객체지향개발방법론
Practice #7

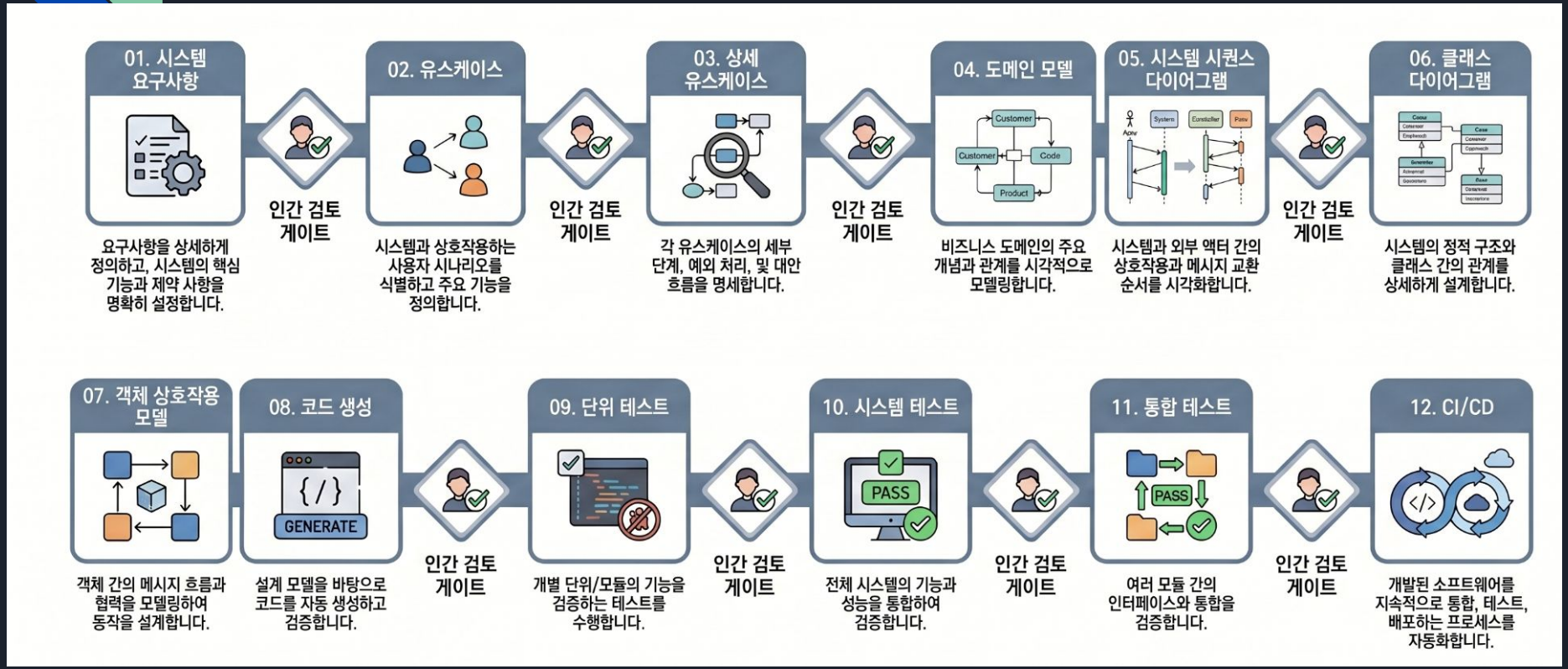
202211287 김태인

202311252 곽수호

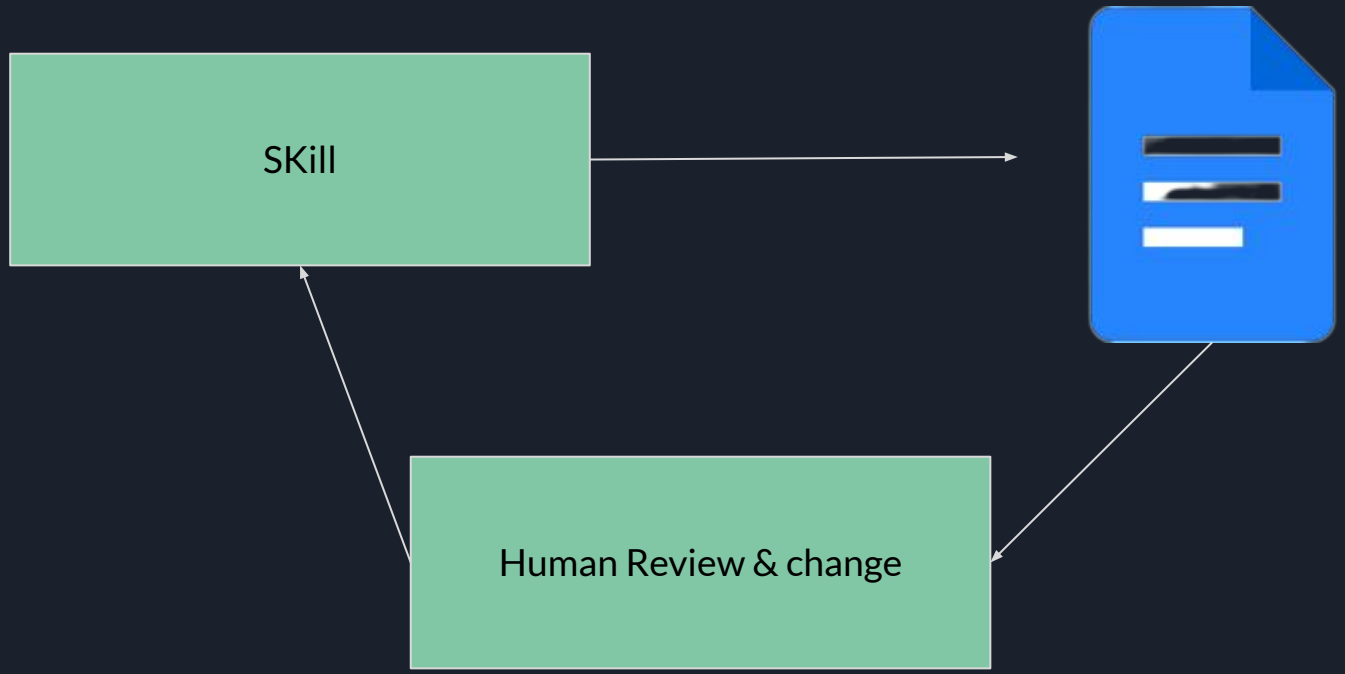
202111368 정선민

202211378 조성원

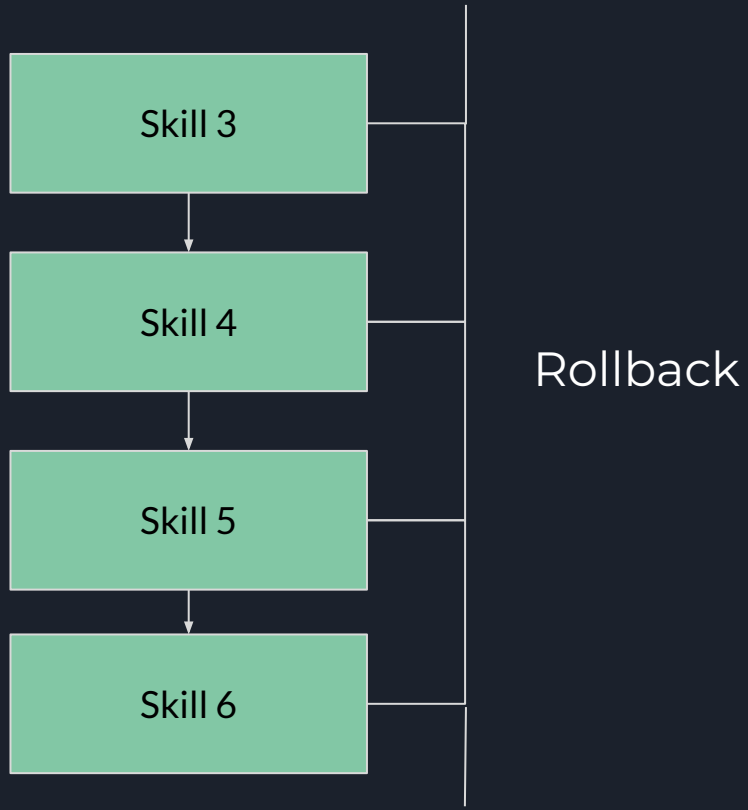
Skill-chained Process(SCP)



Skill-chained Process(SCP)



Skill-chained Process(SCP)



AI agent와 사람의 명확한 경계 .. ?

Cursor 를 사용하면서 느낀점:

1. 특정 객체에 기능과 책임이 과도하게 집중되는 문제가 발생
    a. 낮은 응집도, 높은 결합도로 유지보수 및 기능 확장 어려움
2. 테스트 코드 품질이 불안정함
    a. 예외 상황 및 경계 조건 누락 가능
3. AI Agent가 이해한 개발 흐름과 개발자가 의도한 SDLC/OOAD 과정이 다를 수 있음
    a. 매 단계마다 배경 설명과 맥락 정렬 필요
    b. 반복적인 프롬프트 수정으로 시간 소요 증가
4. AI agent는 명확한 이해와 정확한 프롬프트가 주어지면 잘한다. (특히 Implementation)

AI agent를 어떻게 이용해야 할까?

1. AI - agent는 OOAD 개발방법론에 대한 전체 이해가 개발자와 맞아야 한다.

   => 전 과정을 skill 로 분해하여 각 단계에 대한 이해와 task를 명시

   => 개발자가 정의한 skill과 이전 skill 출력물만을 입력으로 사용

2. AI - agent는 새로운 이해관계자로 요구사항 추출 과정에 함께 참여한다.
3. AI - agent에 대한 역할을 절제

   => rule file을 통해 각 skill 별 권한 통제, 최종 승인 절차는 반드시 개발자에게

1. rule 설계

1. 전체 프로세스 개요
   a. 총 12개의 skill process에 대한 이해
   b. skill 범위 통제
2. AI Agent는 모든 Skill에서 다음 흐름을 따라야 한다.
