결과는 현장 실무자들이 AI 기반 정적 분석 및 버그 탐지 도구를 활용하여 업무 시간 단축을 넘어 테스트 시나리오의 다양성과 결함 탐지의 정확도까지 확보하고 있음을 보여준다. 단 2.3%만이 생산성 향상이 낮다고 응답한

결과 역시, 테스트 스크립트 작성의 피로도를 낮추고 개발 조직과 QA 조직 간의 협업 피드백 루프를 단축하는 데 AI가 안정적인 매개체로 작동하고 있음을 나타낸다.

5) 소프트웨어 배포 및 유지보수 단계

[표 4-21]과 [표 4-22]는 소프트웨어 개발 생명주기(SDLC)의 최종 단계인 배포 및 유지보수 환경에서 AI 기술의 실제 활용 수준과 생산성 향상도를 10점 척도로 측정한 결과이다. 시스템 운영과 직결되는 이 단계는 선제적인 장애 예방과 신속한 사후 대응이 전체 서비스의 가용성을 결정짓는 중요한 구간이다.

먼저 [표 4-21]의 AI 활용 수준을 살펴보면, 7점 이상의 상위 구간(높음)에 전체의 70.0%(21명)가 분포하며 평균 7.60점을 기록했다. 이는 앞선 4개의 개발 단계 중 가장 높은 활용도 수치이다. 다만 배포·유지보수 단계 응답자는 30명으로 타 단계(87~131명) 대비 크게 적어, 본 단계의 평균값은 표본 규모가 작다는 점을 전제로 해석해야 한다. 시스템 로그 분석, 인시던트 모니터링, 릴리스 노트 작성 등 방대한 데이터 처리와 텍스트 요약이 요구되는 운영 업무의 특성상, AIOps나 지능형 모니터링 도구가 실무자들의 일상적인 업무 프로세스에 깊숙이 통합되어 있음을 시사한다. 특히 4점 이하의 '낮음' 응답이 전무(0%)하다는 점은 해당 단계에서 AI 도구의 개입이 이미 필수적인 운영 환경임을 보여준다.

[표 4-21] 배포 및 유지보수 단계 활용 수준

전혀 활용안함	1	2	3	4	5	6	7	8	9	10	전면 도입
빈도	0	0	0	0	2	7	4	10	2	5	30
백분율	0.0%	0.0%	0.0%	0.0%	6.7%	23.3 %	13.3 %	33.3 %	6.7%	16.7 %	
구간합계	낮음 0명, 0.0%				보통 9명, 30.0%		높음 21명, 70.0%				

이러한 높은 활용도는 [표 4-22]의 생산성 향상도 지표에서 더욱 극대화된 수치로 이어지고 있다. AI 도입을 통한 생산성 향상도가 '높음(7~10점)'이라고 응답한 비율은 76.6%(23명)에 달하며, 평균 점수는 7.83점으로 집계되었다. 이는 SDLC 5개 전 단계 중 절대적인 점수 기준 최고치에 해당한다. 단, 앞서 밝

혔듯 본 단계 표본은 30명에 불과하여 평균의 신뢰구간이 넓고 타 단계와의 직접 비교에는 한계가 있으므로, 이 최고치는 확정적 우위라기보다 참고 수준으로 받아들여야 한다. 가로축인 활용도 평균(7.60점)과 세로축인 생산성 향상도 평균(7.83점) 간의 격차는 0.23점으로, 활용 수준을 상회하는 뚜렷한 체감 효과가 드러난다.

이와 같은 최고 수준의 지표는 배포 및 유지보수 단계에서 AI의 투입 대비 높은 산출 증대 효과의 단면을 드러낸다. 실제 엔터프라이즈 환경에서는 망분리 제약이나 장애 발생 시의 책임 소재 문제로 인해 AI가 시스템을 완전히 자율적으로 통제하고 복구하는 수준에 이르지는 못했다. 그러나 수백 대의 서버에서 발생하는 비정형 로그 데이터 속에서 이상 징후를 필터링하고, 복잡한 장애 원인 분석(RCA) 보고서의 뼈대를 구성해 주는 기능만으로도 운영 담당자가 감당해야 할 물리적 피로도와 인지적 탐색 비용은 기하급수적으로 감소한다.

[표 4-22] 배포 및 유지보수 단계 생산성 향상도 수준

전혀 활용안함	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**전면 도입**
빈도	0	0	0	0	2	5	7	3	8	5	30
백분율	0.0%	0.0%	0.0%	0.0%	6.7%	16.7%	23.3%	10.0%	26.7%	16.7%	
구간합계	낮음 0명, 0.0%				보통 7명, 23.4%		높음 23명, 76.6%				

결과적으로 운영 단계에서 AI는 시스템의 최종 제어권을 가진 인간 엔지니어가 장애 복구라는 핵심 의사결정에만 집중할 수 있도록, 주변부의 소모적인 모니터링과 문서화 부담을 크게 덜어주는 실질적이고 안정적인 자동화 기반으로 기능하고 있다는 결론을 보여준다.

6) 소프트웨어 개발 단계별 활용수준-향상도 평균

[표 4-23]의 정량적 데이터와 이를 2차원 평면에 도식화한 [그림 4-1]을 결합해 보면, 소프트웨어 개발 생명주기(SDLC) 전반을 관통하는 AI 도구의 실질적 파급력을 한눈에 파악할 수 있다. 도식화된 좌표들의 전체적인 궤적은 기획 및 설계의 초기 단계에서 코딩과 테스트를 거쳐 최종 운영 단계로 나아갈수록 우상향하는 뚜렷한 추세를 그린다.

[표 4-23] SDLC 단계별 활용수준-향상도 평균 점수

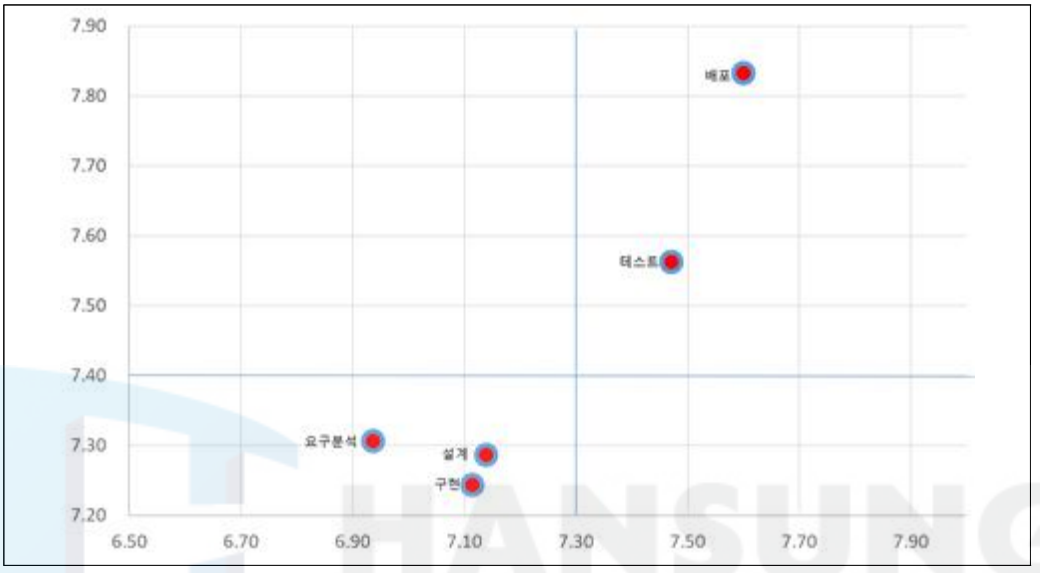
구분		1	2	3	4	5	6	7	8	9	10	빈도 (A)	가중 합계 (B)	평균 (B/A)
1단계 (요구)	활용	0	2	0	5	8	24	33	25	9	5	111	770	6.94
	생산성	0	0	3	2	9	16	27	28	20	6	111	811	7.31
2단계 (설계)	활용	0	0	5	3	8	23	25	36	17	5	122	871	7.14
	생산성	0	0	2	6	6	19	28	34	23	4	122	889	7.29
3단계 (구현)	활용	0	1	0	1	15	25	39	29	14	7	131	932	7.11
	생산성	0	0	0	3	14	24	30	33	21	6	131	949	7.24
4단계 (테스트)	활용	0	0	0	3	2	12	27	24	15	4	87	650	7.47
	생산성	0	0	0	2	3	18	15	23	22	4	87	658	7.56
5단계 (배포)	활용	0	0	0	0	2	7	4	10	2	5	30	228	7.60
	생산성	0	0	0	0	2	5	7	3	8	5	30	235	7.83

* 가중합계(B)는 각 리커트 점수에 빈도를 곱하여 합산한 수치

그래프에서 가장 두드러지는 지점은 양극단에 위치한 1단계(요구사항 분석)와 5단계(배포 및 유지보수)의 상반된 지표다. 먼저 5단계 배포 및 유지보수 구간은 가로축(활용 수준 7.60)과 세로축(생산성 7.83) 모두에서 최고점을 기록하며 우측 상단에 위치한다. 이는 정형화된 로그 데이터 필터링이나 시스템 장애 탐지 같은 패턴화된 반복 업무에서 AI의 지능형 자동화가 가장 즉각적이고 확실한 효용을 제공하고 있음을 보여준다.

반대편에 위치한 1단계 요구사항 분석의 좌표는 더욱 흥미로운 결과를 보여준다. 1단계의 도구 활용 수준은 6.94점으로 전체 5개 단계 중 가장 낮다. 그러나 생산성 향상도는 7.31점으로 크게 상승해 있다. 두 지표 간의 격차가 0.37점으로 전 구간을 통틀어 가장 크다. 분석 단계에서는 도구를 온종일 붙잡고 있지

않더라도, 회의록 요약이나 문서 초안 생성 등 특정 과업에 한정하여 AI를 투입하는 것만으로 체감되는 시간 절감 효과(기저효과)가 가장 크게 체감된다. 투입 대비 효율이 가장 극대화되는 구간이다.

