
객체지향개발방법론 (OOAD)

SW DEVELOPMENT PROCESS

202211291 김호경

202111282 노을영

202111284 문재현

Agenda

- Summary
- SW Development Process

Summary

HW_5

문서화 능력

- AI는 문서화 작업에 뛰어남
- 특히, SRS, SSD를 이미 있는 포맷에 맞춰서 작성해달라 했을 때 강점을 느낄 수 있었음
- 표, 다이어그램 등 다양한 형식을 쉽게 만들 수 있기 때문에 문서화 작업이 많이 간편해짐
- 다만, 표 및 다이어그램 스타일이 미관상 좋지 않거나 난잡할 수 있는데, 사용할 수 있는 도구의 한계로 보임
- 있어빌리티를 높이려던 디자인 부분에서 사람의 개입이 필요하거나, 외부 Tool을 불러서 발전시키는 것이 좋다고 느낌

RVC Control SW Software Design Description

본 문서는 RVC Sim 1016-2009의 SSD-구조를 기준으로 작성한다.

1. Scope

1.1 Purpose

이 문서는 RVC(Robotic Vacuum Cleaner) 자동 청소 제어 소프트웨어의 Software Design Description(SDD)이다. #docx/jrn.xml 에 정의된 요구사항을 참조 C++ 20 구조의 변형하고, 설계 context, stakeholder concern, design viewpoint, design view, design rationale를 정리한다.

1.2 System Scope

설계 대상은 #rcvcontroller 기반 제어 부직과 #gridSimulator 기반 CLI 시뮬레이터이다. controller는 sensor 입력을 기반으로 motor motion과 cleaner power를 포함한 command를 생성한다. simulator는 감속 환경으로서 scenario map에서 sensor 감동 및 command를 적용한다. 본 SDD의 범위에 포함되지 않는 항목은 실제 하드웨어, 실제 센서 하드웨어, 배터리, 내외벽, LNA, 영구 저장소 설계이다.

1.3 Design Goals

RVC Control SW Software Requirements Specification

본 문서는 RVC Sim 1016-2009의 SRS 구조를 기준으로 작성한다.

1. Introduction

1.1 Purpose

이 문서는 RVC(Robotic Vacuum Cleaner) 자동 청소 제어 소프트웨어의 Software Requirements Specification(SRS)을 명세한다. 요구사항은 #docx/jrn.xml 에 정의된 SRS, OAD 산출물, 항목 C++ 20 기반 및 테스트 가능한 CLI 시뮬레이터 범위에 한정되어 작성되었다.

본 문서는 구현 범위의 외부 시스템이 제공해야 할 외부 행위, 인터페이스, 기능 요구사항, 제약, 요구사항, 검증 기준을 명세한다.

1.2 Scope

대당 시스템은 RVC 자동 청소 제어 로직과 제어를 관통하기 위한 CLI 그래픽을 포함 생성한다.

범위에 포함되는 항목은 다음과 같다.

- 자동 청소 시작 및 종료
- 장애 정체를 interrupt 처리
- 좌측, 우측, 전방 센서의 periodic sampling
- 장애물 처리 및 직각 방향 이동
- 전지 고지 시 boost 청소
- CLI 시뮬레이터를 통한 요구사항 검증

범위에 포함되지 않는 항목은 다음과 같다.

- 실제 하드웨어
- 실제 센서 하드웨어
- 배터리, 충전, 작동 관련
- 제거나 전, 하드웨어, 영구 저장소
- 물리적 외장 케이스, 노이즈, 적외선 적외선 검출기

4. Design Content

4.1 System Boundary

코드 내용 분석

- 프롬프트를 통해 SOLID 원칙에 따라 코드를 작성해달라고 요청했으나, RVC와 Simulator에 대해서만 원칙을 적용하였음.
- RVC 내부에 대해서는 적용하지 않아서 잘차지향적인 코드가 작성되었다고 생각함.
- 다음 주차인 SOLID 분석 기간 동안 해당 부분을 수정해서 객체지향적인 코드를 작성해야한다고 느낌.
- Simulator와 RVC를 한번에 같이 만들어달라고 해서 생긴 문제점으로 생각됨.

Principle	Application
SRP	rcvcontroller 는 제어 결정한 담당하고, gridSimulator 는 환경과 이동 적용만 담당한다.
OCF	sensor 입력은 PeriodicSensorData 와 interrupt API로 추상화되어 새 sensor 추가 시 controller 확장이 가능하다.
LSP	simulator와 실제 하드웨어 어댑터는 같은 Command 의미를 따르므로 대체 가능하다.
ISP	controller의 public interface는 시작, 중지, interrupt, tick, 판단에 필요한 작은 operation으로 분리된다.
DIP	controller는 concrete simulator나 hardware에 의존하지 않고 객체와 추상 command에만 의존한다.

