제 27 권 제 1 호
Vol. 27 No. 1

2025
제 27 회
한국 소프트웨어공학 학술대회 논문집

Proceedings of the 27th Korea Conference on Software Engineering (KCSE 2025)

- 일시: 2025년 1월 20일(월) ~ 1월 22일(수)
- 장소: 강원도 평창 한화리조트(휘닉스파크점)

주최: 한국정보과학회, 한국정보처리학회
주관: 한국정보과학회 소프트웨어공학 소사이어티
한국정보처리학회 소프트웨어공학연구회

후원: SOLUTIONLINK
㈜비트컴퓨터
㈜포멀웍스, ㈜다한테크, ㈜모아소프트
브이플러스랩㈜, 슈어소프트테크㈜, ㈜이에스지(ESG)
한국정보통신기술협회(TTA)

세션 E1. 요구사항 분석

- 2025년 1월 22일 (수) 오전 10:40-11:50 / 그랜드홀 2
- 좌장: 이선아 교수(경상대)

드론 충돌 회피에 대한 초기 안전성 분석 기법에서의 GPT-4o 활용 연구
김태환, 이정원, 이선아 (경상국립대)

건전성 테스팅을 위한 공간 및 운용 요구사항 분석 및 테스트 프로그램 설계 방법
양희찬, 최민서, 김진세, 이정원 (아주대)

Software Process 내에서 AI 지향 요구 공학 적용한 사례 (단편논문)
진예진, 서채연 (홍익대), 공지훈 (툰스퀘어), 김기두 (한국정보통신기술협회), 김영철 (홍익대)

복잡한 문맥 개선을 위한 C3Tree 모델 기반 F1-Score 개선 사례 (단편논문)
김장환, 양진모 (홍익대), 장우성 (넥스트솔), 김영철 (홍익대)

[초청발표] PBE-based Selective Abstraction and Refinement for Efficient Property Falsification of Embedded Software (FSE 2024), 김요엘, 최윤자(경북대), The ACM International Conference on the Foundations of Software Engineering (FSE 2024) 국제학술대회 발표 논문
발표자: 김요엘 (경북대)

세션 E2. 지능형 SW 품질

- 2025년 1월 22일 (수) 오전 10:40-11:50 / 세미나실1
- 좌장: 백종문 교수(KAIST)

데이터 포맷에 따른 RAG 성능 비교와 LLM 기반 평가 체계의 실증적 분석 (단편논문)
김민지 (펀진), 강승식 (국민대)

문맥 상호 연관 가중치의 중요성 : 구조화된 데이터에서 RAG 시스템의 환각 현상 완화를 위한 최적화 연구 (산업체논문)
이재문, 김예은, 김현지((주) 시즌 연구팀)

ISO/IEC TS 4213 AI 성능 측정 표준을 이용한 비교 숙련도 프로그램 개발에 관한 연구 (단편논문)
황현후, 김광훈, 김한결 (한국정보통신기술협회)

도메인 특화 LLM 모델의 프롬프트 최적화를 위한 에이전트형 프롬프트 (단편논문)
이지웅, 이철웅 (고려대)

딥페이크 탐지를 위한 지식 증류 기반 경량화 모델 (학부생논문)
이지수, 김지우, 김준화 (건양대)

Software Process 내에서 AI 지향 요구 공학 적용한 사례

진예진 [1], 서채연 [2], 공지훈 [3], 김기두 [4], 김영철 [5]
홍익대학교 소프트웨어공학연구실 [1,2,5], 툰스퀘어 [3], 한국정보통신기술협회 [4]
yejin_jin@g.hongik.ac.kr[1], chaeyun@hongik.ac.kr[2], john.tooning@toonsquare.com[3],
kdkim@tta.or.kr[4], bob@hongik.ac.kr[5]

Best Practices on AI driven Requirement Engineering in Software Development Life Cycle

Yejin Jin[1], Chaeyun Seo[2], Jihoon Kong[3], Kidu Kim[4], R. Young Chul Kim[5]
Software Engineering Laboratory, Hongik University[1,2,5]
Toonsquare[3], Telecommunication Technology Association[4]