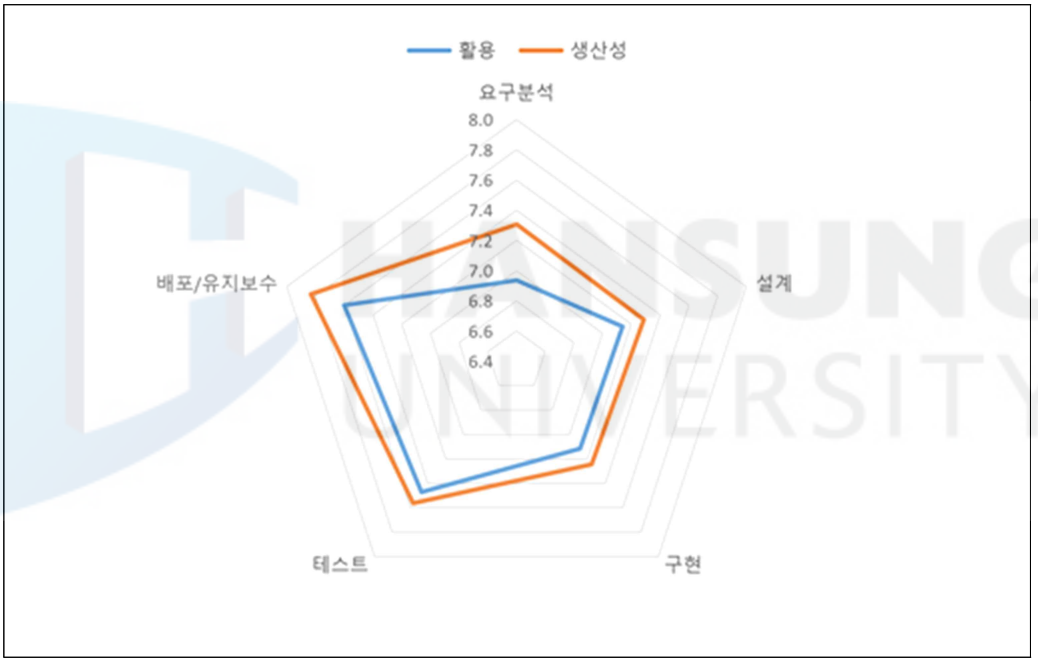


[그림 4-1] SDLC 단계별 활용수준-향상도 평균점수 상대 위치

반면, 가장 격차가 클 것으로 기대했던 3단계 구현/코딩 구간의 결과는 상대적으로 낮은 수준이다. 생산성 향상도 평균이 7.24점으로 5개 단계 중 최하위이며, 도구 활용 수준(7.11점)과의 격차 역시 0.13점에 불과하다. 시각적 분포 역시 상위 구간에 미치지 못한 채 2단계(시스템 설계)와 함께 중앙부에 머물러 있다. 이는 AI 코딩 어시스턴트가 물리적인 타자 속도를 높여주더라도, 생성된 코드의 논리적 결함을 찾아내고 환각(Hallucination) 오류를 방어해야 하는 개발자의 인지적 검증 부담이 여전히 크다는 현실적 한계를 데이터에서 관찰되는 경향으로 해석할 수 있다.

결과적으로 이 통합 지표는 AI를 '단순 구현 보조기'로 보는 단편적인 인식과 배치되는 결과를 보여준다. 오히려 초기 기획 단계의 문서화부터 최종 운영 단계의 지능형 모니터링에 이르기까지, 인간의 피로도를 가중시키는 주변부 작업들을 덜어줄 때 SDLC 전체의 생산성이 가장 안정적으로 개선된다는 사실을 명확히 보여준다.

[그림 4-2]는 소프트웨어 개발 생명주기(SDLC) 5개 단계 각각에서 AI 도구의 '활용 수준'과 '생산성 향상도' 간의 상대적 격차(Gap)를 직관적으로 비교한 지표이다. 이 그래프가 던지는 1차적인 메시지는 명확하다. 모든 개발 단계에서 예외 없이 생산성 향상도가 활용 수준을 웃도는 '양(+)의 격차'가 발생했다는 점이다. 이는 도구를 익히고 업무에 적용하기 위해 실무자가 투입한 인지적 노력보다, 그로 인해 돌려받는 시간 단축 및 품질 개선의 보상이 구조적으로 더 크다는 사실을 데이터가 직접 보여준다. 지표의 변동성을 세밀하게 들여다보면 각 개발 단계가 지닌 업무적 본질과 한계가 극명하게 드러난다.



[그림 4-2] SDLC 단계별 활용수준-향상도 단계별 상대 비교

분석 결과에서 가장 눈여겨볼 구간은 두 지표 간의 격차가 가장 좁게 수렴하는 3단계(구현/코딩, 0.13점 차이)와 4단계(테스트 및 검증, 0.09점 차이)이다. 이 구간은 개발자들이 가장 빈번하고 치열하게 AI를 사용하는 영역임에도 불구하고 활용도 대비 생산성이 폭발적으로 뻗어나가지 못한다. 원인은 '검증 비용'에 있다. AI가 생성하는 코드의 양이 늘어날수록 그 로직의 무결성을 입증하고 환각(Hallucination) 오류를 잡아내기 위한 인간의 인지적 피로도가 반드시 동반되

기 때문이다. 즉, 코딩과 테스트 단계에서의 AI는 투입에 비례하는 효과를 내는 '선형적 생산 도구'로 작동한다. 반면, 1단계(요구사항 분석)는 0.37점이라는 가장 극적인 격차를 만들어냈다. 초기 기획 단계에서는 하루 종일 도구를 켜두고 작업하지 않더라도, 회의록 요약이나 명세서 초안 작성 등에 파편적으로 AI를 개입시키는 것만으로도 큰 기저효과를 얻을 수 있다. 투입 대비 산출 효율이 가장 극대화되는 구간이다. 마지막 5단계(배포 및 유지보수)는 두 지표 모두 전체 최고치를 찍는 동시에 다시 격차를 0.23점으로 벌렸다. 이는 AIOps와 지능형 모니터링 체계가 운영 환경에 안착할 경우, 단순한 작업 보조를 넘어 치명적인 장애 복구 시간(MTTR)을 선제적으로 단축하는 등 구조적 차원의 생산성 향상의 중요 수단으로서의 역할을 수행하고 있음을 보여준다.

아울러 각 단계에서 함께 측정한 정량적 시간 단축률(%)과 10점 척도 생산성 향상도는 대체로 같은 방향의 경향을 보여, 두 측정치가 상호 일관됨을 확인할 수 있다. 이는 체감 지표의 내적 신뢰도를 뒷받침한다.

4.2.2.5 AI 도입의 장애 요인 및 인식

[표 4-24]는 소프트웨어 개발 생명주기(SDLC) 내 AI 도입 과정에서 조직이 직면하는 장애 요인을 직급별로 교차 분석한 결과다. 전체 응답자가 가장 심각하게 받아들이는 위협은 ‘기술적 불확실성 및 환각 현상(33.0%)’과 ‘투자 대비 효율의 불투명성(27.0%)’으로 나타났다. 특히 실무의 중추를 담당하는 중·고급 기술자 그룹에서 환각 현상에 대한 우려가 압도적으로 높다. 이는 앞서 구현 단계의 지표에서 확인했듯, AI가 쏟아낸 결과물을 디버깅하고 논리적 오류를 최종적으로 책임져야 하는 실무진의 인지적 과부하가 수치로 발현된 결과다. 이 통계에서 가장 눈여겨볼 대목은 직급에 따른 '위험 인식의 비대칭성'이다. 전체 통계에서는 5.5%로 최하위를 기록한 ‘기존 개발 프로세스 및 조직 문화와의 부조화’ 항목을 18년 이상의 관리자 그룹은 무려 25%(12명 중 3명)나 주요 우려 사항으로 지목했다.

코드를 직접 다루는 실무진이 AI의 기술적 결함과 ROI라는 미시적 문제에 집중할 때, 프로젝트의 의사결정권자들은 새로운 도구가 기존의 폭포수 방법론이나 사내 거버넌스와 충돌하며 만들어 낼 거시적 저항을 더 큰 위협으로 판단

하고 있다. 결국 AI의 성공적인 안착을 위해서는 실무진의 불안을 잠재울 기술적 방어 체계와 관리자를 설득할 변화 관리 지표가 동시에 수립되어야 함을 데이터가 보여주고 있다.

[표 4-24] AI 도입 시 가장 크게 우려되는 기술적/조직적 문제(등급별)

구분		등급 분류					빈도	백분율
전체		관리자	특급	고급	중급	초급	200	100.0
		12	24	58	91	15		
기술적/조직적 문제	보안 및 지식재산권 유출	3	5	8	15	2	33	16.5%
	기술적 불확실성 및 환각	4	3	24	29	6	66	33.0%
	투자 대비 효율의 불투명성	1	7	10	32	4	54	27.0%
	사내 AI 숙련도 부족 및 교육 부재	1	8	12	14	1	36	18.0%
	기존 개발 프로세스 및 조직 문화와의 부조화	3	1	4	1	2	11	5.5%

[표 4-25]는 AI 기술이 향후 개발자의 직무역량에 미칠 영향에 대한 인식을 분석한 지표다. 이 결과는 일반적인 통념과 상반된다. 현장의 개발자들은 AI로 인한 '직무 대체 및 고용 불안(7.5%)'을 가장 덜 위협적인 요소로 꼽았다. 그들이 체감하는 실재하는 위기는 '전문성 저하 및 도구 의존(32.0%)'과 '지속적 학습에 대한 심리적 압박(23.0%)'에 집중되어 있다. 즉, 현장 인력들은 AI를 자신을 쫓아낼 대체재로 보는 것이 아니라, 기술에 종속되어 엔지니어링 기본기와 스스로의 문제 해결 능력을 잃어버릴 수 있다는 점을 가장 심각한 리스크로 받아들인다. 직급별 교차 분석은 이러한 현상을 한층 더 입체적으로 보여준다. 18년 이상 경력의 관리자 그룹에서는 '업무 생산성 극대화'를 선택한 비율이 타 직급 대비 가장 높았다. 반면 코드를 직접 생산하는 초급부터 특급에 이르는 대다수의 실무자 그룹은 한결같이 전문성 저하와 학습 압박을 최상위 리스크로 지목했다. 조직의 리더는 AI를 통한 고부가가치 창출과 전사적 효율성에 주목하지만, 현장 실무자는 도구에 의존하다 하향 평준화되는 자신의 역량 저하를 크게 우려하고 있다. 이러한 인식 격차는 개발 조직 내에 AI를 도입할 때 단순히 유료 라

이선스를 일괄 지급하는 방식에서 벗어나, 실무자의 전문성을 보호하고 인간 중심의 직무 재설계가 반드시 동반되어야 한다는 사실을 시사한다.

[표 4-25] AI 기술이 향후 귀하의 직무역량에 미칠 영향(등급별)

구분		등급 분류					빈도	백분율
	전체	관리자	특급	고급	중급	초급	200	100.0
		12	24	58	91	15		
AI 기술이 직무역량에 미치는 영향	업무 생산성 극대화	5	5	7	14	1	32	16.0%
	직무 역할의 전환	1	2	14	24	2	43	21.5%
	전문성 저하 및 도구 의존	2	7	17	31	7	64	32.0%
	지속적 학습에 대한 심리적 압박	2	7	13	19	5	46	23.0%
	직무 대체 가능성 및 고용 불안	2	3	7	3	0	15	7.5%

4.3 전문가 그룹 인터뷰

연구 계획에 따라 소프트웨어 개발 현장에서 활동하고 있는 전문가를 대상으로 설문조사 결과의 내용에 대해 소프트웨어 개발 현장에서 AI를 활용하여 생산성을 높이기 위한 인터뷰를 실시하였다.