코드 작성 능력

- 처음부터 만드는 것 보단, PLAN 모드로 계획을 잡고 시작했을 때 결과물이 훨씬 좋았음
- 코드 작성은 쉽고 빠르지만, 일부 직관적이지 않는 부분도 있었음 → 주석을 달거나 문서를 좀 더 보는 것으로 해결
 - 일반적으로 3 단계에 거쳐서 해결되는 문제 (양 옆 값 받아오고 → 값 검증하고 → 다음 판단 내리고)를 한 줄에 해결하려 하는 경향
- 다만, 앞의 Trouble Shooting과 같이 요구 사항을 빠뜨리는 경향도 있기 때문에 사람의 검증이 필요함

RVC 제어 소프트웨어와 CLI 시뮬레이터 구현 계획

요약

- #docx/jrn.xml 의 4-6쪽 요구사항을 기준으로 RVC 자동 청소 제어 소프트웨어와 테스트용 CLI 기반 시뮬레이터를 C++로 구현한다.
- OAD 산출물은 #docx/oad_analysis.xml 에 정리하고, 코드는 제어 로직과 시뮬레이터를 분리해 본다.
- 유닛 테스트는 구급테스트로 작성하고, 시스템 테스트는 시뮬레이터 시나리오 실행 결과로 검증한다.

주요 구현

- CMake 기반 C++ 20 프로젝트를 만든다.
- 제어 소프트웨어는 센서, 모터, 청소기 시계 입력을 추상 인터페이스로 두고 #rcvcontroller 가 해당 인터페이스를 구현하게 한다.

SSD-05 방에서 loop를 돌 때 front is open이면 빠져나오는데, 후진 한 번 하면 바로 앞에 장애물이 없는 상태가 되므로 결국 앞으로 갔다 뒤로 가는 상황이 되, 이 문제를 해결하기 위해선 좌/우의 방향이 0, 1, 0, 1에서 0, 1, 1, 0로 변경된 후진 로직을 반영해야 함

변경된 로직에 따라 실제 후진하는 코드 변경 후진하는 테스트 케이스 추가, Unit test, Simulator를 이용한 Test 코드가 수정된 문서도 그에 맞게 함께 수정

3m 34s 동안 작업 >

수정 완료했습니다.

핵심 변경은 #RvcController.cpp (line 54)에서 escaping 상태의 정출 조건을 front/left/right, 전제 아니라 left/right에 한 것으로 바꾼 것입니다. 이때 후진 직후 정출이 일어도 좌/우가 모두 막혀 있으면 계속 backward를 발행하고, 좌/우 중 하나가 열릴 때만 회전합니다.

추가한 검증:

- Unit test: #controller_tests.cpp (line 102)
- Simulator test: #system_tests.cpp (line 71)
- 문서 명산: requirements, SSD-05, SD-05, class/domain diagram, traceability

검출 결과:

- cmake --build build --config Debug 통과
- ctest --test-dir build -C Debug --output-on-failure 통과, 14/14
- scenario/backward_escape2_rvc_수동 실행에서도 tick 2 이후 frontInterrupt=false 인제 계속 후진하다가 side open에서 회전하는 것 확인했습니다.

테스트 작성 능력

- 테스트 작성 능력은 월등히 뛰어남
- 짧은 시간 안에 많은 테스트를 만들어 냄
- 다만, 테스트 개수가 많을 뿐이지 실제로 유용한 테스트 인지는 역시 사람이 검증해야 하는 문제가 있음
 - 시뮬레이터 검증 테스트 부분이 쓸데 없이 많다고 생각하긴 함
- AI가 테스트에 실패하면, 코드를 고치는 게 아니라 테스트가 통과하도록 고치려는 경향이 있음
- 위 내용에 따라, 테스트 불통 시 인간이 코드를 직접 고치거나, TDD를 수행하는 것도 좋은 방법으로 생각함

3. 컨트롤러 유닛 테스트 결과

컨트롤러 유닛 테스트는 결정적인 센서 입력과 인터럽트 입력을 사용해 #rcvcontroller 의 상태 전이, 회전 판단, 정출 판단, 전지 boost 제어 규칙을 검증한다.