   a. 승인된 이전 Skill 산출물만 읽는다.
   b. 현재 Skill 프롬프트를 읽는다.
   c. 현재 Skill만 수행한다.
   d. 현재 Skill에 대해 사전에 정의된 산출물을 생성한다.
   e. 작업을 멈추고 Human Review를 기다린다.
   f. 인간 검토자의 승인이 있어야 다음 Skill로 진행한다.

1. rule 설계

3. Human review gate

전 skill 과정에서 산출물 간 C&C가 유지되는지

다음 스킬로 가긴 위한 승인 절차/ 직접 산출물 수정

4. 변경 및 rollback 규칙

AI-agent는 이전에 승인된 산출물을 정해진 skill 구조에 맞게 다시 수정해야한다.

문제점 혹인 변경된 요구사항에 대해 인간이 직접 rollback skill을 지정

Rollback이 발생하면 영향을 받은 skill과 그 이후의 모든 skill은 새로 수정된 산출물 기준 다시 waterfall

1. rule 설계

5. 설계 품질 규칙:

1. 하나의 class가 너무 많은 책임을 가지고 있는지, responsibility를 가지는지
2. coupling이 최소화 되었고 cohesion이 높은지?
3. 테스트 품질 규칙:
    a. 테스트 case는 positive, negative test case를 전부 포함

2. skill 설계

각 skill을 설계시 다음 내용을 포함한다

- 해당 단계의 목적과 범위 정의
- 읽어야 할 문서, 우선순위, 충돌 처리기준
- 생성 , 수정해야 할 산출물 목록
- 해당 단계에서 하면 안 되는 작업 명시

2. skill 설계

stage 01 - system requirements

읽기 규칙, 해당 단계의 목적 제시

사람과 에이전트의 맥락공유를 서로 질문과 답변을 통해 상호작용하며 진행.

1. 시스템 요구사항

해당 단계 시작 전 rule 파일을 확인한다.

해당 단계에 속한 작업과 프롬프트 마다 이 파일을 확인한다.

해당 단계는 시스템요구사항의 파악과 도출을 목적으로 한다.

해당 과정은 플랜모드로, 사용자와 질문과 답변으로 상호작용 하며 진행한다.