4.3.1 인터뷰 개요

인터뷰는 설문조사 결과 데이터를 전문가에게 제시하고 전문가가 답한 내용을 요약하여 정리하는 방식으로 진행하였다. 인터뷰 질의 사항은 상기 설문조사 결과에 따라 SDLC 단계별 AI 도구 활용 현상 분석, AI 활용 수준과 생산성 향상도 결과 분석, AI 도입의 장애 요인 및 인식 분석으로 진행하였다.

- 질의1. SDLC 단계별 AI 도구 활용 현상 분석
- 질의2. AI 활용수준과 생산성 향상도 결과 분석
- 질의3. AI 도입의 장애 요인 및 인식 분석

4.3.2 인터뷰 참여자

인터뷰는 금융SI 분야에서 활동하는 경력 28년, 20년의 전문가 2명과 공공 SI 분야에서 활동하는 경력 25년의 전문가 1명을 포함한 총 3명의 전문가에게 설문 결과를 설명하고, 이에 대한 해석과 대안 등을 청취하여 정리하는 방식으로 진행하였다.

[표 4-26] 인터뷰 참여자 현황 (20년 이상 관리자급 3명)

구분	경력	관련 산업	참여기관 (비실명화)
전문가1	개발자(경력 28년)	금융 SI	B사 Y상무
전문가2	개발자(경력 20년)	금융 SI	P사 J부장
전문가3	개발자(경력 25년)	공공 SI	I사 L이사

한편 인터뷰 참여자 3인은 모두 금융·공공 SI 분야의 대형 프로젝트 경력자로 구성되어, 폐쇄망·규제 환경에 대한 관점이 상대적으로 강하게 반영될 수 있다. 스타트업·자체 솔루션 개발사 등 상이한 환경의 전문가 의견은 후속 연구에서 보완될 필요가 있다.

4.3.3 분석 결과 검토 의견

4.3.3.1 질의1. SDLC 단계별 AI 도구 활용 현상 분석

[표 4-27]은 설문조사에서 도출된 정량적 데이터가 실제 개발 현장에서 어떤 맥락으로 작동하고 있는지, 20년 이상 경력의 금융 및 공공 SI 전문가 3인의 심층 인터뷰를 통해 분석한 결과다. 전문가들의 진단은 AI가 단순한 코딩 보조 도구에 불과하다는 단편적 인식을 깨뜨리고, 소프트웨어 개발 생명주기(SDLC) 전반에 걸친 기술 도입의 현실과 한계를 뚜렷하게 짚어낸다. 가장 눈에 띄는 현상은 프로젝트 초기 단계(요구사항 분석 및 시스템 설계)에서의 예상치 못한 도입 확산이다. 흔히 코딩 단계에 활용이 집중될 것이라는 통념과 달리, 현장에서는 명세서 초안 작성, 요구사항 누락 식별, 산출물 표준화 등 초기 기획 및 분석 작업에 AI를 적극적으로 투입하고 있다

[표 4-27] (인터뷰 요약1) SDLC 단계별 AI 도구 활용 현상 분석

구분	전문가 의견
전문가1	○ 프로젝트 현장에서는 주로 구현/코딩 단계에서 AI 도구를 사용하고 있는데, 요구사항 분석 단계, 시스템 설계 단계에서 주요 산출물 생성 시 AI 도구 사용률이 예상보다 높은 것이 특징적이다. ○ 시스템 분석/설계 단계에서 AI 도구 사용률은 지속적으로 증가할 것으로 예측된다.
전문가2	○ SDLC 전반에 AI 도구 활용이 확산되어 주변에서도 분석 자료대로 요구사항·설계·구현·테스트 단계까지는 활발히 쓰이지만, 운영 배포 및 유지보수 활용하기가 힘들다(내부전용망 등). ○ AI 코드 생성은 결과 검증이 즉각적이고 실패 비용이 낮은 반면, 운영은 책임 소재와 장애 리스크가 크다. 실제로 배포 단계 AI는 릴리스 노트·장애 보고서 등 문서 작성에 집중되고, 핵심 운영 및 위험도가 높은 영역일수록 아직 사람이 쥐고 있음을 보여주고 있다고 생각한다.
전문가3	○ 설문 결과를 보면 AI 도구 활용은 구현/코딩 단계에만 집중된 것이 아니라 요구사항 분석, 설계, 테스트, 배포·운영까지 SDLC 전 단계로 확산되고 있는 것으로 보입니다. 특히 구현 단계에서는 AI 네이티브 에디터와 IDE 확장형 코딩 어시스턴트의 활용 비중이 높고, 설계 단계에서는 다이어그램·시각화 AI와 클라우드/인프라 설계 보조 AI가 의미 있게 활용되고 있다는 점이 인상적이다. ○ 다만, 단계별로 활용 성격은 다른데. 요구사항·설계 단계에서는 초안 작성, 누락·모순 식별, 산출물 표준화 같은 문서·분석 보조 성격이 강하고, 구현·테스트 단계에서는 코드 생성, 리팩터링, 버그 탐지, 테스트 시나리오 생성처럼 직접적인 생산성 향상 효과가 나타나는 것 같다. ○ AI는 아직 SDLC 전체를 자동화하는 수준이라기보다, 각 단계의 반복적·문서적·검증적 업무를 보조하면서 아직까지는 개발자의 업무를 지원을 강화하는 방향으로 활용되고 있다고 판단된다.

이는 AI가 프로젝트를 주도적으로 이끌 수는 없어도, 개발 주기 초반에 몰려 있는 방대한 문서 작업의 피로도를 낮추고 산출물의 품질을 상향 평준화하는 실질적인 수단으로 잡았음을 보여준다. 반면, 배포 및 유지보수와 같은 운영 단계로 넘어갈수록 AI의 자율성은 급격히 축소되며 엄격한 인간의 통제 아래 놓이는 현상이 목격된다. 전문가들은 구현이나 테스트 단계에서의 AI 생성 코드는 즉각적인 검증이 가능하고 실패 비용이 낮지만, 실제 운영 환경에서의 결함은 감당

해야 할 리스크와 책임 소재가 막대하다고 지적한다. 특히 금융 및 공공 SI 사이트 고유의 폐쇄망(망분리) 환경이라는 인프라적 한계가 더해져, 시스템 운영 구간에서 AI의 역할은 자율 제어가 아닌 릴리스 노트나 장애 보고서 작성과 같은 안전한 보조 업무에 머물고 있다. 결과적으로 현장 전문가들의 시각은 기술의 한계를 인정하는 동시에 그 효용을 극대화하는 방향으로 수렴한다. 현재 실무 환경에서 AI는 SDLC 전 과정을 인간의 개입 없이 수행하는 완전 자동화 수단이 아니다. 오히려 초기 단계에서는 문서 보조와 분석을, 후반 단계에서는 코드 생성과 버그 탐지를 담당하며, 각 단계마다 인간 개발자가 감당해야 할 소모적이고 반복적인 검증 작업을 덜어주는 '강력한 실무 보조자'로서 기능하고 있다.

4.3.3.2 질의2. AI 활용수준과 생산성 향상도 결과 분석

[표 4-28]에 요약된 현장 전문가 3인의 검토 의견은 앞선 설문조사의 정량적 지표가 내포한 실무적 의미와 한계를 함께 지적한다. 이들은 공통적으로 모든 개발 단계에서 도구 활용 수준보다 체감 생산성 향상도가 높게 도출된 현상에 주목하며, 현장 실무진이 부분적인 도구 도입만으로도 뚜렷한 기저효과를 누리고 있다는 사실을 인정한다.

그러나 전문가들의 논의는 단순한 현상 긍정에 머무르지 않고, 생산성 향상의 '질적 차이'를 구분하는 데 집중된다. 테스트 시나리오 초안을 짜거나 운영 단계의 릴리스 노트를 작성하는 등 반복적이고 소모적인 업무에서는 도구의 단편적 투입만으로도 즉각적인 피로도 감소 효과를 거둘 수 있다. 반면, 전체 프로젝트 리소스의 상당 부분이 집중되는 구현(코딩) 및 단위 테스트 단계의 실질적인 효율을 극대화하려면 개인의 활용을 넘어선 조직적 접근이 요구된다. 파편화되어 있는 개별 개발자들의 프롬프트를 팀 단위로 표준화하고, 이를 프로젝트의 핵심 자산으로 내재화하는 체계적인 가이드라인 수립이 프로젝트 성공의 필수 전제조건으로 대두된다.

나아가, 설문 데이터가 보여준 긍정적인 지표를 이성적으로 경계해야 한다는 비판적 시각도 핵심 시사점으로 도출된다. 현재 실무자들이 설문을 통해 보고한 높은 생산성 점수는 다분히 '주관적 체감'에 기반하고 있다. 개발자 개인의 코딩 속도가 빨라지고 문서 작성의 번거로움이 줄어들었다는 심리적 만족감이 실제

비즈니스 가치 창출이나 소프트웨어 품질의 절대적 향상으로 직결된다고 단정할 수는 없다. 결국 현재의 체감 지표가 진정한 엔지니어링 혁신으로 인정받기 위해서는, 향후 실제 프로젝트의 개발 리드타임(Lead Time) 단축률, 배포 후 결함률(Defect Rate), 장애 복구 시간(MTTR) 등 조작 불가능한 객관적 하드 데이터(Hard Data)를 통한 교차 검증이 반드시 뒤따라야 함을 현장 전문가들은 한목소리로 지적하고 있다.