번호	테스트 케이스	검출 대상	결과
1	#rcvcontrollerTest::ControlerMovesForwardWhenFrontIsClear	전방 장애물 제거 시 정출을 시작하고 전진하는지 검증	Passed
2	#rcvcontrollerTest::SideControlerReturnsToBackWhenFrontIsInterrupt	side 상태에서 전방 인터럽트와 전지 감지를 무시하고 정지/회전 상태를 유지하는지 검증	Passed
3	#rcvcontrollerTest::StopCleaningTurnsStopAndOff	청소 중지 요청 시 인터럽트와 전지 고지 명령을 반환하는지 검증	Passed
4	#rcvcontrollerTest::FrontInterruptTriggersImmediateAvoidance	전방 장애물 인터럽트가 즉시 회피 동작으로 이어지고 출력이 커지는지 검증	Passed
5	#rcvcontrollerTest::FrontInterruptStopsCleaningWhenTurn	전방 인터럽트가 1 tick 후 소시터의 정지 명령으로 복귀하는지 검증	Passed
6	#rcvcontrollerTest::TurnsTowardOpenSide	열린 측면 방향으로 회전하고 회피 중 tick이 커지는지 검증	Passed
7	#rcvcontrollerTest::AlterateWhenBothSidesAreOpen	좌우가 모두 열릴 때 회전 방향을 번갈아 선택하는지 검증	Passed
8	#rcvcontrollerTest::AlterateWhen		
9	#rcvcontrollerTest::AllSidesAre		

RvcControllerTest (17)

- AllSidesAreOpenEscapingAndKeepsBacklogUp
- AllSidesAreOpenControllerClearsObstaclesWhenBoostBudget
- AlternatesWhenBothSidesAreOpen
- AlternationPersistsAcrossClearTicks
- AvoidanceOutputsStayOpenWhenBoostStatesMaintained
- ControllerMovesForwardWhenPathsClear
- DustBoostLastsConfiguredTicks
- DustDetectionRefreshesBoostBudget
- EscapingIgnoresOpenFrontUntilSideOpens
- EscapingTurnsTowardRightSideExit
- EscapingUsesAlternationWhenBothSidesOpen
- FrontInterruptsConsumedAfterOneTick
- FrontInterruptTriggersImmediateAvoidance
- IdleControllerReturnsStopAndIgnoresFrontInterrupt
- StopCleaningReturnsStopAndOff
- TurnsTowardOpenSide
- ZeroTickBoostConfigurationDoesNotBoostOrOut

HW_6

Oracle vs Vibe

Oracle 코드는 오랜 기간 고민하면서 나온 구조이기 때문에 조원 모두가 구조를 명확하게 인지하고 있었고, 어떤 부분을 고쳐야할지 바로 알고 수정하였음.

좌우 확인 후 뒤로가는 요구사항에 대해서 어떻게 해야 올바르게 동작할지 로직을 짜는 과정에 대부분의 시간을 소요함.

이후 코드 및 테스트에서는 별 다른 시간 소요 없이 마무리 할 수 있었음

변경사항이 크게 없다는 것이 제일 중요함

Vibe Coding의 코드를 AI가 짜주고 결과물에 대해서 컨펌만 진행해왔기 때문에 요구사항에 대해 어떤 부분을 변화시켜야할지 인지하고 있지 못했고, AI에게 모든 일을 일임시키니 수정된 부분이 여러 있어 확인하는데 시간이 오래 소요되었음.

변경된 요구사항을 전달하는 것만으로 짧은 기간만에 수정이 완료되었으나, 코드가 직관적이지 않고 사람이 파악하기 힘든 부분이 다소 있었음

이에 따라 코드를 쉽게 풀어서 표현한 문서나 보조 도구가 있는 것이 중요하다는 것을 깨달음

Oracle vs Vibe

Oracle와 Vibe coding 둘 다 State 변수를 통해 RightSensor 제거 상황을 해결했다는 점은 동일하나

Oracle 코드보다 훨씬 단순하게 Vibe coding 코드가 작성되어 있어서 문제가 분명 있을 것이라 생각하고 코드를 전부 확인함.

그 과정에서 FrontSensor가 Interrupt로써 처리되고 있지 않음을 확인함.

FrontSensor 값을 받아오는 함수의 이름을 FrontObstacleInterrupt라고 명명하여 발견이 어려웠음.

```
void RvcController::onFrontObstacleInterrupt() {  
    if (running_) {  
        frontInterruptPending_ = true;  
    }  
}
```

원래 구조에서 '인터럽트 감지 즉시 모터 정지'만 추가한다면 범위는 적습니다.

- RvcHardwareAdapter.cpp의 emergencyStop(): 같은 메서드 추가: 1-2줄
- Rvc.cpp (line 24)에서 frontInterrupt 감지 직후 hardwareAdapter->emergencyStop(): 호출: 1-1줄
- SimulatedHardwareAdapter.cpp의 구현 추가: 5-15줄
- 실제 하드웨어 adapter가 있다면 가기에 모터 stop 호출 추가: 5-20줄
- 테스트 추가: 20-60줄

다만 시뮬레이터 기준으로는 지금도 interrupt가 있는 tick에서는 Forward가 아니라 TurnLeft, TurnRight, Backward 등으로 바뀌어서 '인진 명령을 계속 내리지 않는다'는 요구는 이미 만족해요.