1. 프로그램의 목표, 목적, 뭐하는 프로그램인지
2. 이해 관계자 파악
    a. 프로그램의 타겟 사용자
    b. 유지보수 주체 - 본인이 유지보수하는지 업체등에 외주를 맡겨서 유지보수하는 지 파악
    c. 프로그램의 의뢰자
    d. 프로그램의 개발 주체
3. 해당 프로그램의 선제적 요구사항, 제한사항 (PFR) 입력
4. 해당 프로그램의 시스템 범위
    a. 시스템의 제외 범위: 방향성을 잃지 않기 위해 해당 프로그램이 다룰 범위를 제한하기 위한 질문
    b. 시스템의 포함 범위와 요소: 시스템이 포함해야할 요소(ex: 특정 객체, 특정 역할을 하는 함수 포함)

해당 과정이 끝나면 질문과 답변을 정리하여 사용자에게 보여주고, 보충이 필요하다고 판단되거나 보충이 필요한 내용이 있다면 사용자에게 추가 질문을 한다. 보충 질문이 전부 완료된 후, 다시 정리하여 사용자에게 보여주고 사용자에게 허가를 받는다.

해당 단계에서 도출된 내용을 정리한 마크다운 파일을 만들어 지정된 경로에 저장한다.

해당 단계의 산출물이 있다면, 산출물을 만들어 지정된 경로에 저장한다.

2. skill 설계

2. 기능 / 비기능 요구사항

해당 단계 시작 전 rule 파일을 확인한다.

해당 단계에 속한 작업과 프롬프트 마다 이 파일을 확인한다.

해당 단계는 기능/비기능 요구사항의 파악과 도출을 목적으로 한다.

1단계에서 확립된 정보들을 바탕으로 하여 작업한다.

해당 과정은 플랜모드로, 사용자와 질문과 답변으로 상호작용 하며 진행한다.

기능 요구사항

1. 핵심 사용자 시나리오 - 프로그램 이용자가 이 프로그램을 사용할때의 행동흐름을 물어본다.
    a. 1단계에서 정의한 프로그램의 목적에 따라 프로그램의 쓰임이 여러 종류 일 수 있으므로 정의된 주제에 따라 에이전트는 사용자에게 해당 질문을 여러번 한다.
        i. 해당 질문을 반복 중일때 시나리오가 더 있는지도 물어본다.
        ii. 만약 사용자로부터 더 이상의 시나리오가 없어도 된다는 답변이 나오면 다음 질문으로 넘어간다.
    b. 분리하여 관리 가능한 시나리오가 있다면, 에이전트는 시나리오를 분리 후 결과를 보여주고 사용자에게 허가/ 비허가/ 수정 요청으로 반응을 물어본다.
        i. 비허가 시 - 분리 전 상태를 유지한다
        ii. 수정 요청 시 - 어느 부분을 어떻게 수정하면 좋을지 사용자의 의도를 물어본다.
2. 반드시 포함해야하는 객체 확인 - 사용자가 프로그램에 사용하기로 의도한 객체를 준비해 놓은게 있는지 확인하기 위함.
3. 시나리오를 기반으로 한 핵심 비즈니스 로직 정리 및 제안
    a. 사용자가 제시한 비즈니스 로직이 파악되면 해당 비즈니스 로직을 이용, 파악되지 않는다면 제안하여 사용자의 의도와 개발 방향성을 파악함.
    b. 비즈니스 로직은 여러개가 될 수 있음
4. 해당 비즈니스 로직의 예외 처리 (<< 유스케이스 단계에서 대체 기능 요

해당 과정이 끝나면 질문과 답변을 정리하여 사용자에게 보여주고, 보충이 필요하다고 판단되거나 보충이 필요한 내용이 있다면 사용자에게 추가 질문을 한다. 보충 질문이 전부 완료된 후, 다시 정리하여 사용자에게 보여주고 사용자에게 허가를 받는다.

정리된 내용을 바탕으로 기능요구사항을 산출하고, 사용자에게 확인 및 허가를 요청한다. 허가되면 FR.md에 저장한다.

stage 02 - FR and NFR

기능 요구사항

2. skill 설계

stage 02 - FR and NFR

비기능 요구사항

비기능 요구사항

0. 비기능 요구사항에 관련된 브레인스토밍을 진행하기 전에, iso 25010 표준에 따른 소프트웨어 퀄리티 종류 사진을 사용자에게 제시한다.

SOFTWARE PRODUCT QUALITY								
FUNCTIONAL SUITABILITY	PERFORMANCE EFFICIENCY	COMPATIBILITY	INTERACTION CAPABILITY	RELIABILITY	SECURITY	MAINTAINABILITY	FLEXIBILITY	SAFETY
FUNCTIONAL COMPLETENESS	TIME BEHAVIOUR	CO-EXISTENCE	APPROPRIATENESS RECOGNIZABILITY	FAULTLESSNESS	CONFIDENTIALITY	MODULARITY	ADAPTABILITY	OPERATIONAL CONSTRAINT
FUNCTIONAL CORRECTNESS	RESOURCE UTILIZATION	INTEROPERABILITY	LEARNABILITY	AVAILABILITY	INTEGRITY	REUSABILITY	SCALABILITY	RISK IDENTIFICATION
FUNCTIONAL APPROPRIATENESS	CAPACITY		OPERABILITY	FAULT TOLERANCE	NON-REPUDIATION	ANALYSABILITY	INSTALLABILITY	FAIL SAFE
			USER ERROR PROTECTION	RECOVERABILITY	ACCOUNTABILITY	MODIFIABILITY	REPLACEABILITY	HAZARD WARNING
			USER ENGAGEMENT		AUTHENTICITY	TESTABILITY		SAFE INTEGRATION
			INCLUSIVITY		RESISTANCE			
			USER ASSISTANCE					
			SELF-DESCRIPTIVENESS					

iso25000.com

1. 성능 관련 질문
2. 백업 관련 질문
3. 사용성 - ui,ux, 시뮬레이터 등의 형식 지정
4. 보안
5. 유지보수성
6. 라이선스
7. 외부 프로그램 연결
8. 완성 데드라인
9. 프로그램 운영 예산

등과 같은 질문을 프로그램의 구조와 특성에 맞게 에이전트가 질문하여 비기능 요구사항을 확립한다.