[표 4-28] (인터뷰 요약2) AI 활용수준과 생산성 향상도 결과 분석

구분	전문가 의견
전문가1	○ 설문조사결과 모든 프로젝트 라이프사이클에서 생산성 향상도 수준이 높은것으로 나타난다. 다만, 구현/코딩 단계와 테스트 단계가 프로젝트에서 차지하는 비중이 높기 때문에 구현/코딩 단계와 테스트 단계에서의 활용도를 높이는 것이 프로젝트 성공의 중요한 포인트라 볼 수 있다. ○ 구현/코딩 단계에서는 프로젝트 내에서 표준화된 프롬프트 작성 및 공유를 통해 생산성 및 효율성을 보다 더 높일 수 있을 것이다.
전문가2	○ 활용 수준과 생산성 향상도를 비교하면, 모든 단계에서 생산성 체감 평균이 활용 수준보다 높게 나타난다. AI를 도입해서 그 효과를 체감하고 있는듯 하며, 향후 활용도가 높아질수록 생산성 기대가 크다고 생각한다.
전문가3	○ AI 활용 수준과 생산성 향상도 평균 점수를 보면 모든 SDLC 단계에서 생산성 향상도 점수가 활용 수준 점수보다 높게 나타난다. 이는 아직 AI를 전면적으로 도입하지 않았더라도, 부분적으로만 활용해도 체감 생산성 개선 효과가 높다는 의미로 해석될 것 같다. ○ 특히 배포 및 유지보수 단계와 테스트 단계에서 평균 점수가 높게 나온 점이 특이하다. 일반적으로 AI 생산성 효과는 코딩 단계에서 가장 클 것으로 생각하였는데, 본 설문결과에서는 테스트, 배포·운영 단계에서도 높은 효과가 확인되서, 실제 현장에서 테스트 시나리오 작성, 결함 원인 분석, 릴리스 노트 작성, 운영 보고서 작성 등 반복적이고 시간이 많이 소요되는 업무에서 AI가 빠르게 효과를 만들어낼 수도 있다는 뜻으로 해석할 수도 있다. ○ 다만, 생산성 향상도는 응답자의 주관적 체감이 반영될 가능성이 많으므로, 향후 연구에서는 실제 개발 리드타임, 결함률, 테스트 커버리지, 장애 복구 시간, 문서 작성 시간 등 객관적 지표와 함께 검증하면 설득력이 더 높아질 것이다.

4.3.3.3 질의3. AI 도입의 장애 요인 및 인식 분석

[표 4-29]에 요약된 현장 전문가들의 견해는 AI 도입을 가로막는 장애 요인을 단기적인 기술 한계와 장기적인 인적 리스크로 분리하여 진단한다.현장에서 당면한 1차적 장벽은 외부 프로젝트 환경의 보안 취약점, 라이선스 과금, 그리고 환각(Hallucination) 현상이 유발하는 막대한 검증 부담이다.

[표 4-29] (인터뷰 요약3) AI 도입의 장애 요인 및 인식 분석 (정성적)

구분	전문가 의견
전문가1	○ 외부에서 수행하는 프로젝트의 경우 보안 취약점에 대한 이슈가 지속적으로 제기되고 있으며, 사용량 증가에 따른 과다한 과금 또한 AI 도입의 장애 요인이 될 수 있다.
전문가2	○ 앞선 자료에서 응답자들의 AI 사용 빈도는 높지만, 환각·검증 비용(33%), 다른 하나는 전문성 저하·도구 의존(32%)이다. 환각·검증 비용은 AI 성능 발전에 따라 점차 줄어들 기술적 문제로 보인다. ○ 반면 전문성 저하와 도구 의존은 기술이 아닌 사람의 역량 문제이며, 특히 의존도가 중급·고급 실무자에게 집중되는 만큼 단순히 시간이 해결해 주지 않는다. 따라서 검증 책임과 기초 역량 유지를 관리·감독자가 의도적으로 설계해야 하며, 그 역할의 비중이 점점 커질 것으로 보인다.
전문가3	○ AI 도입의 가장 큰 우려 요인으로 기술적 불확실성과 환각 문제가 가장 높게 나타난 것은 타당한 결과라고 생각한다. 실제 개발 현장에서도 AI가 만든 코드나 설계안이 그럴듯해 보여도 정확성, 보안성, 운영 적합성에 대해서 반드시 검증이 필요하므로 AI 활용이 곧바로 공수 절감으로만 이어지지는 않을 것 같다. 즉, 검증 비용과 책임 소재가 함께 증가할 수 있다는 점도 간과하면 안될 것 같다. ○ 투자 대비 효율의 불투명성이 두 번째로 높게 나온 것도 현실적인 것 같다. 많은 조직이 AI 도구를 도입하고 있지만, 실제로 어느 단계에서 얼마만큼의 비용 절감이나 품질 개선이 있었는지를 정량적으로 측정하는 체계는 아직 부족한 것 같다. ○ 직무 영향 측면에서는 직무 대체 불안보다 전문성 저하와 도구 의존에 대한 우려가 더 높게 나타난 점이 중요한 것 같다. 이는 개발자들이 AI에 과도하게 의존할 경우 문제 해결 능력 등이 부족해질 수 있다는점을 리스크로 보고 있다는 의미인 것 같다. 따라서 향후 AI 도입 방향은 “모든 문제를 AI가 답을 내는 구조”가 아니라 “중요한 결정은 사람이 판단 및 결정하고 AI가 지원하는 구조”로 설계되어야 하며, 교육 역시 프롬프트 사용법뿐 아니라 결과 검증, 코드 리뷰 등의 역량을 함께 강화하는 방향으로 진행해야 된다고 생각한다.

AI가 생성한 코드를 실제 프로덕션 환경에 적용하기 위해서는 책임 소재를 담보할 철저한 검증이 필수적이다. 따라서 도구를 투입하더라도 그것이 곧바로 순수한 인력 투입 절감이나 명확한 ROI(투자 대비 효율) 확보로 이어지지는 않는 현장의 구조적 모순을 통계 데이터와 일치시켜 짚어낸다.

그러나 이들이 공통적으로 경고하는 가장 큰 위협은 기술의 불완전성이 아닌 '전문성 저하'라는 인적 리스크에 있다. 환각 오류와 같은 기술적 결함은 AI 모델이 고도화됨에 따라 점진적으로 해소될 통제 가능한 변수다. 반면, 실무 개발자들이 도구에 종속되어 스스로의 문제 해결 능력과 엔지니어링 기초 체력을 상실하는 현상은 시간이 해결해 주지 않는 영구적인 위기다. 단순 코딩을 기계에 위임할수록 인간은 산출물을 비판적으로 검토하고 전체 아키텍처를 통제하는 상위 수준의 역할을 맡아야 하지만, 도구에 맹목적으로 의존하는 업무 환경에서는 이러한 역량 전이가 원천적으로 차단된다. 결과적으로 이러한 진단은 기업의 AI 도입 전략이 전면 수정되어야 함을 강력하게 시사한다. 단순히 유료 라이선스를 일괄 지급하고 프롬프트 작성법을 교육하는 1차원적 접근은 도구 종속성만 가중시킬 뿐이다. 성공적인 기술 안착을 위해서는 '최종 의사결정은 인간이 책임지고 AI는 이를 보조하는 구조'를 시스템적으로 강제해야 한다. 개발 파이프라인 내에 산출물 교차 검증 의무를 제도화하고, 사내 교육의 무게중심을 도구 사용법에서 '코드 리뷰 및 품질 검증 역량 강화'로 즉각 전환하는 의도적인 직무 재설계(Job Redesign)가 조직 차원에서 실행되어야 한다.

4.4 소결

본 절은 제4장에서 도출한 정량적 설문조사 결과와 정성적 심층 인터뷰 내용을 결합하여 종합적으로 분석한다. 소프트웨어 개발 생명주기(SDLC) 단계별로 나타난 AI 기술 활용의 객관적 실태와 인식을 종합하여 최종적인 통계적 결론을 도출한다.

4.4.1 SDLC 단계별 활용 수준 및 생산성 향상 효과

설문조사와 전문가 인터뷰 결과는 소프트웨어 개발 생명주기 전반에서 AI 도구의 물리적 활용 수준보다 실무진이 체감하는 생산성 향상 지표가 일관되게 상회하는 양(+)의 격차가 나타났다. 이러한 격차의 분포는 각 개발 단계의 고유한 과업 특성과 검증 비용의 크기에 따라 상이한 궤적을 그린다. 초기 기획 및 요구사항 분석 단계는 도구의 활용 빈도가 전 단계 중 가장 낮음에도 생산성 체감도와의 격차를 가장 크게 벌린다. 회의록 요약이나 문서 초안 자동 생성 등 제한적인 개입만으로도 작업 리드타임을 획기적으로 절감하는 강력한 기저효과가 작동하기 때문이다. 반면 대중적 기대감이 큰 구현(코딩) 단계는 실무자들의 도구 활용 수준이 매우 높으나 생산성 향상도와의 지표 간 간극이 가장 좁게 수렴한다. AI가 코드를 신속하게 출력하더라도 논리적 정합성 검증, 기술적 환각 오류 수정, 보안 취약점 디버깅 등에 투입해야 하는 인간 개발자의 인지적 검증 부담이 상존하는 업무 구조적 한계 때문이다. 품질 검증과 배포 및 유지보수 등 후반부 단계는 정량 지표상 최고 수준의 생산성 향상 수치를 기록한다. 그러나 이는 시스템 제어권의 전면 위임이 아닌, 단위 테스트 코드 자동 생성, 에러 로그 필터링, 릴리스 노트 작성 등 반복적인 품질 보증 및 문서화 작업을 줄인 결과다. 실제 프로덕션 환경은 실패 비용의 엄중함과 금융·공공 사이트 특유의 망분리 규제로 인해 AI의 자율적 복구보다는 철저한 인간의 통제 아래 운영되는 보조적 활용 상태에 머문다.

4.4.2 AI 도입의 장애 요인 및 직무역량 인식

조직이 직면한 최상위 도입 장애 요인은 기술적 불확실성과 환각 현상(33.0%)이며, 이 우려는 소스코드 정합성을 최종 책임져야 하는 중·고급 기술자 그룹에서 가장 높게 발현된다. 두 번째 장애 요인인 투자 대비 효율(ROI)의 불투명성(27.0%)은 고비용의 유료 도구 도입 대비 실질 성과를 정량적으로 계측할 조직 차원의 평가지표 체계가 부재한 현실을 투영한다. 특히 통계는 직급에 따른 위험 인식의 비대칭성을 명확히 드러내는데, 실무진이 기술적 환각과 미시적

오류에 집중하는 반면 18년 이상의 관리자 그룹은 기존 폭포수 방법론 및 거버넌스와의 거시적 충돌 가능성을 주요 우려 사항으로 지목한다. AI가 향후 직무 역량에 미칠 영향에 대한 인식은 고용 불안이나 직무 대체 가능성(7.5%)을 가장 낮게 평가한다. 대신 도구 의존 심화로 인한 엔지니어링 기본기 약화와 전문성 저하(32.0%)를 실재하는 위협으로 식별한다. 직급별 교차 분석에서도 관리자 그룹은 고부가가치 창출과 전사적 생산성 극대화에 초점을 맞추는 반면, 실무 개발자들은 기술 종속에 따른 개인 역량의 하향 평준화와 지속적 학습 압박을 경계한다. 이처럼 조직 리더와 현장 실무진 간의 상이한 리스크 인식 분포는 기술 도입 과정에서 발생하는 내부적 격차와 조직 관리적 과제를 정량적으로 보여준다.