원래 차이는 controller 한단 전체 actuator 레벨에서 먼저 stop을 막는겁니다.

그래서 내가 구현한다면 최소 변경은 이런 느낌입니다.

```
cpp  
if (!isFrontObstacleInterrupt) {  
    hardwareAdapter->emergencyStop();  
    controller->onFrontObstacleInterrupt();  
}
```

본 설계 변경은 아니고, '하드웨어 adapter에 긴급정지 종료 하나 추가'하는 정도예요. 단, 로그에 emergency stop을 남기거나 ISR-thread/atomic/debounce까지 전체 원메시디언게 넣으면 100줄 안팎까지는 권장 않아요.

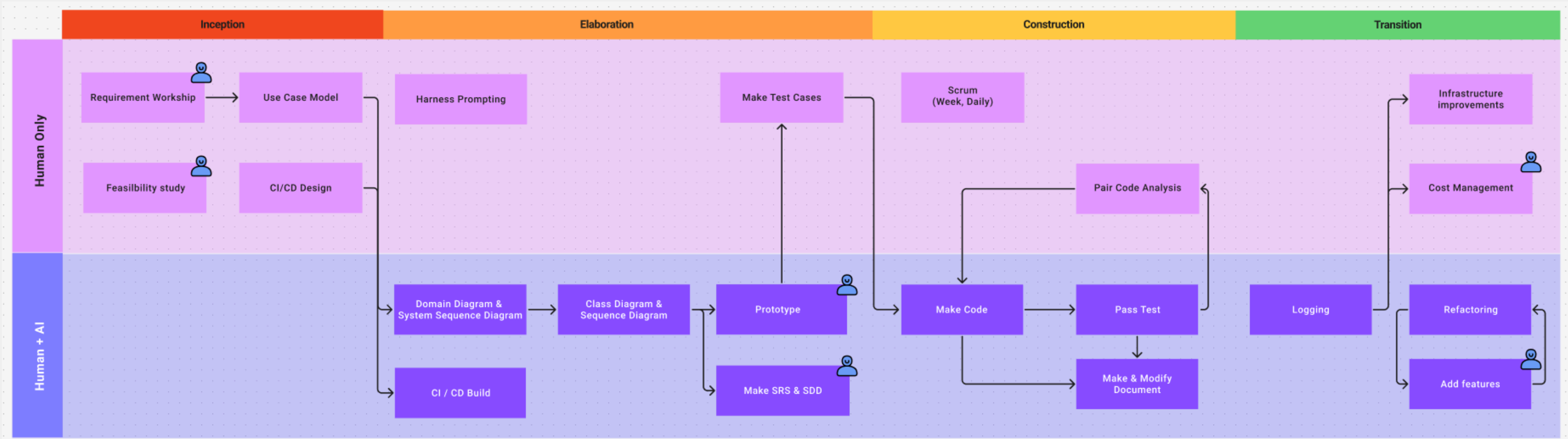
Summary

이전까지의 발표 내용

- 문서화 능력 good
- Plan 모드를 통한 코드 작성 능력 good
- 코드 작성에 대한 원칙을 명확하게 제시를 해야함
- 짧은 시간 안에 많은 테스트를 만들어 낼뿐 실제로 유용한 테스트인지 검증 필요
- AI가 짜준 코드에 대한 설명 문서나 보조 도구가 필요(직관적으로 알기 힘든 코드들이 존재)
- CI/CD 환경 구축은 Agent가 잘함
- 요구사항 파악 및 검증에 있어서 사람의 손길이 필요