요약

기존 요구공학에서는 고객과 개발자들이 정형/비정형 요구사항을 정의 및 분석해야 하는 한계가 있다. 최근 다양한 AI 기반 도구는 블랙박스 기반으로 자연어 분석 및 코드 생성을 수행하려 한다. 그러나, 그들의 결과물(코드, 테스트케이스)에 대해 오류, 품질, 신뢰성 등을 보장 및 검증이 불가능하다. 이를 해결하기 위해 소프트웨어 개발 생명주기 내에 단계적으로 AI 접목이 필요하다. 이를 위해 AI Driven Requirement Engineering process를 제안한다. 1) 요구공학에서 자연어 요구사항을 체계적으로 분석하기 위해 두 가지의 언어학적 분석을 접목한다. 즉, 자연어 요구사항의 문법적 분석을 위한 Chomsky의 언어학과 의미론적 분석을 위한 Fillmore의 언어학을 적용한다. 2) GPT 기반 자연어 분석 프로세스 통해 UML을 생성하고, 3) 이를 바탕으로 코드를 생성한다. 제안한 메커니즘을 위해서 간단한 제어 시스템을 사례로 설명한다.

Abstract

In the traditional requirements engineering, there is a limitation that customers and developers must define and analyze formal/informal requirements. Recently, various AI-based tools try to perform natural language analysis and code generation based on a black box. However, it is impossible to guarantee and verify errors, quality, reliability, etc. for their results (code, test cases). To solve this problem, AI needs to be gradually incorporated into the software development life cycle. For this purpose, we propose an AI Driven Requirement Engineering process. 1) To systematically analyze natural language requirements in requirements engineering, two linguistic analyses are incorporated. That is, we apply Chomsky's linguistics for grammatical analysis of natural language requirements and Fillmore's linguistics for semantic analysis. 2) UML is generated through a GPT-based natural language analysis process, and 3) code is generated based on this. For our proposed mechanism, we explain one example of a simple control system.

1. 서론

소프트웨어 개발 생명 주기는 고품질 소프트웨어를 제작하기 위해 시스템 요구사항 분석부터 유지보수까지의 모든 과정을 체계화한다[1]. 과거에는 각 단계를 자동화하는 도구들이 개발되었고, 수동 작업 대비 높은 생산성을 제공하였다. 그러나 기존 자동화 도구들은 정해진 규칙 내에서만 작동하기 때문에 다양한 상황에 대한 처리 능력이 부족하다. 최근에는 대규모 데이터로 학습된 인공지능은 규칙 기반의 자동화 도구의 한계를 극복하여 다양한 상황에 적응할 수 있다. 특히, Large Language Model(LLM)은 다양한 분야에서 접목되고 있으며 소프트웨어 공학 분야에서는 요구사항 관리, 프로젝트 예측 등으로 활용되고 있다. 이러한 도구들은 자연어 입력을 분석하여 소프트웨어 설계 및 구현 단계의 작업을 자동화할 수 있다. 하지만, 우리는 그 과정을 알 수 없어 AI 도구의 결과를 신뢰할 수 없다[2].

따라서 우리는 소프트웨어 개발 생명 주기의 단계적 절차를 기반으로 생성형 AI를 적용하여 더욱 체계적이고 일관되게 개발하고자 한다. 이를 위해, 자연어 분석 단계에서 두가지의 언어학을 적용한다. 1) Chomsky의 이론으로 문장의 구조 및 형태소 분석을 진행하고, 2) Fillmore의 이론을 통해 각 형태소에 대한 의미적 분석을 진행한다.

자연어 분석 후, 자연어 요소와 UML 요소를 매핑하여 UML 다이어그램 설계를 생성한다. 마지막으로 설계를 바탕으로 메타 모델링을 통해 다양한 언어의 코드를 생성한다. 본 연구에서는 생산성이 높은 AI 도구와 소프트웨어 개발 방법론의 단계적 프로세스를 결합한 방식을 제안한다.

2장에서는 관련된 연구로 소프트웨어공학에서 사용되는 생성형 AI 도구에 대해 설명한다. 3장에서는 생성형 AI와 SW 공학 접목에 대한 메커니즘을 제안한다. 4장에서는 연구의 결과와 한계점, 그리고 향후 연구를 언급한다.

2. Text-to-Code AI 도구

자연어 입력으로부터 코드를 제공하는 생성형 AI 모델이 출시되고 있다. 이들은 코딩 능력에 특화되어 다양한 언어를 출력할 수 있다.