HANSUNG UNIVERSITY

Ⅴ. 결론 및 제언

5.1 연구 결과 요약 및 시사점

5.1.1 연구 결과 요약

본 연구는 소프트웨어 개발 생명주기(SDLC)의 5단계를 기반으로 각 단계별 AI 기술의 활용 실태와 정량적·정성적 효과를 확인하고, 조직적·기술적 장애 요인을 규명하고자 실무자 200명의 설문 데이터와 3인의 전문 관리자 심층 인터뷰를 교차 분석했다. 도출된 구체적 연구 결과는 다음과 같다.

첫 번째, 소프트웨어 개발 전 주기에서 AI 도구의 '활용 수준'보다 실무진이 체감하는 '생산성 향상도'가 일관되게 상회하는 양(+)의 격차를 확인하였다. 특히 1단계 요구사항 분석 단계에서는 활용 수준(6.94점)과 생산성 향상도(7.31점) 간의 격차가 0.37점으로 전 구간 중 가장 크게 벌어졌다. 이는 초기 기획 단계에서 AI 도구를 상시적으로 붙잡고 작업하지 않더라도 회의록 요약이나 문서 초안 생성 등 특정 과업에 한정하여 개입시키는 행위 자체가 실무진에게 강력한 기저 효과를 제공함을 보여준다.

두 번째, 대중적 기대감이 가장 집중되었던 3단계 구현(코딩) 구간은 활용 수준(7.11점)과 생산성 향상도(7.24점)의 간극이 0.13점으로 좁게 수렴하는 독특한 한계를 노출했다. AI 코딩 어시스턴트가 물리적인 상용구(Boilerplate) 타이핑 시간은 획기적으로 덜어주지만, 생성된 코드의 논리적 무결성을 검증하고 기술적 환각 오류나 잠재적 보안 취약점을 디버깅하기 위해 개발자가 직접 부담해야 하는 '인지적 검증 비용'이 필연적으로 수반되는 업무 구조적 병목을 투영한 결과로 보인다.

세 번째, 후반부 단계인 테스트 및 검증(7.56점)과 배포 및 유지보수(7.83점) 영역은 정량 지표상 최고 수준의 성과를 기록했다. 다만 배포·유지보수 단계는 응답자가 30명으로 적어 단계 간 비교 시 표본 규모의 차이를 고려해야 한다. 그러나 심층 전문가 인터뷰 분석 결과, 이는 시스템 제어권의 완전한 자동 위임이 아니라 단위 테스트 스크립트 작성, 대량의 로그 필터링, 운영 문서 자동화

등 소모적인 주변부 과업을 효율화한 성격이 강하다. 실제 프로젝트 운영 환경은 실패 비용의 엄중함과 금융·공공 사이트 고유의 망분리 제약으로 인해 철저한 인간의 통제 아래 보조적 수단으로만 기능을 수행하고 있다.

네 번째, AI 도입을 가로막는 최상위 기술적 장애 요인은 '환각 현상에 따른 불확실성(33.0%)'이며, 이 우려는 소스코드의 정합성을 최종 책임지는 중·고급 기술자 그룹에 집중된다. 직무역량에 미칠 위협 인식의 경우 고용 불안(7.5%)보다는 '전문성 저하 및 도구 종속성 심화(32.0%)'를 실재하는 리스크로 식별했다. 특히 통계는 직급 간 위험 인식의 비대칭성을 드러냈는데, 실무진이 AI의 기술적 한계(미시적 오류와 기술적 환각)를 경계할 때 18년 이상 경력의 관리자 그룹은 기존의 룰과 프로세스의 충돌 가능성을 우려하는 양극화를 나타냈다.

5.1.2 시사점

본 연구의 데이터는 실무자의 주관적 체감에 기반한 기술 통계 분석 결과이므로 인과관계를 단정하기는 어렵다. 이러한 전제 아래 본 연구는 향후 AI 전환을 준비하는 소프트웨어 산업계에 세 가지 거시적 시사점을 제시한다.

첫 번째, AI 기술은 단순한 '코드 작성 보조기'에 머무르지 않고, 요구사항 명세 검증부터 최종 운영 로그 분석까지 SDLC 전 영역을 유기적으로 연결하는 '오케스트레이터(Orchestrator)'로 확장되고 있는 것으로 보인다. 따라서 기업은 특정 개발팀에 라이선스를 일괄 지급하는 미시적 도입 전략에서 벗어나, 전체 소프트웨어 개발 프로세스의 통합 관점에서 인프라를 재설계해야 한다.

두 번째, 단순히 코드를 많이, 빨리 작성한다고 해서 전체 개발 생산성이 높아진다는 과거의 착각에서 벗어나야 한다. 이제는 AI가 만들어낸 코드가 정확한지 '확인하고 수정하는 데 들어가는 시간과 노력'을 어떻게 효율적으로 관리하느냐가 성과 창출의 핵심이다.

이는 곧 미래 개발자에게 요구되는 핵심 역량이 프로그래밍 문법을 외워 코드를 빠르게 짜는 기술이 아님을 의미한다. 오히려 AI가 내놓은 결과물에서 숨은 오류를 날카롭게 찾아내고, 그 코드가 전체 시스템 구조와 충돌 없이 안전하게 작동할지 판단해 내는 '문제 해결 및 검증 능력'이 결국 조직 전체의 최종 생산성을 결정짓는다.

세 번째, 개발 조직 내 실무자와 관리자가 서로 다르게 느끼는 불안감을 조율하는 과정이 시급하다. 실무자는 AI에 의존하다 자신의 개발 역량이 떨어질 것을 불안해하고, 관리자는 기존의 업무 관리 체계가 흔들릴 것을 우려한다. 이 두 입장이 엇갈린 채 방치되면 신기술 도입은 조직의 강한 거부감에 부딪히게 된다.

이를 해결하려면 위아래가 원활하게 소통할 수 있는 구조를 만들어야 한다. 실무자에게는 AI의 오류를 안전하게 걸러낼 수 있는 공식적인 검증 규칙을 마련해 주고, 관리자에게는 AI 도입 효과를 명확한 숫자로 보여줄 수 있는 객관적인 평가 기준을 함께 제시해야만 신기술이 조직 내에 성공적으로 자리 잡을 수 있다.

5.2 정책 제언 및 연구의 한계

5.2.1 정책 제언

본 연구에서 관찰된 실무자 인식을 바탕으로 국내 소프트웨어 산업 발전을 위해 기업과 정부가 실행해야 할 세 가지 핵심 정책을 제언한다.

첫 번째, 기업 차원에서는 고부가가치 업무 집중 및 AI 코드 검증을 의무화해야 한다. 단순 코딩과 문서 작업은 AI에 맡기고, 개발자는 시스템 전체 설계나 보안 검토 같은 더 중요하고 어려운 업무에 집중하도록 업무 구조를 바꿔야 한다. 또한, AI에 무비판적으로 의존하다가 개발자 본연의 실력이 떨어지는 것을 막기 위해, AI가 짠 코드는 동료가 반드시 교차로 검증하도록 사내 규칙을 만들어야 한다.

두 번째, 정부 및 산업 차원에서는 개발자 등급 및 단가 산정 방식을 개편해야 한다. 현재의 개발자 임금(노임단가) 산정 방식은 '몇년 일했는가(경력)'에만 묶여 있다. 이래서는 AI를 활용해 혼자서 여러 명의 몫을 해내는 고효율 개발자의 가치를 제대로 보상해 줄 수 없다. AI를 잘 다루고 결과물의 오류를 잡아내는 능력을 객관적으로 평가하고 임금에 반영할 수 있는 '새로운 기술자 역량 평가 체계'가 필요하다.

세 번째, 정부 차원에서는 현실적인 보안 규제 완화 및 기업 내부용 AI 구축 지원이 이루어져야 한다. 공공이나 금융 분야는 강력한 보안 규제(망분리 등) 때문에 소스코드가 외부로 유출될까 봐 AI 도입을 꺼리고 있다. 이를 해결하려면 인터넷 연결 없이 기업 내부에 안전하게 자체 구축할 수 있는 AI(폐쇄형 언어모델)를 도입하도록 중소형 SI 기업에 보조금과 세제 혜택을 지원해야 한다. 동시에, 현실에 맞게 데이터 보안 규제를 풀어주는 한국형 가이드라인 제정이 시급하다.

5.2.2 연구의 한계와 후속 연구

본 연구는 현장 실무자 중심의 데이터를 통해 SDLC 단계별 AI 활용 실태를 파악했다는 의의가 있으나, 다음과 같은 네 가지 한계점을 지니며 이를 보완할 후속 연구를 제안한다.

첫 번째, 설문과 인터뷰 등 실무자의 주관적 인식에 의존했기 때문에 응답자의 성향이나 사내 문화에 따라 결과가 편향되었을 수 있다. 따라서 향후에는 AI 도입 전후의 실제 개발 소요 시간, 결함 발생률 등 측정 가능한 객관적 수치를 바탕으로 한 실증 연구가 뒷받침되어야 한다.

두 번째, 조직의 규모나 보안 및 인프라 환경이 미치는 영향을 세밀하게 통제하지 못했다. 향후 연구에서는 공공 및 금융 SI, 자체 솔루션 개발사, 인력 아웃소싱 기업 등 구체적인 산업 특성과 환경 조건을 구분하여 AI 도입 효과를 비교 분석하는 연구가 필요하다.

세 번째, 2026년 상반기를 기준으로 AI 기술의 활용을 분석했다는 시점의 한계가 있다. 향후 코딩부터 배포까지 스스로 수행하는 자율형 AI 에이전트가 보편화될 것으로 전망 되는 점을 고려하면, 후속 연구는 그 시기에 따른 제품들과 환경에 맞게 지속적으로 이루어져야 한다.

네 번째, 표본을 AI 도구를 이미 활용 중인 개발자로 한정하였다. 실제 사용하는 개발자들의 활용 수준과 생산성 향상도를 알아보기 위한 연구였기 때문이다. 이와 다른(AI도구를 사용하지 않는 사람이 포함된) 연구가 이루어진다면 도입 장벽과 효과를 보다 균형 있게 평가할 수 있을 것이다.

참 고 문 헌

1. 국내 문헌

방미영, 박성갑, 이한진, 이동수. (2026). 생성형 AI 코딩에 대한 IT 관리자와 개발자 간 인식 격차: 상호지향성(Co-orientation) 접근. 『한국컴퓨터정보학회논문지』, 31(2), 한국컴퓨터정보학회, 2026.2, 97-110.