SW Development Process

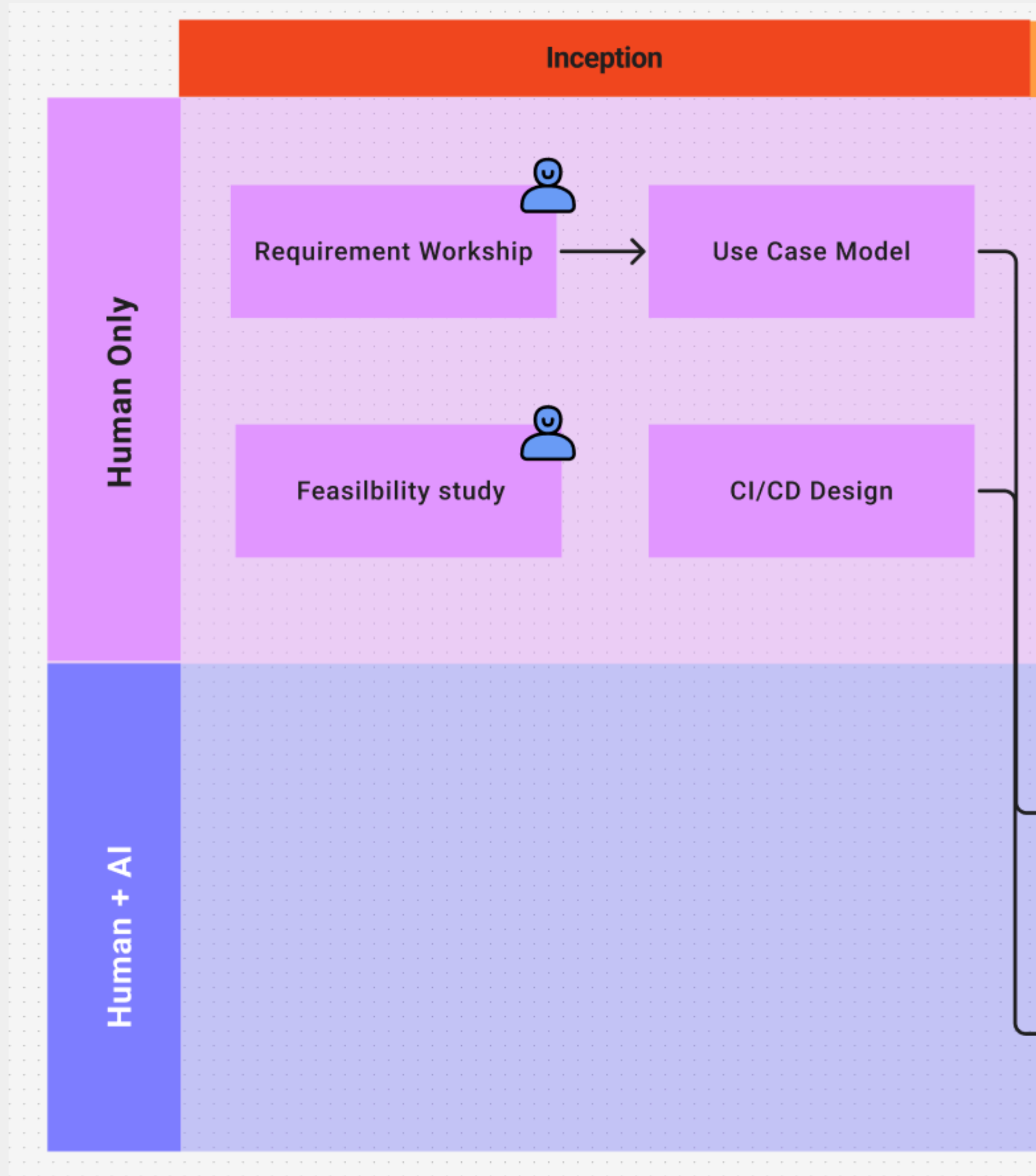
SW Development Process



프로젝트가 뿌러질 정도의 책임이 필요한 사항은 무조건 사람이 주체/ 단순 개발에 있어서 AI를 적극 활용하는 방향

AI를 쓰기 전에는 항상 그 환경을 구축하는 작업 필요 ⇒ Inception에서 AI를 안 쓸거면 안 써도 되고, 쓸거면 환경 구축도 같이 해야 함