2. skill 설계 - stage 03 - usecase

3. Use Case

해당 단계 시작 전 rule 파일을 확인한다.

해당 단계에 속한 작업과 프롬프트 마다 이 파일을 확인한다.

해당 단계는 usecase의 파악과 brief format과 유스케이스 다이어그램의 도출을 목적으로 한다.

이전 단계인 1,2 단계에서 확립된 정보들을 기반으로 하여 작업한다.

해당 과정은 플랜모드로, 사용자와 질문과 답변으로 상호작용 하며 진행한다.

1. 지금까지의 과정에서 에이전트가 판단한 액터 제시
2. 액터 추가 식별, 수정을 위한 질문한다.
    a. 외부 api
    b. 외부 하드웨어
    c. 사용자
    d. 시스템 운영자
    e. 연결된 별도 프로그램
    f. 에이전트가 제시한 액터 중 수정할 부분
3. 답변을 종합하여 정리하고, 사용자의 허가를 요청
4. 액터를 FR에 매칭한 결과를 보여주고, 유스케이스를 brief 포맷으로 만들어 리스트나 표로 제시한다.
    a. 해당 포맷에는 유스케이스 일련번호, 이름, 관여하는 액터, 유스케이스에 대한 설명과 포함된 기능 요구사항 일련번호를 포함해야 한다.
5. 사용자의 의도에 따라 추가할 유스케이스 추가하는 작업
    a. 해당 단계에서 추가되는 유스케이스도 액터 지정부터 진행한다.
    b. 해당 과정에서 변경사항이 있다면 변경, 삭제, 추가 내용을 강조하여 표시하고 4번 과정 처럼 표나 리스트로 다시 제시하여 사용자의 허가를 요청한다.
6. 유스케이스 간 관계 확인을 위한 질문
    a. 유스케이스 중에서 <include>등의 관계를 적용 할 수 있는 것을 에이전트가 판단하여 제시하거나 의도한 게 있는지 사용자에게 질문한다.
    b. 해당 과정에서 변경사항이 있다면 변경, 삭제, 추가 내용을 강조하여 표시하고 4번 과정 처럼 표나 리스트로 다시 제시하여 사용자의 허가를 요청한다.
7. 유스케이스 다이어그램 제작
    a. 수정을 거치고 사용자의 허가를 받은 brief 포맷 유스케이스를 바탕으로 유스케이스 다이어그램을 만든다.
    b. 다이어그램은 플랜트uml 코드와 이미지 파일 두가지를 다 만들어야한다.
    c. 생성 완료 후, 다이어그램의 이미지를 사용자에게 보여주고 피드백을 받는다.

해당 과정이 끝나면 질문과 답변을 정리하여 사용자에게 보여주고, 보충이 필요하다고 판단되거나 보충이 필요한 내용이 있다면 사용자에게 추가 질문을 한다. 보충 질문이 전부 완료된 후, 다시 정리하여 사용자에게 보여주고 사용자에게 허가를 받는다.

해당 단계에서 도출된 내용을 정리한 마크다운 파일을 만들어 지정된 경로에 저장한다.

해당 단계의 산출물을 만들어 지정된 경로에 저장한다.

2. skill 설계 - stage 04 - 세 부 usecase

4. 세부 Use Case - 디테일 포맷으로 변경

해당 단계 시작 전 rule 파일을 확인한다.

해당 단계에 속한 작업과 프롬프트 마다 이 파일을 확인한다.

해당 단계는 이전 단계에서 제작한 usecase brief format을 기반으로 자세한 유스케이스 문서 도출을 목적으로 한다.

이전 단계인 1,2,3 단계에서 확립된 정보들을 기반으로 하여 작업한다.

해당 과정은 플랜모드로, 사용자와 질문과 답변으로 상호작용 하며 진행한다.

0. 모든 유스케이스가 사용자에게 허가 될때 까지 1~4를 반복한다.
1. 사용자에게 유스케이스 중 어떤걸 대상으로 만들지 지정 요청
    a. 기능이 유사하거나, 정 반대로 작동하는 유스케이스 (예: 전원 키기, 끄기) 등은 예외로 한번에 작업 가능하다.
2. 에이전트는 이전 단계들을 기반으로 하여 상세 명세 중 일부를 작성하여 제시한다.
    a. 사전 조건
    b. 사후 조건
    c. 정상 흐름
3. 사용자에게 해당 부분의 수정할 부분이 있는지 검토을 요청하고, 사용 허가를 받는다.
4. 정상 흐름을 확정한 후, 대체, 예외, 오류발생시의 흐름을 제시하고 수정사항 피드백과 승인을 받는다.
    a. 에이전트가 추론한 내용 중 로직에 결정적인 예외 상황이나 정책에 대해서는 사용자에게 반드시 질문해야하고, 답변을 객관식 혹은 주관식으로 받은 후 완성한다.
    b. 대체 흐름
    c. 오류, 예외 흐름
5. 모든 유스케이스 작성이 끝나면 정리하여 표등의 형식으로 정라하여 사용자에게 보여준다.

해당 과정이 끝나면 질문과 답변을 정리하여 사용자에게 보여주고, 보충이 필요하다고 판단되거나 보충이 필요한 내용이 있다면 사용자에게 추가 질문을 한다. 보충 질문이 전부 완료된 후, 다시 정리하여 사용자에게 보여주고 사용자에게 허가를 받는다.

해당 단계에서 도출된 내용을 정리한 마크다운 파일을 만들어 지정된 경로에 저장한다.

해당 단계의 산출물을 만들어 지정된 경로에 저장한다.