최승윤, 이동건, 김준구, 장연주, 김현철. (2023). 프로그래밍 학습을 위한 LLM 기반 코드 리뷰 학습 환경 개발. 『컴퓨터교육학회 논문지』, 26(5), 컴퓨터교육학회, 2023.9, 1-11.

최종명. (2023). ChatGPT 기반 소프트웨어 요구공학. 『사물인터넷융복합논문지』, 9(6), 한국사물인터넷학회, 2023.12, 45-50.

황혜민, 이장원. (2025). AI 기반 짝 프로그래밍 기법을 활용한 프로그래밍 수업모델 설계. 『컴퓨터교육학회 논문지』, 28(1). 컴퓨터교육학회, 2025.2, 13-22.

황진석, 목일상, 차태준, 박영식. (2025). AI와 Vibe Coding을 활용한 Spec-Driven Development에 관한 연구. 『한국IT정책경영학회 논문지』, 17(3), 한국IT정책경영학회, 2025.9, 4007-4013.

정보통신기획평가원(IITP). (2024). 『2024년 인공지능산업 실태조사』.

TTA. (1998). 『소프트웨어 생명주기 공정 (TTAS.IS-12207)』. 『한국정보통신기술협회』.

한국인공지능·소프트웨어산업협회. (2025). 『2025년 적용 SW기술자 평균임금 공표』. 『한국인공지능·소프트웨어산업협회』.

SK플래닛. (2024). 『개발자 대상 GitHub Copilot 활용 실태조사』. 『SK플래닛 기술블로그.

CIO Korea. (2024). 『AI 코딩 얼리어답터' 기업 6곳이 전하는 생성형 AI 활용팁』. 『CIO Korea』.

2. 국외문헌

Alshahwan, N., Chheda, J., Finegenova, A., Gokkaya, B., Harman, M., Harper, I., Marginean, A., Sengupta, S., & Wang, E. (2024). Automated unit test improvement using large language models at Meta. In Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 185-196.

Boehm, B., & Basili, V. R. (2001). Software defect reduction top 10 list. Computer, 34(1), 135-137.

Borg, M., Hewett, D., Graham, D., Couderc, N., Söderberg, E., Church, L., & Farley, D. (2024). Does co-development with AI assistants lead to more maintainable code? A registered report. Lund University.

Butler, J., Suh, J., Haniyur, S., & Hadley, C. (2024). Dear diary: A randomized controlled trial of generative AI coding tools in the workplace. 319-329.

Cannizzaro, M., Chau, X., & Parrish, B. P. (2025). Systems engineering automation through artificial intelligence (AI) and natural language processing (NLP)-based software. INCOSE International Symposium, 35(1), 1109-1125.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., & Zaremba, W. (2021). Evaluating large language models trained on code.

Coutinho, M., Marques, L., Santos, A., Dahia, M., França, C., & Santos, R. (2024). The role of generative AI in software development productivity: A pilot case study. In Proceedings of the 1st ACM

International Conference on AI-Powered Software (AIware 2024). Association for Computing Machinery, New York, NY, USA, 131-138.

Dong, Y., Jiang, X., Qian, J., Wang, T., Zhang, K., Jin, Z., & Li, G. (2025). A survey on code generation with LLM-based agents.

Guglielmi, E., Arnoudova, V., Bavota, G., Oliveto, R., & Scalabrino, S. (2025). How do Copilot suggestions impact developers' frustration and productivity? In Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems.

Guimaraes, E., & Nascimento, N. (2025). AI in the software development lifecycle: Insights and open research questions. In Companion Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, 1353-1357.

Hymel, C. (2024). The AI-native software development lifecycle: A theoretical and practical new methodology.

Klemmer, J. H., Horstmann, S. A., Patnaik, N., et al. (2024). Using AI assistants in software development: A qualitative study on security practices and concerns. In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security.

Paradis, E., Grey, K., Madison, Q., Nam, D., Macvean, A., Meimand, V., Zhang, N., Ferrari-Church, B., & Chandra, S. (2024). How much does AI impact development speed? An enterprise-based randomized controlled trial. Google.

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot.

Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do users

write more insecure code with AI assistants? In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, 2785-2799.

Royce, W. W. (1970). Managing the development of large software systems. Proceedings of IEEE WESCON, 26(8), 1-9.

Schäfer, M., Nadi, S., Eghbali, A., & Tip, F. (2024). An empirical evaluation of using large language models for automated unit test generation. IEEE Transactions on Software Engineering, 50(1), 85-105.

Wang, Y., Guo, S., & Tan, C. W. (2025). From code generation to software testing: AI Copilot with context-based RAG. Nanyang Technological University.

Weber, T., Brandmaier, M., Schmidt, A., & Mayer, S. (2024). Significant productivity gains through programming with large language models. LMU Munich.

Weisz, J. D., Kumar, S., Muller, M., Browne, K.-E., Goldberg, A., Heintze, E., & Bajpai, S. (2025). Examining the use and impact of an AI code assistant on developer productivity and experience in the enterprise. In Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems.

IEEE. (1990). IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990). IEEE.

Project Management Institute. (2013). A guide to the project management body of knowledge (PMBOK guide) (5th ed.). Project Management Institute.

Washizaki, H. (Ed.). (2024). Guide to the software engineering body of

knowledge (SWEBOK Guide) (Version 4.0). IEEE Computer Society.

Bakal, G., Dasdan, A., Katz, Y., Kaufman, M., & Levin, G. (2025). Experience with GitHub Copilot for developer productivity at Zoominfo.

Becker, J., Rush, N., Barnes, E., & Rein, D. (2025). Measuring the impact of early-2025 AI on experienced open-source developer productivity

Dohmke, T., Iansiti, M., & Richards, G. (2023). Sea change in software development: Economic and productivity analysis of the AI-powered developer lifecycle.

EPAM Systems. (2024). AI in software development: State of adoption. EPAM Insights.

Gartner. (2024). Hype cycle for emerging technologies. Gartner Inc.

GitHub. (2024). GitHub Octoverse 2024: AI and the state of developer experience.

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (1956). A proposal for the Dartmouth summer research project on artificial intelligence.

McKinsey & Company. (2023). The economic potential of generative AI: The next productivity frontier. McKinsey Global Institute.

Stack Overflow. (2024). 2024 developer survey.

Amazon Web Services. (2024). AI-driven development life cycle: Reimagining software engineering. AWS DevOps & Developer Productivity Blog.

Amazon Web Services. (2024). AI-driven development life cycle:

Reimagining software engineering. AWS DevOps & Developer Productivity Blog.
https://aws.amazon.com/blogs/devops/ai-driven-development-life-cycle/

Anysphere. (2024). Cursor: The AI code editor.
https://cursor.com/

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., & Thomas, D. (2001). Manifesto for Agile software development.
https://agilemanifesto.org/

CircleCI. (2024). The new AI-driven SDLC. CircleCI Blog.
https://circleci.com/blog/ai-sdlc/

CloudQA. (2025). How LLMs are reshaping QA in 2025: GenAI & self-healing. CloudQA Blog.
https://cloudqa.io/blog/

Jackson, G., Kosinski, M., & Holdsworth, J. (2025, July 25). What is the software development lifecycle (SDLC)? IBM.
https://www.ibm.com/think/topics/sdlc

[부록] 설문지

[설 문 지]

HANSUNG UNIVERSITY

설 문 지

안녕하세요, 설문조사에 응해 주셔서 진심으로 고맙습니다.

이 설문조사는 '소프트웨어 개발 단계별 인공지능 기술 활용방안'을 연구하는 석사학위 논문 작성을 위한 기초자료 수집을 목적으로 진행하고 있습니다.

귀하의 설문 내용은 무기명으로 처리하고, 통계분석 목적으로만 사용합니다. 「통계법」 제33조에 따라 개인, 법인, 단체 등에 속하는 사항은 비밀을 보호하겠습니다.

2026년 5월

☐ 연구명 : 소프트웨어 개발 단계별 인공지능 기술 활용 방안 연구
☐ 소 속 : 한성대학교 지식서비스 & 컨설팅대학원
스마트융합컨설팅학과
☐ 연구자 : 양병헌 (지도교수 최동준)
☐ E-mail : yangbyounghun@gmail.com

귀하는 소프트웨어 개발 생애주기(SDLC: 기획, 설계, 구현, 테스트, 운영 등)에 직접 참여하는 직군입니까? (핵심 필터 질문)

① 예 (참여 직군): 개발자, PM/PMO, 서비스 기획자, 아키텍트, QA, DevOps 등

② 아니오 (미참여 직군): 일반 영업, 인사, 총무, 재무, 마케팅(비기술), 단순 관리 등 -> 설문조사 종료

Part 1 AI 기술 도입 및 일반 활용 현황

1. 현재 업무에서 AI 도구(LLM, 코딩 어시스턴트 등)를 얼마나 자주 사용하십니까?

① 매일 사용 ② 주 3~4회 ③ 주 1~2회 ④ 월 1~2회 ⑤거의 사용 안 함

2. 업무 전반에서 가장 주로 사용하는 AI 도구는 무엇입니까? (최대 2개)

① 범용 대화형 LLM (ChatGPT, Claude, Gemini 등) : 주로 로직 설계, 코드 스니펫 생성, 검색 용도

② IDE 확장형 코딩 어시스턴트 (GitHub Copilot, Amazon Q, JetBrains AI 등) : 기존 에디터 내 자동완성 위주

③ AI 네이티브 코드 에디터 (Cursor, Windsurf, Zed AI 등) : AI가 코드 전체 맥락을 이해하는 개발 환경

④ 분석 및 품질/보안 특화 도구 (DeepCode, CodeRabbit, Snyk AI 등) : 리뷰, 디버깅, 취약점 분석 특화

⑤ 자율형 에이전트 및 자동화 도구 (Devin, Claude Code, OpenDevin 등) : 특정 과업 전체를 자율 수행

3. 귀하의 AI 도구 활용 숙련도(Proficiency)는 어느 수준이라고 생각하십니까?

① 입문 - 기본적인 대화나 단순한 검색 용도로만 사용함

② **기초** - 간단한 코드 스니펫 생성 및 문서 요약 등에 활용함

③ **중급** - 프롬프트 엔지니어링을 통해 복잡한 로직 구현 및 업무 흐름에 적용함

④ **상급** - AI 도구의 특성을 이해하고 상황에 맞는 최적의 도구를 선택 및 조합하여 활용함

⑤ **전문가** - 사내 AI 도입을 주도하거나 커스텀 에이전트/API 연동 등 고도의 자동화를 구현함

4. 귀하의 소속 조직은 AI 도구 활용에 대해 어떤 환경을 제공하고 있습니까?