- **Copilot**: OpenAI에서 출시된 GitHub 연동이 가능한 AI 코딩 도구이다. 그러나, 실제 GitHub의 프로젝트에서 Copilot이 생성된 코드 조각 중 약 29.6%에서 보안 취약점이 발견되었으며, 이를 고려한 대응책 마련이 필요함이 강조된다[3].
- **Codeium**: 코드 자동 완성이나 리팩토링 기능을 제공하는 AI 코딩 도구이다. 쉬운 문제에서는 효과적이지만, 어려운 문제에 대해서는 오답이나 시간 초과 오류를 발생한다[4].
- **Codewhisperer**: Amazon에서 개발한 AI 코딩 도구로, 통합 개발 환경에서 실시간으로 주석을 분석하여 코드를 제안한다. 그러나, 생성한 코드의 평균 정확성은 51.95%로 지속적인 발전이 필요하다[5].

AI가 생성한 코드에 대한 신뢰성이 문제이다. 우리는 내부 작동 방식을 알 수 없기 때문에 절차적인 방식에 따라 각 부분 별로 AI를 적용하여 결과를 확인해야 한다. 본 연구에서는 개발방법론에서의 AI 사용에 대한 새로운 시각을 제공한다.

3. GPT API를 적용한 자연어로부터 코드 발생

자연어 요구사항으로부터 코드를 발생시키기 위해 우리는 요구사항 분석, 설계, 구현 과정을 거친다. 자연어 요구사항에 대한 체계적인 분석을 자동화하기 위해 해당 과정에 GPT API를 사용한다. 그림 1은 본 연구의 전체 프로세스를 보여준다.

요구사항 분석 단계는 세가지로 이루어진다. 문장을 전처리하고, 이후 구조 분석에는 Chomsky의 Syntactic Structure Analysis 이론을 사용하고, 의미 분석에는 Fillmore의 Semantic Roles 이론을 사용한다. Chomsky와 Fillmore의 언어학을 사용하여 체계적인 자연어 분석을 진행한다. 분석된 문장은 UML 다이어그램으로 나타날 수 있고, 각 다이어그램의 요소들은 코드의 정보들과 매핑된다. 우리는 스마트 빌딩 환경 제어 시스템 사례를 들어 체계적인 자연어 분석 과정을 보여준다. 그림 2는 자연어 요구사항이다.

3.1 자연어 요구사항 분석

우리는 자연어 요구사항을 분석하기 위해 복잡한 문장과 단순한 문장을 구분한다. 여러 개의 동사나 주어로 이루어진 문장은 복잡한 문장으로 처리하고, 하나의 동사와 주어를 가진 단문을 단순한 문장이다. 자연어 전처리 과정을 거친다. 전처리 과정에서 우리는 해당 단계에 GPT API를 사용하여 다음과 같이 프롬프팅을 진행한다.