2. skill 설계
stage 05 - SSD,stage 06 - Domain Model
stage 07 - SD, stage 08 - Class Diagram

1. Purpose → 이 Stage가 뭘 하는지
2. Input Contract → 이전 Stage 산출물(docs/)에서 무엇을 읽을지
3. Execution → 실제 작업 순서 (Step 1~N)
4. Output → 만들 파일·형식 정의
5. ID Conventions → OP-ID, DC-ID 등 번호 규칙
6. Do Not Do → 하지 말 것
7. Validation → 완료 전 체크리스트

2. skill 설계

stage 05 - SSD, stage 06 - Domain Model

stage 05 - SSD

- 액터와 :System 간 operation을 SSD로 작성한다. 시스템 내부 객체는 표현하지 않는다

stage 06 - Domain Model

- 요구사항·유스케이스·05 SSD operation을 반영한 도메인 모델을 작성한다

2. skill 설계
stage 05 - SSD, stage 06 - Domain Model : Input Contract

2. Input Contract

공통 읽기 규칙

1. `docs/{StageFolder}/` 전체 탐색
2. `result.md`, 본문 `.md`, `diagrams/*.puml` 읽기
3. `docs/05_SystemSequenceDiagram/result.md` **Inputs Used**에 `Stage | Path | Files Read | Notes` 기록
4. 파일명이 달라도 동일 역할 산출물이 있으면 읽기

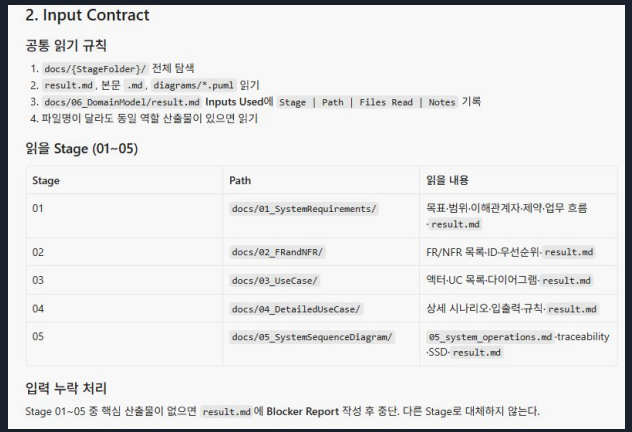
읽을 Stage (01~04)

Stage	Path	읽을 내용
01	`docs/01_SystemRequirements/`	목표·범위·이해관계자·제약·업무 흐름 · `result.md`
02	`docs/02_FRandNFR/`	FR/NFR 목록·ID·우선순위· `result.md`
03	`docs/03_UseCase/`	액터·UC 목록·다이어그램· `result.md`
04	`docs/04_DetailedUseCase/`	상세 시나리오·입출력·규칙· `result.md`

입력 누락 처리

Stage 01~04 중 핵심 산출물이 없으면 `result.md`에 **Blocker Report** 작성 후 중단. 다른 Stage로 대체하지 않는다.

stage 05



2. Input Contract

공통 읽기 규칙

1. `docs/{StageFolder}/` 전체 탐색
2. `result.md`, 본문 `.md`, `diagrams/*.puml` 읽기
3. `docs/06_DomainModel/result.md` **Inputs Used**에 `Stage | Path | Files Read | Notes` 기록
4. 파일명이 달라도 동일 역할 산출물이 있으면 읽기

읽을 Stage (01~05)

Stage	Path	읽을 내용
01	`docs/01_SystemRequirements/`	목표·범위·이해관계자·제약·업무 흐름 · `result.md`
02	`docs/02_FRandNFR/`	FR/NFR 목록·ID·우선순위· `result.md`
03	`docs/03_UseCase/`	액터·UC 목록·다이어그램· `result.md`
04	`docs/04_DetailedUseCase/`	상세 시나리오·입출력·규칙· `result.md`
05	`docs/05_SystemSequenceDiagram/`	`05_system_operations.md`·traceability ·SSD· `result.md`

입력 누락 처리

Stage 01~05 중 핵심 산출물이 없으면 `result.md`에 **Blocker Report** 작성 후 중단. 다른 Stage로 대체하지 않는다.

stage 06

2. skill 설계
stage 05 - SSD, stage 06 - Domain Model : Execution

3. Execution

Step 1. Preflight

```
[ ] Stage 01~04 읽고 Inputs Used 기록
[ ] SSD 대상 UC 선정
```

Step 2. SSD 대상 유스케이스 선정

`docs/03_UseCase/` 와 `docs/04_DetailedUseCase/` 를 기준으로, 상세 시나리오가 있는 모든 UC를 SSD 대상으로 선정한다.

각 UC에 대해 기록:

- UC-ID, 이름, 주 액터, SSD 파일명

Step 3. Operation 식별

각 UC 흐름(기본/대안)에서 액터→System 메시지를 추출.

각 operation:

- Operation ID (예: OP-UC01-001)
- 이름 (동사구, 업무 언어: `submitOrder`, `cancelReservation`)
- 연결 UC-ID, 흐름 단계
- 입력 파라미터 (업무 타입명)
- 출력/반환 (업무 타입명 또는 void)

금지: `save()`, `update()`, `getData()` 등 업무 의미 없는 CRUD명

Step 4. SSD 다이어그램 작성 (PlantUML)

파일 규칙:

```
docs/05_SystemSequenceDiagram/diagrams/ssd_UC{ID}.puml
```

한 UC에 기본+대안 흐름이 크면:

- `ssd_UC01_main.puml`, `ssd_UC01_alt1.puml` 등 분리
- 또는 `alt` / `opt` / `loop` fragment 사용

PlantUML 템플릿:

```
@startuml ssd_UC01
title SSD — UC-01: Place Order
```

stage 05

3. Execution

Step 1. Preflight

```
[ ] Stage 01~05 읽고 Inputs Used 기록
```

Step 2. 도메인 모델 본문 작성

`docs/06_DomainModel/06_domain_model.md` 한 파일에 아래 내용을 작성한다.

§1 Domain Terms — 핵심 명사·동사·역할 (Term ID, Term, Definition, Notes). 05 operation·타입명과 정렬.

§2 Concepts — 업무상 중요 개념 (Concept ID, Name, Type, Definition). 05 SSD 입출력 타입에서 도출.

§3 Associations — 개념 간 관계 (From, To, Multiplicity, Role)

§4 Business Rules — (Rule ID, Rule, Applies To, Notes)

Step 3. PlantUML 도메인 모델 다이어그램 작성

`docs/06_DomainModel/diagrams/domain_model.puml` 작성.