① **개인적 활용** : 조직 차원의 지원이나 정책 없이 개인 계정으로만 사용함

② **제한적 허용** : 보안 규정 내에서 일부 무료 도구 사용만 허용함

③ **유료 계정 지원** : 조직에서 업무용 유료 계정(ChatGPT Plus, Copilot 등) 비용을 지원함

④ **전용 인프라 구축** : 사내 전용 AI 포털이나 자체 구축 LLM(sLLM) 환경을 제공함

⑤ **강력한 권장/필수** - AI 사용을 표준 프로세스로 지정하고 정기적인 교육 및 가이드를 제공함

5. AI 도구를 업무에 도입하게 된 가장 주된 계기(Motivation)는 무엇입니까?

① **개인적 호기심**: 최신 기술 트렌드에 대한 관심과 자기 계발을 위해

② **업무 과부하 해결**: 과도한 업무량과 반복 작업 시간을 줄이기 위해

③ **기술적 한계 극복** : 본인의 지식만으로 해결하기 어려운 기술적 난제를 해결하기 위해

④ **조직의 권고/지시** : 회사나 팀 차원의 도입 방침에 따라 사용하게 됨

⑤ **동료/업계 트렌드** : 동료들의 사용 사례나 업계의 보편적 흐름에 뒤처지지 않기 위해

Part 2 SDLC 단계별 AI 도구 활용 및 정량적 효과 (심화)

본 파트는 논문의 핵심인 SDLC 5단계를 기준으로 합니다.
각 단계에서 활용 경험이 있는 경우에만 답변해 주십시오.

다음의 SDLC 개발 5단계 중 본인이 경험하신 단계를 모두 체크해 주세요
(해당 단계로 이동)

☐ 1단계: 요구사항 분석 (유스케이스 도출, 요구사항 명세서 작성 등)

☐ 2단계: 시스템 설계(아키텍처 추천, DB 스키마 설계, 다이어그램 생성 등)

☐ 3단계: 구현/코딩 (코드 자동완성, 로직 생성, 리팩터링 등)

☐ 4단계: 테스트 및 검증 (테스트 케이스 생성, 버그 탐지, 품질 분석 등)

☐ 5단계: 배포 및 유지보수(장애 분석, 로그 모니터링, 릴리스 노트 작성 등)

▣ 1단계: 요구사항 분석 (유스케이스 도출, 요구사항 명세서 작성 등)

6. 주요 활용 도구(활용 빈도에 따라 순서대로 기재)

① **범용 대화형 LLM (ChatGPT, Claude, Gemini 등)** : 요구사항 초안 작성 및 아이디어 도출

② **회의/음성 분석 AI (클로바노트, Otter.ai, MS Copilot 등)** : 고객/이해관계자 미팅 녹취 기반 요구사항 추출

③ **문서 및 이슈 관리 내장 AI (Notion AI, Atlassian Intelligence(Jira/Confluence) 등)** : 명세서 정형화 및 에픽/스토리 티켓 생성

④ **다이어그램 및 시각화 AI (Eraser.io, Mermaid AI, Whimsical 등)** : 유스케이스 다이어그램 및 워크플로우 시각화

⑤ **기타** (직접 입력: _____)

⑥ AI 도구를 활용하지 않음

7. 주요 효과 (최대 2개 선택)

① **요구사항 구체화 및 초안 작성 시간 단축**: 아이디어 브레인스토밍부터 요구사항 목록(Backlog) 초안 생성까지의 물리적 시간 절감

② **요구사항 간 논리적 모순 및 누락 식별**: 텍스트 간의 충돌을 탐지하거나 정의되지 않은 예외 케이스(Edge case)를 발견하여 분석 품질 향상

③ **표준화된 산출물(문서) 자동 생성**: 요구사항 명세서(SRS), 사용자 스토리, 유스케이스 명세서 등을 정해진 템플릿에 맞춰 자동 구조화

④ **비즈니스** : 기술 언어 간의 상호 번역: 모호한 비즈니스 요구사항을 개발자가 이해 가능한 기술적 상세 규격이나 데이터 모델로 변환 용이

⑤ **이해관계자 간 의사결정 및 합의 지원**: 유스케이스 시각화나 프로토타입 시나리오 생성을 통해 고객/PM/개발자 간의 명확한 의도 파악 및 소통 강화

8. 정량적 성과: 기존 방식 대비 투입 시간/비용이 약 몇 % 단축되었습니까?

① 단축 효과 없음 또는 오히려 증가 (0% 이하)

② 20% 미만 ③ 20~40% ④ 40~60% ⑤ 60% 이상

▣ 요구사항 분석단계에서 활용수준을 숫자로 표기해 주세요

전혀 활용 안함	1	2	3	4	5	6	7	8	9	10	전면 도입

▣ 요구사항 분석단계에서 생산성 향상도를 숫자로 표기해 주세요

변화 없음	1	2	3	4	5	6	7	8	9	10	혁신적 도입

▣ 2단계: 시스템 설계(아키텍처 추천, DB 스키마 설계, 다이어그램 생성

등)

9. 활용 도구

① 범용 대화형 LLM (ChatGPT, Claude, Gemini 등): 아키텍처 패턴 비교, API 스펙(OpenAPI/Swagger) 및 스키마 구조 설계

② AI 기반 다이어그램/시각화 도구 (Eraser.io, Mermaid AI, Draw.io AI 등) : 텍스트/코드 기반 시스템 구성도, 시퀀스/클래스 다이어그램 자동 생성

③ 클라우드 및 인프라 설계 특화 AI (Amazon Q, Google Cloud Gemini, 인프라스트럭처 애즈 코드(IaC) 보조 AI 등) : 클라우드 아키텍처 추천 및 프로비저닝 스크립트 설계 :

④ 데이터베이스/ERD 특화 AI (AI 지원 ERD 툴, SQLDBM, DataGrip AI 등) : DB 정규화 검토, 스키마 모델링 및 인덱스 설계 보조

⑤ 기타 (직접 입력: _____)

⑥ AI 도구를 활용하지 않음

10. 주요 효과(1개 선택)

① 초기 설계 및 산출물 작성 시간 단축: API 명세서, DB 스키마 보일러플레이트(기본 구조) 생성 및 설계 문서화 속도 향상

② 최적의 아키텍처 및 기술 대안 비교: 모놀리식 vs MSA, RDBMS vs NoSQL 등 요구사항에 적합한 디자인 패턴 및 기술 스택 결정 지원

③ 다이어그램 등 시각적 구조화 자동화: 복잡한 비즈니스 로직이나 시스템 컴포넌트 간의 상호작용을 다이어그램으로 즉각 변환하여 가시성 확보

④ 구조적 결함 및 보안 병목 사전 식별: 설계 단계에서 단일 장애점(SPOF), 트랜잭션 충돌 가능성, 데이터 무결성 위협 등을 사전 검토

⑤ 설계 표준화 및 일관성 유지: API 네이밍 컨벤션, 데이터 타입, 인터페이스 규격 등을 팀 또는 업계 표준에 맞게 통일

11. 정량적 성과: 설계 단계에서의 오류 발견율이나 작업 속도가 약 몇 % 개선되었습니까?

① 단축 효과 없음 또는 오히려 증가 (0% 이하)

② 20% 미만

③ 20~40%

④ 40~60%

⑤ 60% 이상

▣ 시스템 설계 단계에서 활용수준을 숫자로 표기해 주세요

전혀 활용 안함	1	2	3	4	5	6	7	8	9	10	전면 도입

▣ 시스템 설계 단계에서 생산성 향상도를 숫자로 표기해 주세요

변화 없음	1	2	3	4	5	6	7	8	9	10	혁신적 도입

▣ 3단계: 구현/코딩 (코드 자동완성, 로직 생성, 리팩터링 등)

12. 활용 도구

① **범용 대화형 LLM (ChatGPT, Claude, Gemini 등)** : 외부 웹 브라우저에서 로직 질의응답 및 코드 스니펫 복사/붙여넣기

② **IDE 확장 플러그인 (GitHub Copilot, JetBrains AI, Amazon Q 등)** : 기존 에디터 내에서 실시간 한 줄/함수 단위 자동완성

③ **AI 네이티브 에디터 (Cursor, Windsurf 등)** : 프로젝트 전체(Workspace) 맥락을 이해하고 다중 파일 단위로 코드를 수정/생성

④ **특화 생성기 및 자율형 에이전트 (v0.dev, bolt.new, Aider, Devin 등)** : 프롬프트만으로 UI 컴포넌트나 전체 모듈을 자율적으로 구현

⑤ 기타 (직접 입력: _____)

⑥ AI 도구를 활용하지 않음

13. 주요 효과(1개 선택)

① **상용구(Boilerplate) 및 반복 코드 자동 생성**: 단순 타이핑을 줄여 물리적인 코딩(구현) 시간 단축

② **복잡한 로직 및 알고리즘 구현 보조**: 정규표현식, 복잡한 쿼리문, 특정 자료구조 알고리즘 등 난도 높은 코드 작성 지원

③ **레거시 코드 해석 및 주석 자동화**: 타인이 작성했거나 오래된 기존 코드의 의도를 파악하고 문서화/주석 작성 시간 절감

④ **코드 가독성 향상 및 리팩터링**: 클린 코드(Clean Code) 원칙이나 디자인 패턴을 적용하여 기존 코드 구조 최적화

⑤ **신규 기술 및 프레임워크 학습 곡선(Learning Curve) 단축**: 익숙하지 않은 언어나 API 사용 시 실시간 예제와 가이드를 제공받아 학습 및 적용 시간 단축

14. 정량적 성과:기존 방식 대비 코딩 완료 속도가 약 몇 % 향상되었습니까?

