Object-Oriented Analysis and Design 3227

객체지향개발방법론

1팀 7주차 발표

Optimized and Balanced SW Development Process

202211327 윤승모

202211261 김강민

202212353 문서인

202011362 정상현

CONTENTS

Part1 문제 정의

: 왜 새 프로세스가 필요한가

같은 요구사항 변경(RVC Right Sensor 제거)을 두 방식에 독립 적용한 결과, 두 방식 모두 한쪽 약점이 뚜렷했다.

전통 UP/OOAD (사람 중심)

- 변경 표면적이 작아 국소·신속
(production +32줄)
- System Test 20개를 수동 일괄 수정
→ 휴먼 에러 여지
- 변경 의미가 내부 상태에 묻혀 추적이 어려움

Vibe Coding

- 문서·코드·테스트를 동시 생성·정합
- 소규모 변경을 과설계로 키움
(production +254줄)
- 1차 산출물에 결함(후진 버그)
→ 사람 검증 필요

[결론] 전통은 느리지만 빠짐없고, AI는 빠르지만 과설계·미검증 위험. 두 강점만 살리려면 역할을 나누고 게이트를 두는 프로세스가 필요 → AID-UP

Part2 핵심 원칙

: 방법론의 세 가지 원칙

01

AI proposes,
Human disposes

- AI : 초안 생성·반복 노동·동기화 담당
- 사람: 의도·아키텍처·설계원칙·최종 승인 담당
- AI는 어떤 게이트도 스스로 통과 못함

02

Dynamic Model-Driven

- Use Case → SSD → Domain →
Sequence → Class → Code 순서를 보존
- 전통의 Pair Programming을
사람 ↔ AI 게이트로 확장한 것

03

Continuous Traceability

- 요구↔UML↔코드↔테스트↔운영을
양방향 동기화
- 변경 발생 시 전 산출물에 변경 후보를 전파
→ Maintenance Nightmare 방지

Part2 통제 장치

: AI를 통제하는 3가지 장치

"AI가 만들고 사람이 승인한다"만으로는 부족하다. 생성 이전에 작동하는 통제 장치가 필요하다.

01

Core Invariant

절대 깨지면 안 되는 최소 금지선

- 충돌 위험 방향 이동 금지
- 회전 시 좌측 우선
- 먼지 감지 → 5초 내 Boost
- 모드 흐름 유지
- 절차지향 금지, OO 구조
- Driver→Controller→Processor 준수

02

Prompt Refinement Loop

자연어 요청을 작업 계약으로

- 사람이 자연어로 요청
- AI가 관련 산출물 후보 제시
- AI가 확인 질문지 생성
- 답변을 Decision Log에 축적
- AI가 OOAD Prompt Contract 생성
- 계약 기준으로만 작업