Inception



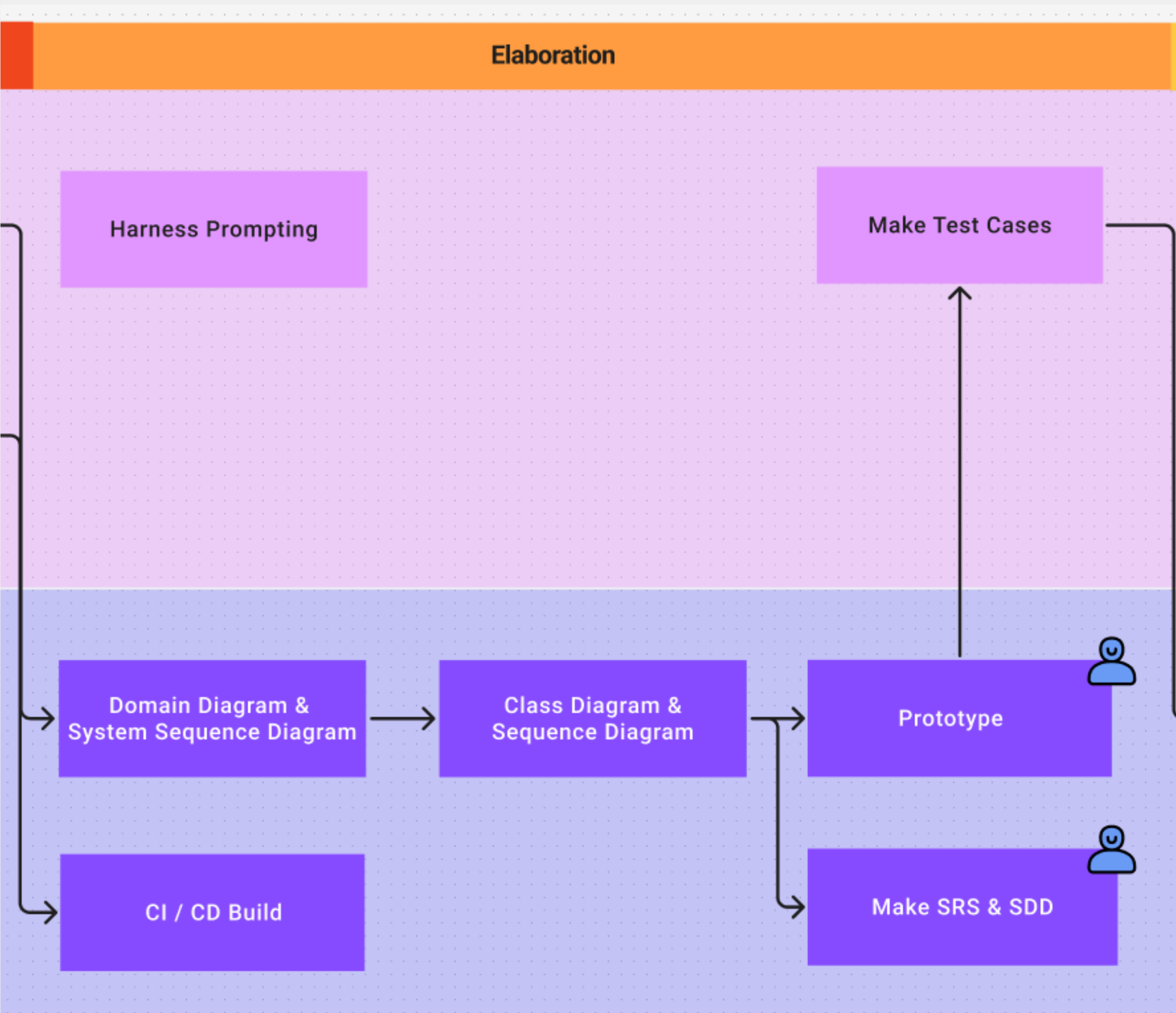
AI가 없이도 할 수 있어야 하는 작업들

- Requirement Workshop: 요구사항을 파악하는 것이 잘못 되면, 개발 전체가 잘못된 방향으로 갈 수 있음
- Use Case Model: 여기까지 만들며 정확한 Actor 및 요구사항을 파악하고, 이후에 변하지 않도록 기초를 다져놓음
- CI/CD Design: 개발 환경을 설계하는 것 또한 인간의 책임. 제일 중요한 부분은 비용 문제와 관련이 되어있기 때문. 궁극적으로는 사람 + AI가 쓰기 편한 구조를 만드는 것
- Feasibility Study: 리스크 분석 및 비용 예측을 하며 프로젝트의 타당성을 검사함. 똑같이 비용이 연관되어 있어서 사람의 책임