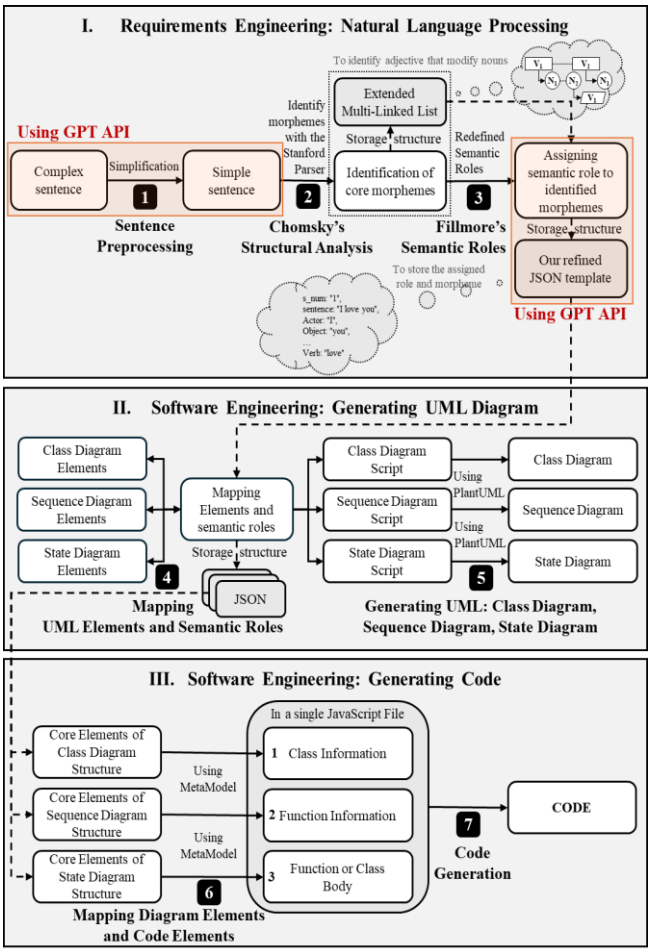

그림 1. 자연어로부터 코드 발생 프로세스

Natural Language Requirement Specification

1. The Smart Building Environment Control System has three functions: ON, OFF, Warning.
2. The manager and user has an ID and a PIN number.
3. The air conditioner has a temperature sensor.
4. The control system can manage the air conditioners.
5. The manager access the control system using ID and PIN number.
6. The manager enter the temperature into the control system.
7. When the temperature sensor measures 30 °C, the control system turns on.
8. Users can turn off their own control system.
9. When the temperature sensor measures 15 °C, the control system turns off.
10. When the stranger gains access to the control system using a virus, the control system's warn system is activated.

ON UseCase
OFF UseCase
Warn UseCase

그림 2. 스마트 빌딩 환경 제어 시스템 요구사항

1) **복문과 중문, 단문의 특징을 제공한다.**
2) **문장의 특징에 따라 단문화 하는 방법을 설명한다.**
3) **대명사를 없애고, 문장 내의 적절한 명사로 변경한다.**
4) **누락된 문장 요소(주어, 동사, 목적어)는 기존 문장의 것으로 사용한다.**
5) **자연어 요구사항 텍스트 파일을 읽어 프롬프트를 수행하고, 하나의 리스트로 저장한다.**

요구사항 7번을 예시로, 전처리 결과를 그림 3으로 나타낸다.

[R7] When the temperature sensor measures 15 °C, the control system turns off.
→ [R7-1] The temperature sensor measures 15 °C.
→ [R7-2] The control system turns off.

그림 3. 자연어 요구사항 전처리

저장된 자연어 요구사항 리스트는 Chomsky의 구문 구조 분석 이론에 따라 문장 분석이 진행한다[6]. 자연어 문장 구조 분석은 선행 연구들에 의해 다양한 라이브러리 및 도구들이 존재한다. 그 중 NLTK 라이브러리를 사용한다. 작동 방법은 다음과 같다.

1) **분석 대상 문장마다 번호를 부여한다.**
2) **문장을 토큰화 하고, 각 토큰에 품사를 태깅한다.**
3) **단어와 품사를 딕셔너리 형태로 저장한다.**
4) **문장 번호, 원본 문장, 품사 태깅 결과를 하나의 JSON 형식으로 저장한다.**

요구사항 7-2번을 예시로, 문장의 구조를 분석하면 결과를 그림 4로 나타낸다.

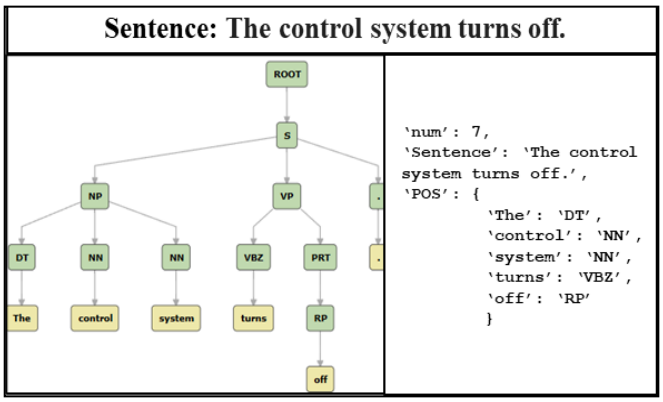

그림 4. 문장의 구조 분석 파싱

문장 구조 분석 결과를 바탕으로 의미적 분석을 진행한다. 의미적 분석에는 Fillmore의 Semantic Roles를 사용한다. Fillmore의 이론은 동사와 관련된 명사에 대한 역할을 부여한다[7]. 표 1은 기존의 Fillmore 이론의 일부 역할에 대해 설명한다.