PlantUML 규칙:

```
@startuml domain_model
skinparam classAttributeIconSize 0
```

stage 06

2. skill 설계
stage 05 - SSD, stage 06 - Domain Model : Output

Step 5. System Operation Catalog 작성

`docs/05_SystemSequenceDiagram/05_system_operations.md` — operation 목록표:

```
Op ID | Operation Name | UC-ID | Flow | Input | Output | Preconditions | Postconditions
```

Step 6. 추적성 매트릭스 작성

`docs/05_SystemSequenceDiagram/05_traceability_matrix.md` 에 이 Stage의 추적성 전부를 기록한다.

```
UC / Detailed Flow → SSD file → System Operation → FR/NFR/BR
```

포함 섹션(4.2 참조):

- OP ↔ UC ↔ SSD 파일 ↔ upstream(04/02) 매핑
- Traceability Gap (`TG-001` ...)

Step 7. result.md 작성

섹션 4.3·4.4 참조. 가정·미결 질문은 있을 때만 4.4에 추가.

Step 8. 산출물 저장

`docs/05_SystemSequenceDiagram/` 아래 저장.

4. Output

4.1 필수 산출물 (3개 + 다이어그램)

```
docs/05_SystemSequenceDiagram/result.md
docs/05_SystemSequenceDiagram/05_system_operations.md
docs/05_SystemSequenceDiagram/05_traceability_matrix.md
docs/05_SystemSequenceDiagram/diagrams/ssd_*.puml
```

stage 05

Step 4. 추적성 매트릭스 작성

`docs/06_DomainModel/06_traceability_matrix.md` 에 이 Stage의 추적성 전부를 기록한다.

```
FR/NFR/UC/OP → Domain Term → Concept → Rule
```

포함 섹션(4.3 참조):

- Term / Concept / Rule / Association ↔ upstream(01~05) 매핑
- Traceability Gap (`TG-001` ...)

Step 5. result.md 작성

섹션 4.4·4.5 참조.

Step 6. 산출물 저장

모든 파일을 `docs/06_DomainModel/` 아래에 저장한다. 기존 파일 덮어쓰기 전 변경 요약을 `result.md` 에 남긴다.

4. Output

4.1 필수 산출물 (3개 + 다이어그램)

```
docs/06_DomainModel/result.md
docs/06_DomainModel/06_domain_model.md
docs/06_DomainModel/06_traceability_matrix.md
docs/06_DomainModel/diagrams/domain_model.puml
```

stage 06

2. skill 설계

stage 07 - SD, stage 08 - Class Diagram

stage 07 - SD

- 05 System Operation 하나당 시스템 내부 객체 간 메시지를 상세 Sequence Diagram으로 작성한다.

stage 08 - Class Diagram

- 06 Domain Model과 07 Sequence Diagram을 바탕으로 Class Diagram을 작성한다.

2. skill 설계

stage 07 - SD, stage 08 - Class Diagram：Execution

3. Execution

Step 1. Preflight

```
[ ] Stage 01~06 읽고 Inputs Used 기록
```

Step 2. Operation별 분석

05 `05_system_operations.md` 의 OP-ID마다:

1. UC-ID, SSD 파일, 04 흐름 단계 확인
2. 입·출력 타입 → 06 domain concept 매핑
3. 적용 BR/불변식 확인
4. 필요 lifeline(객체) 식별

Step 3. Participant catalog (객체·lifeline 정의)

Step 2에서 식별한 객체(lifeline)를 `docs/07_SequenceDiagram/07_participants.md` 에 기록:

```
Participant ID | Name | Stereotype | Responsibility | First OP-ID
```

Stereotype: `<<boundary>>` `<<control>>` `<<entity>>` `<<service>>` `<<external>>` — 06 도메인 용어와 이름 일치

Step 4. Sequence 다이어그램 작성 (PlantUML)

05 `05_system_operations.md` 의 OP-ID마다 `diagrams/seq_{OP-ID}.puml` 작성.

```
@startuml seq_OP-UC01-001
actor Customer
participant ":OrderUI" as UI <<boundary>>
participant ":OrderService" as Svc <<control>>
participant ":Order" as Order <<entity>>

Customer -> UI : submitOrder(cartId, paymentInfo)
UI -> Svc : submitOrder(cartId, paymentInfo)
Svc -> Order : createFromCart(cartId)
Order --> Svc : order
Svc --> UI : orderConfirmation
UI --> Customer : orderConfirmation
@enduml
```

- 첫 액터→boundary 메시지 = 05 SSD operation 이름·시그니처 유지
- `alt` / `opt` / `loop` 로 04·05 분기 반영
- 구현 코드·SQL 문자열 금지

stage 07

3. Execution

Step 1. Preflight

```
[ ] Stage 01~07 읽고 Inputs Used 기록
```

Step 2. 클래스 모델 본문 작성

`docs/08_ClassDiagram/08_class_model.md` 한 파일에 아래 내용을 작성한다.

§1 Classes — `07_participants.md` 를 Class로 승격·병합

```
Class ID | Class Name | Stereotype | Responsibility Summary
```

- 동일 개념 다른 이름 → 06 `06_domain_model.md` §1·§2 기준으로 하나로 통합
- <

  > / <

  > / <

  > / <

  > 구분 유지

§2 Attributes — 클래스별 속성 (이름, 타입, 가시성). DB·ORM 표기 금지.

§3 Operations — 클래스별 연산 시그니처

```
Class ID | Operation | Parameters | Return | Visibility
```

- 07 sequence 메시지 → 05 system operation → 06 domain 행위 순으로 도출

§4 Associations — (From, To, Type, Multiplicity, Role)

Step 3. PlantUML Class Diagram 작성

`docs/08_ClassDiagram/diagrams/class_diagram.puml` 작성. 복잡할 때만 `class_diagram_{layer}.puml` 로 분할.