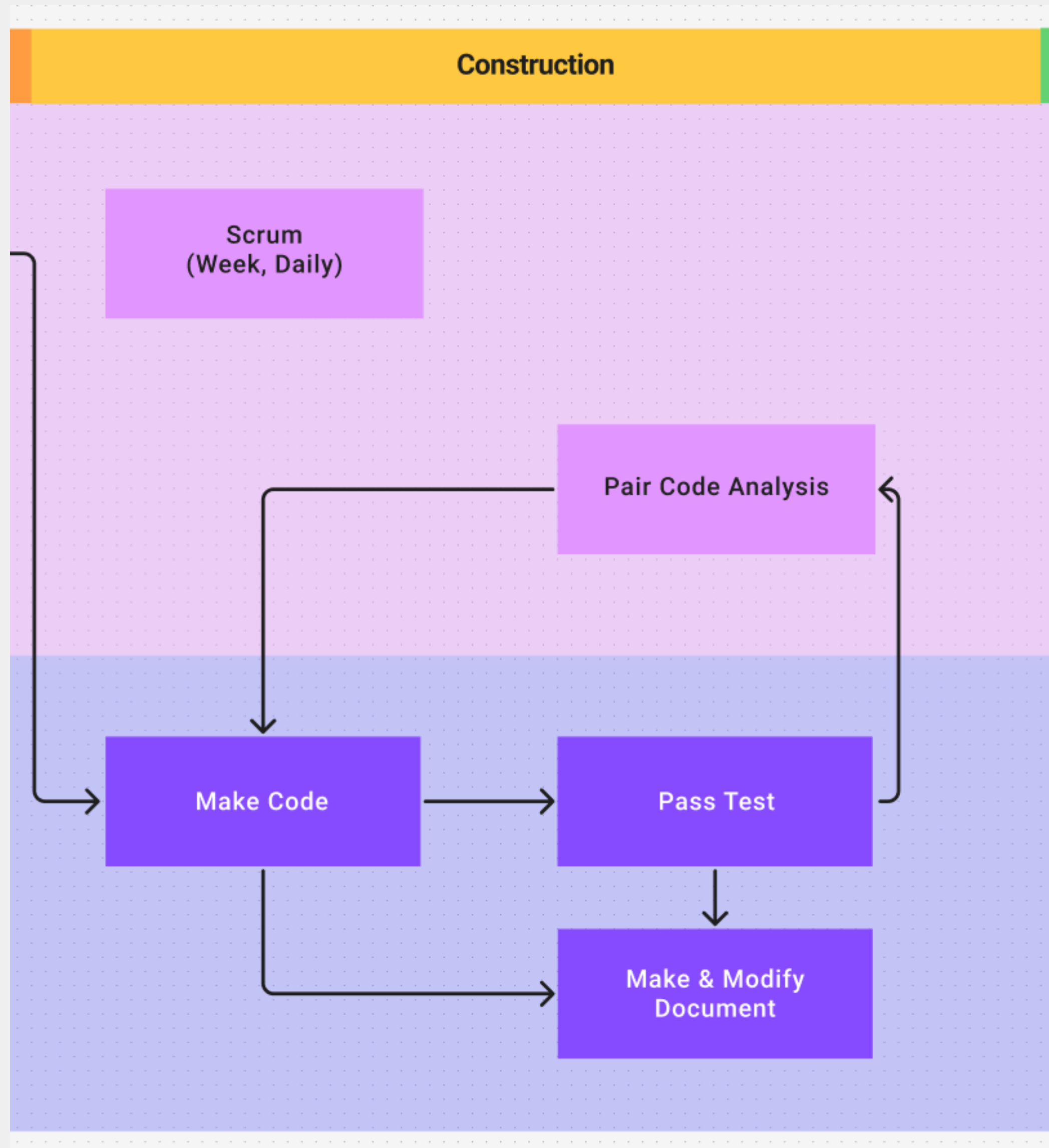
Elaboration

AI가 본격적으로 개입하는 부분

- Harness Prompting: AI가 작동하는 하네스(환경)를 만들어서 품질 놓고 일관적인 결과물을 낼도록 함
- 여러 Diagram: AI의 도움을 빌려 내용을 작성하고, 알고리즘 및 로직을 같이 검증함
- CI / CD Build: 앞에서 설계한 CI / CD 를 AI랑 같이 빌드
- Make SRS & SDD: 프로젝트의 이해관계자 모두가 봐야 하는 Standard 문서
- Prototype: AI를 사용해서 빠르게 프로토타입을 만들고, 프로젝트의 핵심 가치를 검증하고 필요한 테스트 케이스를 발견함
- TDD Apply: 이 개발방법론은 TDD를 기반으로 하고 있어서, Test Code를 만드는 과정을 통해 프로젝트의 기반을 잡음