표 1. 기존 Fillmore의 Semantic Roles의 일부

Roles	Definition
Agent	동사의 행동을 수행하게 하는 개체
Object	타동사의 행동에 직접 영향을 받는 존재
Experiencer	동사의 행동에 영향을 받는 존재
Instrument	동사의 행동에 인과적으로 관여하는 무생물적 존재

이를 UML 생성을 위해 사용하기 때문에 해당 도메인에 적절하게 의미를 축소 및 확장할 필요가 있다. 따라서 그림 5는 제안한 연구에서 사용되는 UML 생성을 위한 역할을 보여준다. 또한, 문장의 주동사를 구분한다.

Roles	Definition
Actor	행동의 주체
Object	사건의 영향을 받는 객체
Source	행위를 하는 객체
Target	행위를 받는 객체
Instrument	행위에 사용되는 도구나 수단

✚

Main Verb	Dynamic Verb	Do Verb	문장의 주요 동사
		Common Verb	
	Stative Verb	Be Verb	
		Has Verb	
Adjective			문장에서 명사를 수식하는 형용사

그림 5. 재정의된 Fillmore의 이론과 주동사 구분

그림 5에서 재정의된 Fillmore의 이론을 바탕으로 다음과 같이 프롬프트를 구성한다.

1) **문장 분석을 위한 케이스 문법에 대한 소개한다.**
2) **분석해야 할 대상의 조건을 제공한다.**
3) **적절한 격이 없는 경우, 이상한 값을 부여하지 않도록 조건을 제공한다.**
4) **문장의 번호, 원본 문장, 격 문법 결과를 하나의 JSON 형식으로 저장한다.**

해당 과정에서 생성된 JSON 파일을 문장 분석의 최종 결과로 사용한다.

3.2 UML 다이어그램 생성

자연어 문장 분석 결과를 바탕으로 유즈케이스 다이어그램을 생성한다. 이후, 규칙 기반으로 클래스 다이어그램, 시퀀스 다이어그램, 상태 다이어그램을 생성한다. 클래스 다이어그램은 객체의 구조에 대해 표현하고, 시퀀스 다이어그램은 객체 간의 상호작용을 표현하고, 마지막으로 상태 다이어그램은 객체 내의 상태 변화를 표현한다. 이들의 구성 요소를 자연어 문장 분석의 최종 결과인 역할론과 매핑하여 그림 6과 같은 다이어그램을 생성한다.