03

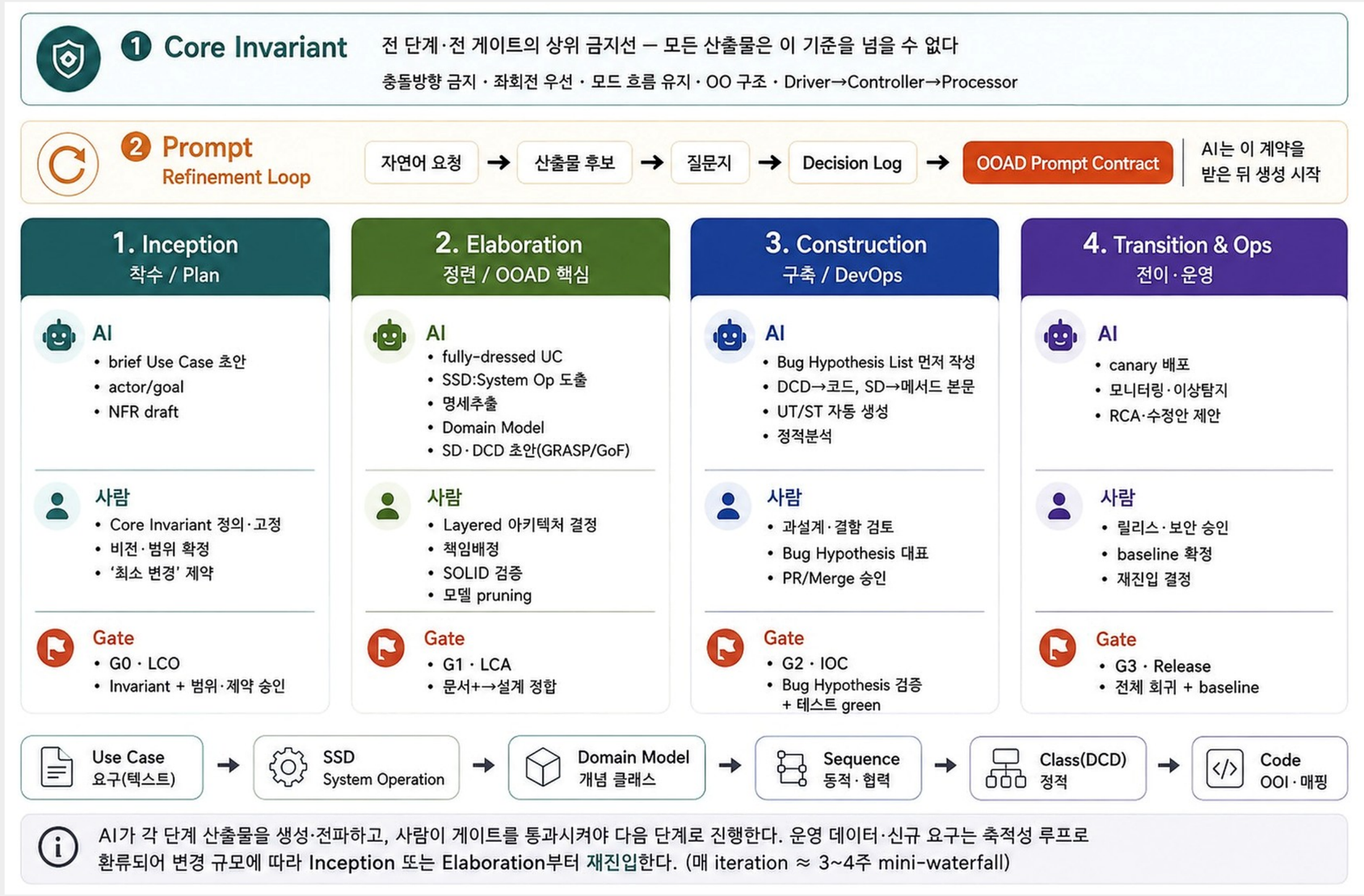
Bug Hypothesis List

코드 전, 예상 실패 시나리오

- 후진→전진 반복 루프
- right 조건·테스트 잔존
- 좌측 우선 회전 정책 붕괴
- 2센서 판단의 방향 편향
- 재측정 없이 바로 이동
- Boost·LowBattery 흐름 충돌

Part3 전체 그림

: AID-UP 한눈에 보기



Part3 단계별 상세

: 무엇을 · 어떻게 · 어디까지

단계	AI agent	사람	게이트
Inception	Core Invariant·Prompt Contract 후보, brief UC 초안	Invariant 고정 · 범위·"최소 변경" 제약 명시	G0
Elaboration	fully-dressed UC·SSD·Domain·SD·DCD 생성(GRASP)	책임배정·SOLID 검증 · 문서↔설계 정합	G1
Construction	Bug Hypothesis 먼저 → 코드·UT/ST·정적분석	과설계 점검 · 가설 대조 · PR 승인	G2
Transition	+136canary 배포·모니터링·RCA·수정안 제안 / −78	릴리스 승인 · baseline 확정 · 재진입	G3

Part4 단계별 상세

: 연속 추적성과 피드백 루프

양방향 추적 체인

User Requirement ↔ Use Case ↔ SSD / System Op
↔ Sequence Diagram ↔ Class Diagram(DCD)
↔ Source Code ↔ Test Case ↔ 운영 지표

변경이 어느 지점에서 발생하든, AI가 위·아래로 영향을 추적해 연결된 모든 산출물에 변경 후보를 만들어 둔다.

피드백 루프 작동 순서

1. 운영에서 이상징후·신규 요구 발생
2. AI가 추적 체인으로 영향 범위 분석
3. AI가 UC·UML·코드·테스트에 변경 후보 전파
4. 사람 게이트에서 승인·우선순위 결정
5. 규모에 따라 Inception 또는 Elaboration부터 재진입

Part4 실증

: 게이트의 필요성

게이트는 우리 팀 6주차 실측(Right Sensor 제거)에서 도출됐다. 두 방식의 약점이 곧 게이트의 존재 이유.

실측 1 : 변경 비용은 다른 산출물로 전파

전통(사람) production +32 / 합계 +423

Vibe(AI) production +254 / 합계 +596

핵심은 수정량의 크기보다 변경 비용이 발생하는 위치이다.
AID-UP은 이를 게이트 단계에서 선제적으로 관리한다.

실측 2 : AI는 최소 변경을 구조 개선으로 해석

- Vibe +254의 거의 전부가 Controller.cpp(+202)
- 한 파일 상태 4개·메서드 10개로 펼쳐 과설계.

→ G0 "최소 변경" 제약 + G2 과설계 점검의 근거
→ G2 테스트 통과 조건 + Bug Hypothesis: 후진 버그를 코드 전 가설로 뽑아 ST로 포착

[핵심] 단순 패치 속도는 전통, 명시적 구조·추적성은 Vibe. AI에게 생성·전파를 맡기되 사람이 게이트에서 과설계·결함만 잡아 속도와 유지보수성을 동시에 가져간다.

Part4 적용 사례

: RVC "Right Sensor 제거" 한 바퀴

Inception — Invariant 고정·계약

- 사람: Core Invariant(좌회전 우선·모드 흐름) 고정 + "최소 변경" 확정 / AI: 산출물 후보·질문지 → Prompt Contract 확정

G0

Elaboration — 모델 수정

- AI: UC5 right 분기 삭제·"우회전 후 재측정" 추가 → SSD 2회 호출 → DCD Boolean[3]→[2] / 사람: "right는 회피 후보" 책임 재배정

G1

Construction — 가설 먼저

- AI: 코드 전 Bug Hypothesis 작성 → 가설별 테스트 → 코드·UT/ST 전파 / 사람: 과설계 점검 + 후진 버그 ST로 포착·수정

G2

Transition — 배포

- AI: 전체 회귀 green 확인·diff check / 사람: baseline 확정 승인

G3

감사합니다