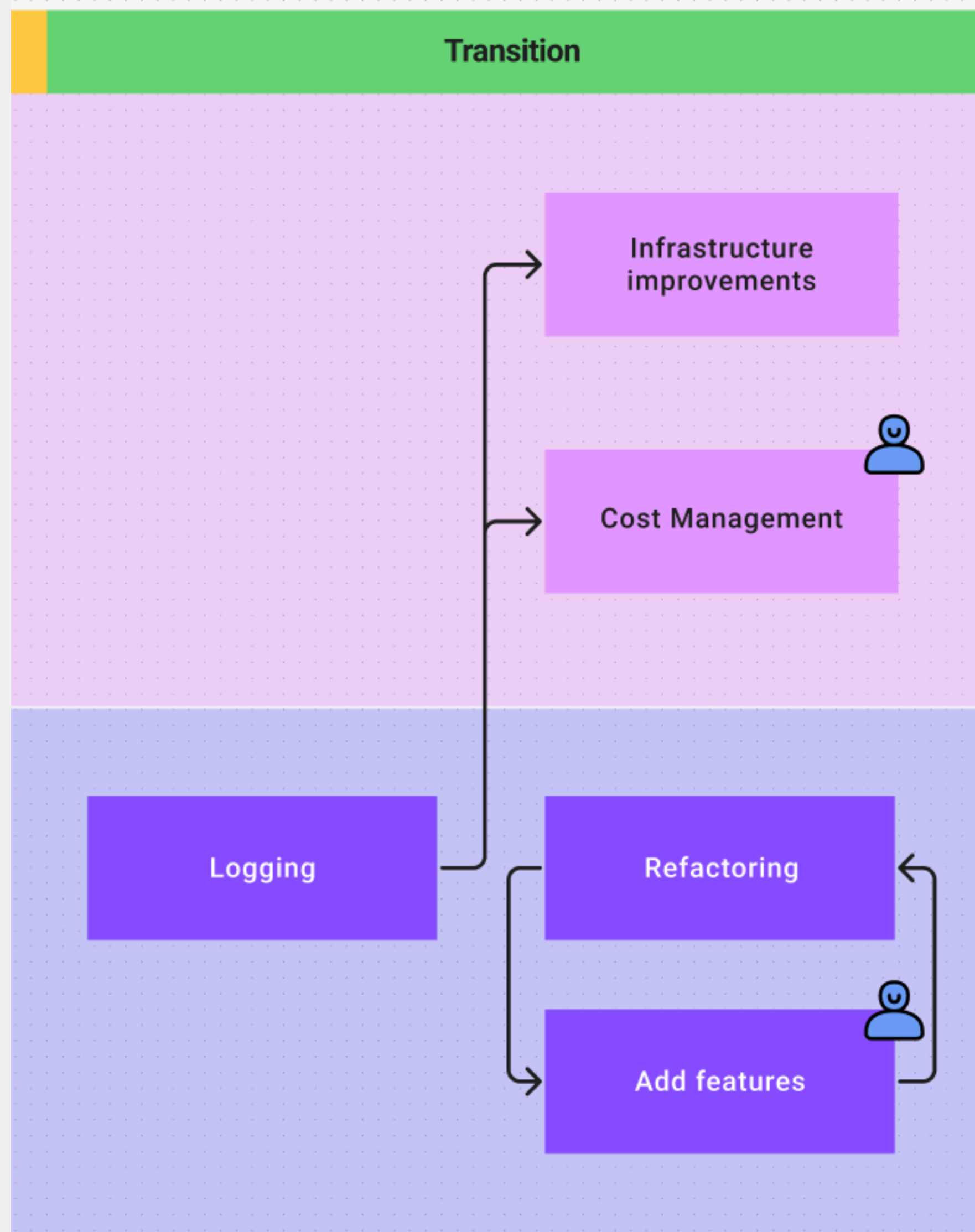
Construction



기존 문서를 기반으로 AI를 적극 활용하여 제작

- Pair Programming
 - 2명이 조를 이루어 각자 코드를 AI를 통해 생성한 뒤바꾸어 살펴보고 리뷰를 진행, 페어 코드 책임 리뷰의 관점
 - 조는 주마다 바뀌가며 진행한다.
- Scrum
 - 데일리 스크럼, 위크 스크럼(주 시작 / 주 종료)
 - 위크 스크럼: 내가 AI를 이렇게 썼는데 이런 문제가 있었다 이거 어떻게 개선하면 좋은가? (반복 개선)
 - 데일리 스크럼: 내가 이 코드를 이런 plan으로 짰는데 관찬은가를 팀단위로 논의하고 진행
- AI Context 문서
 - 코드를 만들면 그거에 대한 문서가 반드시 하나 나와야 한다 → CI환경 연결
- 테스트 통과
 - 이미 제작된 TestCase들을 바탕으로 AI가 Test 수행

Transition



프로젝트를 유지보수하는 단계

- Logging: AI를 사용한 로그 수집 및 분석을 통해, 프로젝트의 문제점 및 개선점을 파악함
- 리팩토링 및 기능 추가
 - Add features: 크지 않은 기능을 추가
 - 큰 기능이 들어간다면, Add features에서 하지 않고 다시 처음으로 돌아가서 검증을 먼저 수행
 - Refactoring: 코드 품질 검증 및 구조 개선
- Infrastructure: 인프라 개선. 고가용성 / 장애 대비 / 접속 시간 향상 등
- Cost Management: 비용 관리. 쓸모 없는 인스턴스 잘라내기, 스토리징 전략 바꾸기 등

New Disciplines



Discipline	Artifact	Inception	Elaboration	Construction	Transition
AI Coding Agent	Project Rule		S	R	
	Skill		S	R	R
	AI Context file		S	R	R
	MCP Tool		S	R	R

Summary

발표 내용과 적용된 내용 비교

- 문서화 능력 good → Make SRS & SDD, Make & Modify Document
- Plan 모드를 통한 코드 작성 능력 good → Scrum
- 코드 작성에 대한 원칙을 명확하게 제시를 해야함 → Harness Prompting
- 짧은 시간 안에 많은 테스트를 만들어 낼뿐 실제로 유용한 테스트인지 검증 필요 → TDD
- AI가 짜준 코드에 대한 설명 문서나 보조 도구가 필요(직관적으로 알기 힘든 코드들이 존재)
- CI/CD 환경 구축은 Agent가 잘함 → CI / CD Build
- 요구사항 파악 및 검증에 있어서 사람의 손길이 필요 → Inception