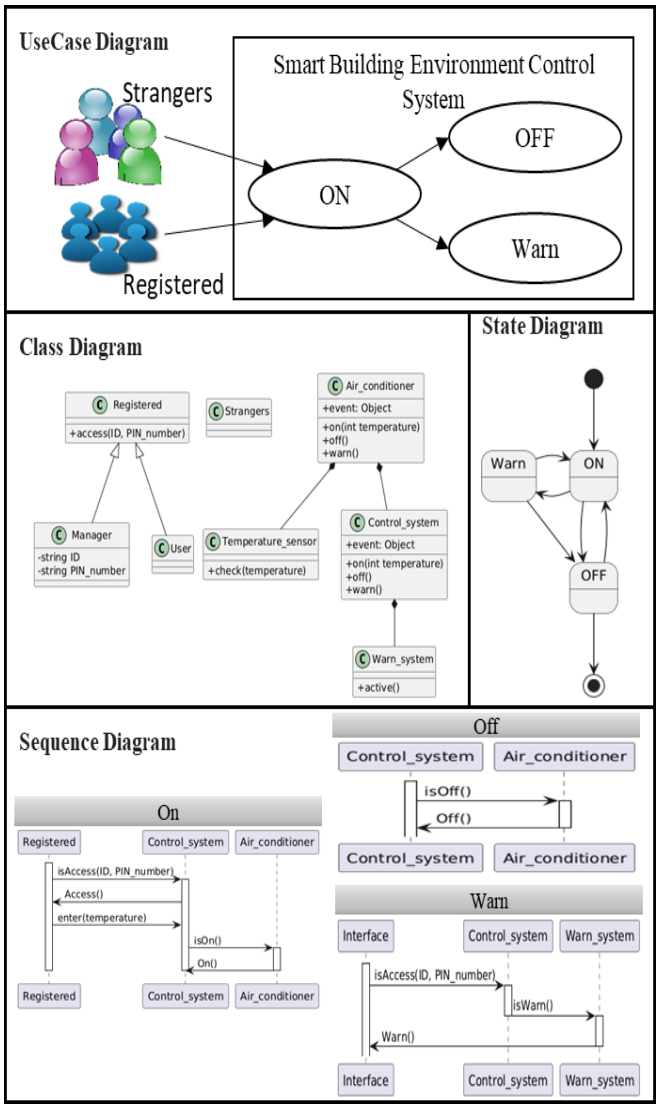

그림 6. 자연어로부터 생성된 다이어그램

3.3 코드 생성

자연어 분석 정보를 가지고 생성된 3가지의 UML을 바탕으로 스켈레톤 코드를 생성한다. 다양한 언어의 코드를 생성하기 위해 메타 모델링을 통한 모델 변환을 수행한다. Model Translator에서 다이어그램의 정보와 코드의 정보를 교환하기 위해 Translator Rule을 정의한다.

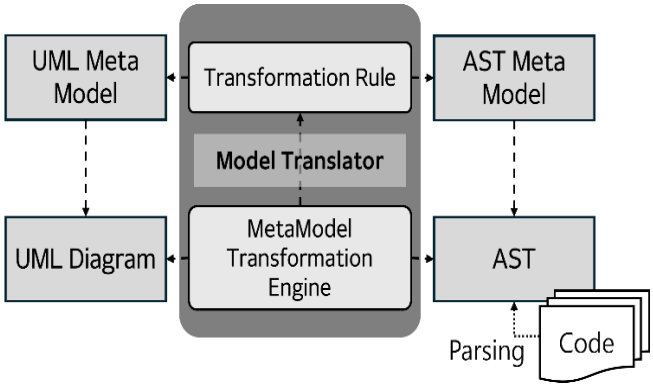

그림 7. 메타 모델링 방식

4. 결론 및 향후 연구

우리 소프트웨어공학자도 소프트웨어 개발 생명 주기의 체계적 절차와 생성형 AI 기술을 결합하는 방식이 필요하다. 이를 해결하기 위해 소프트웨어 개발 생명주기 내에 단계적으로 AI 접목이 필요하다. 이를 위해 AI Driven Requirement Engineering process를 제안한다.

이는 AI 지향 요구공학을 적용한 사례로, 자연어 분석 과정에서 Chomsky의 구조 언어학 이론과 Fillmore의 의미적 분석 이론을 활용한다. 자연어 분석 결과를 기반으로 설계 단계에서는 UML 다이어그램 생성과 메타 모델링을 통한 소스 코드를 자동 생성이 가능하도록 한다.

그러나, 프로젝트 규모가 크거나 요구사항이 더 복잡한 프로젝트를 통해 사례를 검증하는 것에는 여전히 한계가 존재한다. 또한, 프로젝트의 규모가 커질 수록 생성되는 산출물의 양이 많아지기 때문에 이 모든 단계적 프로세스에는 자동화가 필수적이다. 향후, 각 단계별로 자동화 시스템을 구축하기 위한 툴체인에 대한 연구를 수행할 예정이다. 이를 통해, AI 지향 요구공학 적용사례들을 늘리고 메커니즘을 보완할 수 있기를 기대한다.

Acknowledgement

본 연구는 2024년도 문화체육관광부의 재원으로 한국콘텐츠진흥원 (과제명: 인공지능 기반 사용자 대화형 멀티모달인터랙티브스토리텔링 3D 장면 저작 기술 개발, 과제번호: RS-2023-00227917) 지원을 받아 수행된 연구임.

Reference

[1] N. B. Ruparelia, “Software development lifecycle models,” ACM SIGSOFT Software Engineering Notes, vol. 35, no. 3, pp. 8–13, 2010.

[2] 김길수. 인공지능의 신뢰에 관한 연구. 한국자치행정학보, vol. 34, no. 3, pp. 21–41, 2020.

[3] Majdinasab, Vahid, et al. "Assessing the Security of GitHub Copilot's Generated Code-A Targeted Replication Study." 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2024.Codeium

[4] Ovi, Md Sultanul Islam, et al. "Benchmarking ChatGPT, Codeium, and GitHub Copilot: A Comparative Study of AI-Driven Programming and Debugging Assistants." arXiv preprint arXiv:2409.19922 2024.

[5] Yetiştiren, Burak, et al. "Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt." arXiv preprint arXiv:2304.10778 (2023).

[6] N. Chomsky, Syntactic structures, USA: Mouton de Gruyter, 2002.

[7] C. J. Fillmore, “The Case for Case,” New York: HR&W, 1968.