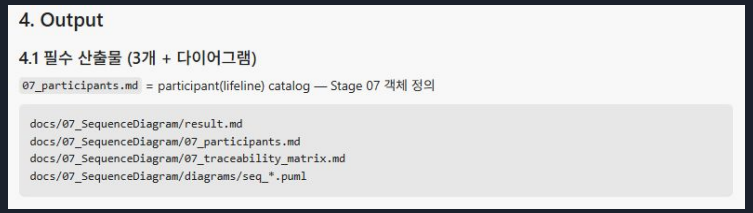
PlantUML 템플릿:

```
@startuml class_diagram_domain
skinparam classAttributeIconSize 0
```

stage 08

2. skill 설계

stage 07 - SD, stage 08 - Class Diagram : Output



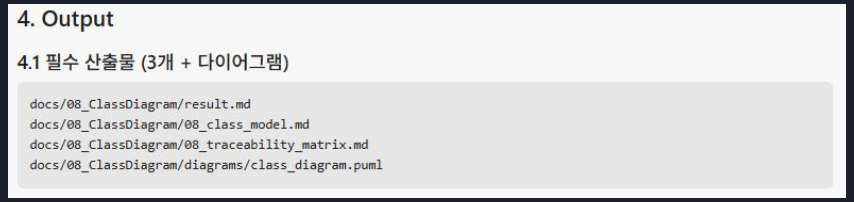
4. Output

4.1 필수 산출물 (3개 + 다이어그램)

`07_participants.md` = participant(lifeline) catalog — Stage 07 객체 정의

```
docs/07_SequenceDiagram/result.md
docs/07_SequenceDiagram/07_participants.md
docs/07_SequenceDiagram/07_traceability_matrix.md
docs/07_SequenceDiagram/diagrams/seq_*.puml
```

stage 07



4. Output

4.1 필수 산출물 (3개 + 다이어그램)

```
docs/08_ClassDiagram/result.md
docs/08_ClassDiagram/08_class_model.md
docs/08_ClassDiagram/08_traceability_matrix.md
docs/08_ClassDiagram/diagrams/class_diagram.puml
```

stage 08

2. skill 설계
stage 09 - Generate Code

09 Generate Code SKILL

1. SKILL 목적

OOAD 파이프라인의 9단계: Generate Code를 수행한다.
4단계 세부 UseCase, 5단계 도메인 모델, 6단계 System Sequence Diagram, 7단계 시퀀스 다이어그램, 8단계 클래스 다이어그램을 활용하여 코드를 제작한다.

2. 사용 시점

8단계: 클래스 다이어그램이 완료된 시점에 시작한다.

3. 사용하지 않는 시점

- 이 파이프라인 이외의 코딩
- 10단계: Unit Test 작성
- 다이어그램의 변경이 필요한 경우
- 사람의 리뷰 없이 승인 처리

4. 요구되는 input

2단계: FR/NFR의 요구 명세
4단계: 흐름과 분기 시점
5단계: Domain Model
7단계: Sequence Diagram
8단계: Class Diagram
docs/

5. Output

산출물은 docs/09_GenerateCode에 09_result.md를 생성한다.
코드 산출물은 src/, include/, CMakeLists.txt 로 저장된다.

6. 사람의 역할

승인 사항을 사람이 통과시켜야 완료된다.

- 코드가 다이어그램과 일치하는지
- 생성 후 컴파일을 확인하여 빌드가 제대로 수행되는지

2. skill 설계
stage 10 - Unit Test

10 Unit Test SKILL

1. SKILL 목적

OOAD 파이프라인의 10단계: Unit Test를 수행한다.
9단계 생성된 코드를 바탕으로 2단계 FR/NFR, 3단계 UseCase, 4단계 세부 UseCase,
7단계 시퀀스 다이어그램, 8단계 클래스 다이어그램을 활용하여 GoogleTest 단위 테스트로 변환한다
분할 SKILL 패키지로, 10단계의 작업은 서브 SKILL들 순서대로 수행한다

10_1_test_case_design
테스트 케이스로 변환하여 코드 작성
10_2_coverage_review
커버리지 측정
10_3_final
모든 산출물에 대한 결과물 작성.
사람이 승인해야 하는 항목 작성.

2. 사용 시점

9단계: 코드 생성이 완료된 시점에 시작한다.

3. 사용하지 않는 시점

신규 코드 생성
11단계: System Test 작성
12단계: CICD 환경 구성
사람의 리뷰 없이 승인 처리

4. 요구되는 input

2단계: FR/NFR의 요구 명세
4단계: 흐름과 분기 시점
7단계: 코드 흐름
8단계: 테스트 단위 그룹
9단계: 빌드 가능 코드
src/
CMAKElists.txt
include/

5. 서브 SKILL

10_1_test_case_design: UT 설계 및 제작
10_2_coverage_review: 커버리지 측정, 90% 이상을 목표로 함
10_3_final: 최종 통합, 승인 사항

6. Output

산출물은 docs/10_UnitTest에 10test_case.md, 10_coverage_report.md, 10_result.md를 생성한다.
코드 산출물은 test/Mock.h, *_test.cpp, CMakeLists.txt 로 저장된다.

7. 사람의 역할

10_3 단계의 final 승인 사항을 사람이 통과시켜야 완료된다.

FR/NFR 요구 명세
커버리지 exclusion 사유

2. skill 설계
stage 10 - Unit Test

10_1_test_case_design SKILL

1. SKILL 목적

작성 완료한 2단계 FR/NFR, 3단계 UseCase, 4단계 세부 UseCase, 7단계 시퀀스 다이어그램, 8단계 클래스 다이어그램을 활용하여 GoogleTest 단위 테스트로 변환한다.

2. 사용 시점

10단계 Unit Test의 서브 SKILL 1로 사용한다.

3. 사용하지 않는 시점

11단계 SystemTest

4. 실행 절차

2단계 FR/NFR에서 테스트 가능한 동작 추출.
4단계 상세 Use Case에서 흐름, 분기를 Unit Test Case로 변환
Mock함수 제작
센서 값에 대해서 경계값 케이스를 추가
상호 작용에 대해서 7단계 시퀀스 다이어그램에 있는 호출순서와 호출 횟수를 활용하여 케이스 추가
테스트 케이스들에 대해서 기록: TC-UT
테스트하지 않는 항목에 대해서 사유 작성

5. 고려 사항

기대결과는 반드시 관측 가능해야 한다.
EXPECT_CALL의 호출 횟수/순서 제약은 시퀀스 다이어그램에 따라서 제작한다.
많은 케이스 수가 주 목적이 아니다. 중복 케이스는 합친다.