① 단축 효과 없음 또는 오히려 증가 (0% 이하)

② 20% 미만 ③ 20~40% ④ 40~60% ⑤ 60% 이상

▣ 구현 단계에서 활용수준을 숫자로 표기해 주세요

전혀 활용 안함	1	2	3	4	5	6	7	8	9	10	전면 도입

▣ 구현 단계에서 생산성 향상도를 숫자로 표기해 주세요

변화 없음	1	2	3	4	5	6	7	8	9	10	혁신적 도입

▣ 4단계: 테스트 및 검증 (테스트 케이스 생성, 버그 탐지, 품질 분석 등)

15. 활용 도구

① **범용 대화형 LLM (ChatGPT, Claude, Gemini 등)** : 테스트 시나리오 초안 작성 및 테스트 데이터(Mock Data) 생성

② **유닛 테스트 자동 생성 도구 (Diffblue Cover, TestGen-LLM, GitHub Copilot 등)** : 기존 코드 분석을 통한 단위 테스트 코드 및 보일러플레이트 자동 생성

③ **자율형 E2E 및 QA 에이전트 (mabl, Sauce Labs AI, Blinq.io 등)** : 사용자 시나리오 기반의 종단 간(E2E) 테스트 자동화 및 UI 변경 대응

④ **AI 기반 정적 분석 및 버그 탐지 도구 (CodeRabbit, Qodo Merge, Snyk, SonarQube AI 등)** : PR 단계에서의 자동 코드 리뷰, 보안 취약점 및 런타임 버그 사전 탐지

⑤ 기타 (직접 입력: _____)

⑥ AI 도구를 활용하지 않음

16. 주요 효과(1개 선택)

① **테스트 커버리지 확대 및 엣지 케이스 발견**: 수동으로 확인하기 어려운 예외 상황(Edge case)을 탐지하여 미커버 영역 최소화

② **테스트 스크립트 작성 및 유지보수 시간 단축** : 테스트 코드 작성 속도 향상 및 UI 변경 시 테스트가 자동 수정되는 '자기 치유' 기능으로 관리 효율화

③ **결함 탐지 정확도 향상 및 근본 원인 분석(RCA) 지원** : 단순 버그 발견을 넘어 로그 분석을 통한 에러 발생 원인과 수정 방안 제시

④ **테스트 시나리오의 다양성 및 현실성 확보** : 실제 사용자 행동 패턴을 모사한 다양한 테스트 시나리오와 대량의 합성 데이터 생성

⑤ **개발-QA 협업 효율성 및 배포 신뢰도 향상** : 코드 리뷰 자동화와 신속한 피드백 루프를 통해 결함 수정 시간 단축 및 안정적인 배포 지원

17. 정량적 성과:테스트 커버리지가 이전 대비 약 몇 % 향상되었습니까?

① 단축 효과 없음 또는 오히려 증가 (0% 이하)

② 20% 미만

③ 20~40%

④ 40~60%

⑤ 60% 이상

▣ 테스트 및 검증 단계에서의 활용 수준을 숫자로 표기해 주세요

전혀 활용 안함	1	2	3	4	5	6	7	8	9	10	전면 도입

▣ 테스트 및 검증 단계에서 생산성 향상도를 숫자로 표기해 주세요

변화 없음	1	2	3	4	5	6	7	8	9	10	혁신적 도입

▣ 5단계: 배포 및 유지보수(장애 분석, 로그 모니터링, 릴리스 노트 작성 등)

18. 활용 도구

① **지능형 통합 모니터링/AIOps 플랫폼 (Datadog Bits, Dynatrace Davis, New Relic AI 등)** : 실시간 지표 분석 및 이상 징후 탐지

② **로그 분석 및 패턴 인식 특화 AI (Splunk AI, Elastic AI, Sumo Logic**

등) : 대량의 비정형 로그 데이터 필터링 및 에러 패턴 식별

③ 인시던트 대응 및 협업 보조 AI (PagerDuty AI, Slack AI, MS Teams Copilot 등) : 장애 상황 요약, 담당자 할당 및 유사 장애 이력 검색

④ 릴리스 및 운영 문서화 AI (ChatGPT/Claude, Jasper, 전용 릴리스 노트 생성기 등) : 코드 변경점 기반의 릴리스 노트, 기술 매뉴얼 및 장애 보고서(Post-mortem) 작성

⑤ 기타 (직접 입력: _____)

⑥ AI 도구를 활용하지 않음

19. 주요 효과(1개 선택)

① 장애 복구 시간(MTTR) 단축 및 가용성 향상: 인시던트 발생 시 실시간 알림과 빠른 원인 분석을 통해 서비스 중단 시간 최소화

② 근본 원인 분석(RCA)의 자동화 및 정확도 향상 : 복잡한 마이크로서비스(MSA) 환경에서 장애의 시작점과 연쇄 파급 효과를 지능적으로 추적

③ 운영 문서화 및 보고 업무의 효율화 : 릴리스 노트, 운영 가이드, 고객 대응 FAQ 등 반복적인 문서 작성 공수 절감

④ 선제적 모니터링을 통한 장애 예방 : 과거 데이터를 기반으로 시스템 임계치 초과나 성능 저하 징후를 사전에 탐지하여 대응

⑤ 운영 데이터 기반의 의사결정 지원 : 로그 및 성능 데이터를 분석하여 인프라 증설, 비용 최적화, 코드 수정 우선순위 도출

20. 정량적 성과:장애 대응 시간이나 유지보수 공수가 약 몇 % 절감되었습니까?

① 단축 효과 없음 또는 오히려 증가 (0% 이하)

② 20% 미만

③ 20~40%

④ 40~60%

⑤ 60% 이상

▣ 배포 및 유지보수 단계에서 활용수준을 숫자로 표기해 주세요

전혀 활용 안함	1	2	3	4	5	6	7	8	9	10	전면 도입

▣ 배포 및 유지보수 단계에서 생산성 향상도를 숫자로 표기해 주세요

변화 없음	1	2	3	4	5	6	7	8	9	10	혁신적 도입

Part 3 AI 도입의 장애 요인 및 인식 (이성적 평가)

21. AI 도입 시 가장 크게 우려되는 기술적/조직적 문제는 무엇입니까? (1순위 선택)

① **보안 및 지식재산권 유출** : 코드 외부 유출 및 데이터 보안 정책과의 충돌

② **기술적 불확실성 및 환각(Hallucination)** : 결과물의 정확성 불신 및 검증 비용 증가

③ **투자 대비 효율(ROI)의 불투명성** : 도구 도입 비용 및 유지보수 대비 실질적 성과 측정 어려움

④ **사내 AI 숙련도 부족 및 교육 부재** :구성원의 AI 리터러시 격차로 인한 도입 속도 저하

⑤ **기존 개발 프로세스(SDLC) 및 조직 문화와의 부조화** : 도구 도입에 따른 업무 흐름의 변화 및 조직적 저항

22. AI 기술이 향후 귀하의 직무역량에 어떤 영향을 미칠 것이라 보십니까? (1순위 선택)

① **업무 생산성 극대화** : 단순 반복 업무 자동화로 설계, 아키텍처 등 고부가가치 업무에 집중

② **직무 역할의 전환** : 코드 작성자(Coder)에서 AI를 관리・조정하는

오케스트레이터(Orchestrator)로 변화

③ **전문성 저하 및 도구 의존** : AI 의존도 심화로 인한 문제 해결 능력 및 기초 기술 역량 약화

④ **지속적 학습에 대한 심리적 압박** : 급변하는 AI 기술 습득을 위한 인지적 과부하 및 번아웃 우려

⑤ **직무 대체 가능성 및 고용 불안** : AI의 수행 범위 확대로 인한 인간 개발자의 역할 축소 및 경쟁 심화

Part 4 응답자 기본 정보 (Demographics)

분석의 객관성을 위해 응답자의 소속과 배경을 수집합니다.

23. 귀하의 총 업무 경력(IT 분야)은 어느 정도입니까?(본인의 학력, 자격증에 맞게 기재해 주세요)

① 초급: 3년 미만

② 중급: 3년 이상 ~ 7년 미만

③ 고급: 7년 이상 ~ 12년 미만

④ 특급: 12년 이상 ~ 18년 미만

⑤ 관리자/전문가 수준: 18년 이상

24. 소속 조직의 규모는 어느 정도입니까?

① 10인 미만 ② 10~100인 ③ 100~500인 ④ 500~1,000인 ⑤ 1,000인 이상

25. 귀하의 주요 직무(Role)는 무엇입니까?

① **프로젝트 관리 및 기획 (PM/PMO/기획)** : 요구사항 분석, 일정 관리, 거버넌스 수립 주도

② **시스템 설계 및 리딩 (Architect/PL)** : 아키텍처 설계, DB 모델링, 파트 리딩 및 기술 결정

③ **어플리케이션 개발 (Backend/Frontend Developer)** : 코어 로직 구현, API 개발 및 UI/UX 구현

④ **품질 관리 및 테스트 (QA/Tester/SDET)** : 테스트 케이스 작성, 검증 자동화 및 품질 분석

⑤ **인프라 및 운영 (DevOps/SRE/Cloud Engineer)** : 배포 파이프라인 구축, 인프라 설계 및 유지보수

- 설문에 응해 주셔서 대단히 고맙습니다 -

HANSUNG UNIVERSITY

ABSTRACT

AI Utilization Strategies across the Phases of the Software Development Life Cycle (SDLC)

Yang, Byoung-Hun

Major in Smart Convergence Consulting

Dept. of Smart Convergence Consulting

Graduate School of Knowledge Service Consulting

Hansung University

Since OpenAI released ChatGPT, a large language model (LLM)-based service, in November 2022, generative artificial intelligence has been rapidly adopted in the workplace. In its early stages, limitations such as the training-data cutoff and the phenomenon of hallucination led many to expect that considerable time would pass before the technology could be used in practice. However, as new models were released in succession, both the user base and the scope of application expanded. Software development has been no exception: developers who use generative AI for writing source code and documentation are now a common sight. Yet empirical research on how this technology is used at each stage of development and how it affects productivity in the field remains insufficient.

This study analyzes the actual use and productivity effects of generative AI across the entire Software Development Life Cycle (SDLC), from requirements analysis to deployment and maintenance. First, a literature review was conducted to organize the relationship between the level of AI use and productivity at each stage. Next, a survey of 200 practicing developers who use generative AI in real projects was carried out to measure and compare the level of use and the perceived productivity improvement at each stage. Finally, three field experts at the project-manager (PM) level or above reviewed the survey results, thereby cross-validating (triangulating) the quantitative analysis with practitioners' interpretations of the field.

The analysis found that, across all SDLC stages, the perceived productivity improvement reported by developers consistently exceeded the level of tool use. The size of this gap, however, varied by stage. In the requirements analysis stage, although the level of use was relatively low, the gap between the two indicators was the largest at 0.37 points, indicating that a substantial effect was perceived from use alone. By contrast, the implementation and testing stages showed the narrowest gaps, at 0.13 and 0.09 points respectively, which is presumably because the cost of verifying and correcting AI-generated code offsetting the benefits. Among the barriers to adoption, technical uncertainty arising from hallucination (33.0%) and the opacity of return on investment (27.0%) were most frequently cited, while concerns about the erosion of professional expertise and over-reliance on tools were prominent at the job-competency level.

This study is significant in that it divides the SDLC into stages and compares the level of use and the productivity effects of generative AI across them. It is hoped that these findings will serve as a reference for practitioners who commission, perform, and supervise projects in

managing them on the basis of generative AI.

【Keywords】 Software Development Life Cycle (SDLC), Generative AI, Development Productivity, Barriers to AI Adoption, LLM Coding Tools

HANSUNG UNIVERSITY