6. Output

산출물을 docs/10_UnitTest에 10test_case.md로 저장한다
코드 산출물을 test/Mock.h, *_test.cpp, CMakeLists.txt 로 저장한다.

2. skill 설계
stage 10 - Unit Test

10_2_coverage_review SKILL

1. SKILL 목적

라인/분기 커버리지에 따라 테스트의 양과 질을 함께 측정한다.
라인 커버리지 90%를 목표로 삼는다.

2. 사용 시점

10단계 Unit Test의 서브 SKILL 2로 사용한다.

3. 사용하지 않는 시점

11단계 SystemTest

4. 실행 절차

커버리지 측정
파일별 수치를 표로 만들고 비교한다.
미커버 코드를 분류하고, 근거를 작성한다.
특이사항들에 대해 수정 필요 / 권고로 분류하여 작성한다
수정 필요 시 10_1_test_case_design로 돌아간다.

5. 고려 사항

라인-분기 커버리지 격차가 크면(예: 라인 90% / 분기 60%대) 그 자체가 특이사항이다.
exclusion 추가는 사유와 함께 제안만 한다.

6. Output

산출물을 docs/10_UnitTest에 10_coverage_report.md로 저장한다

2. skill 설계
stage 10 - Unit Test

10_3_final SKILL

1. SKILL 목적

Stage 10의 모든 공식 산출물을 검사하여 result.md 보고서를 작성한다.
사람의 승인 시 체크 사항을 작성한다. 승인은 사람만 할 수 있다.

2. 사용 시점

10단계 Unit Test의 서브 SKILL 3(마지막)으로 사용한다.

3. 사용하지 않는 시점

11단계 SystemTest
보고서를 제외한 신규 산출물 생산

4. 실행 절차

테스트 케이스/커버리지를 점검하고 요약하여 보고서를 작성한다.

5. 고려 사항

TC-UT <-> 테스트 코드가 1:1로 대비되는가
커버리지 리포트의 exclusion이 문서와 일치하는가
테스트가 안된 항목에 대해서 N/A가 있는지 확인한다.
승인 확인 사항을 작성한다.

6. Output

산출물을 docs/10_UnitTest에 10_result.md로 저장한다

7. 사람의 역할

승인 확인 사항을 사람이 통과시켜야 완료된다.

2. skill 설계
stage 11 - System Test

11 System Test SKILL

1. SKILL 목적

OOAD 파이프라인의 11단계: System Test를 수행한다.
C++ RVC컨트롤러 + Python 시뮬레이터를 실제 기동해 요구사항 시나리오대로 행동하는지 검증한다

2. 사용 시점

10단계: Unit Test가 완료되고 시스템 수준의 검증이 필요할 때.

- 요구사항이 잘 검증되는지 확인
- 환경 결함 여부 확인

3. 사용하지 않는 시점

- 코드 오류 수정
- 커버리지 보충
- 사람의 리뷰 없이 승인 처리

4. 요구되는 input

2단계: FR/NFR의 요구 명세
4단계: 흐름과 분기 시점
9단계: 빌드 가능 코드
10단계: Unit Test 통과 여부

5. 실행 절차

- 각 시나리오마다 사전조건, 절차, 기대 결과, 통과 기준을 제작
- C++ RVC컨트롤러 프로세스 기동
- Python 시뮬레이터 프로세스 기동
- 할당된 고정 포트 사용전 검사
- 각 시나리오의 프로세스 종료 보장
- 로그 작성

6. 고려 사항

포트 충돌시 다른 포트 사용
PASS / FAIL로 시나리오 분류

7. Output

산출물은 docs/11_SystemTest에 11_SystemTestLog*.md, 11_result.md를 생성한다.

8. 사람의 역할

승인 사항을 사람이 통과시켜야 완료된다.

- 시각적으로 RVC가 잘 작동하는지 확인
- FR이 잘 구현되었는지 확인
- 시나리오에 따라서 로그가 잘 작성되었는지 확인
- 릴리즈가 가능한 상태인지

2. skill 설계
stage 12 - CI/CD

12 CICD 환경 구축

1. SKILL 목적

파이프라인의 12단계: CICD Enviroment를 수행한다.
Github Actions Workflow, SonarCloud 커버리지 연동, Python을 활용한 시뮬레이터 테스트 환경이 잘 구축되어 있는지 확인
모든 01~11단계의 산출물 점검

2. 사용 시점

11단계까지 모든 단계에 대해서 사용자가 전부 승인하였을 때

3. 사용하지 않는 시점

- 다이어그램 수정
- 코드 오류 수정
- 테스트 항목 수정

4. 요구되는 input

docs/의 모든 report

5. 실행 절차

- 최종 CMakeList.txt 작성
- Build: `cmake --build build .`
- Test: `cd build && ctest --output-on-failure .`
- Push: Actions 탭에서 초록불이 잘 나오는지 확인
- 완성된 파이프라인의 각 항목에 대해서 보고서 작성

6. 고려 사항

`ubuntu-latest` 가 버전 문제가 생기면 `ubuntu-22.04` 로 고정
pygame import 실패시 python 버전을 3.13으로 고정

7. Output

CMAKE 산출물은 메인폴더에 CMakeLists.txt를 작성한다.
산출물은 docs/12_CICD에 12_result.md를 생성한다.

8. 사람의 역할

SONAR_TOKEN 등록 및 SonarCloud 플랜 한도 확인
승인 사항을 사람이 통과시켜야 완료된다.

- Actions에서 초록불이 나오고 Unit Test를 전부 통과함

최종 평가

장점:

1. 단계별 책임이 명확하고 무분별한 진행을 막을 수 있다.
2. AI로 개발하면서 인간의 승인 단계를 통한 조절 및 통제가 가능
3. 산출물을 통해 잘 모르는 사람도 문서를 보고 이해할 수 있음
4. skill의 정의가 되면 어떤 OOAD개발에 사용할 수 있음

단점:

1. 초기 각 단계별 Skill 파일을 만드는 데 오랜 시간이 걸린다.
2. skill 별로 문서를 관리하다 보니 버전 추적하기 어려워 질 수 있다.

감사합니다.