문제를 보는 눈

퍼실리테이터 가이드

박정수 (Park Jeongsoo) <toracle@gmail.com>

목차

이 문서는 강의자/퍼실리테이터 전용입니다. 참가자에게 배포하지 마세요.

> 이 문서는 강의자/퍼실리테이터 전용입니다. 참가자에게 배포하지 마세요.

워크샵 개요

항목	내용
대상	풀스택 엔지니어 3-4년차
구성	8강 × 100분 (휴식 제외)
형식	강의 + 실습 혼합
인원	강당 6-16명 권장
준비물	워크시트 A/B 인쇄본, 화이트보드, 역할 카드(3강)

고정 케이스 등장인물

워크샵 전체에서 팀오더 케이스의 등장인물을 참조합니다. 이 인물들을 미리 숙지해두세요.

인물	역할	성격 특성	워크샵에서의 기능
민준	BizDev, 31세	기술 비전문가, 현장 감각 날카로움	모호한 요구사항을 가져오는 역할. 나쁜 의도가 아님
지연	풀스택 3년차, 28세	빠르게 만드는 것에 자신감	수강생이 자신을 투영하기 쉬운 인물. "솔루션 중독"의 대표
승현	풀스택 5년차, 32세	성찰적, 팀 시니어	이야기의 시점 인물. 학습의 진행을 보여주는 역할
현태	풀스택 1년차, 26세	막내, 당연한 것을 묻는다	4강부터 등장. 팀 내 다른 이해 수준을 드러내는 역할

> 퍼실리테이터 팁: 인물들을 비판하지 마세요. 지연이 바로 코딩하는 것, 현태가 헷갈리는 것은 수강생의 현재 모습과 같습니다. "틀렸다"가 아니라 "다른 방법이 있었다"는 프레임을 유지하세요.

강의 흐름 — 전체 지도

```
강    주제                          인식론적  성격
```

```
──────────────────────────────────────────────────────
1강  솔루션 중독                  자각 (Awareness)
2강  모호함의 해부학              어휘 습득 (Vocabulary)
3강  질문하는 자가 설계한다      기술 훈련 (Skill)
4강  지도를 그리기 전에           구조 파악 (Structure)
──────────────────────── ← 전환점 ────────────────────────
5강  안개 속에서 걷는 법          인식론 전환 (Paradigm Shift)
──────────────────────── ← Solution Space 진입 ──────────
6강  이 문제, 어디서 본 것 같은데  패턴 연결 (Connection)
7강  어디에 공을 들여야 하는가     판단 훈련 (Judgment)
8강  승현의 지도                      통합 (Integration)
──────────────────────────────────────────────────────
```

강 간 연속성 — 누적 구조

이 워크샵의 핵심 설계 원리는 _누적_입니다.

- _팀오더 케이스_는 1강의 한 문장에서 시작해서 8강까지 동일하게 씁니다
- _워크시트 B_는 수강생의 현업 케이스를 8강에 걸쳐 심화합니다
- 각 강의 과제는 다음 강의 전강 연결로 이어집니다
- 강들이 독립적이지 않습니다. 결강 시 다음 강 진행이 어렵습니다

핵심 메시지 — 워크샵 전체

수강생이 이 워크샵에서 가지고 가야 할 것 하나:

> "문제를 오래 보는 것이 해결책을 빠르게 만드는 방법이다."

Problem Space를 더 오래 탐색하면: * 핵심적으로 달성해야 할 것이 명확해진다 * 나중에 해도 되는 것이 보인다 * 하지 않아도 되는 것이 식별된다 * → 더 작고 더 정확하게 만들어서, 더 일찍 전달할 수 있다

Chapter 1. 솔루션 중독 — Problem Space를 건너뛰는 습관

1.1. 시간 배분

순서	내용	시간
	0. 전강 연결	5분
	1. 오프닝 도발 — 30초 실험	10분
	2. 개념 설명 (Einstellung, Presumptive Architecture, Problem/Solution Space)	20분
	3. 고정 케이스 실습 — 워크시트 A-1	30분
	4. 현업 케이스 슬롯 — 워크시트 B-1	20분
	5. 디브리프	10분
	6. 과제 카드 #1 배포	5분
	합계	100분

1.2. 강의 운영 주의사항

- ★ 1강은 자각(awareness) 강입니다. 옳고 그름이 없다는 분위기를 먼저 만드세요.
- ★ '30초 실험' 후 Solution Space 항목들을 칠판에 적을 때 비판하지 마세요. '이게 왜 Solution Space인지'를 함께 발견하게 하세요.
- ★ Presumptive Architecture가 나쁜 것이 아님을 명확히 하세요. '가정인 줄 알고 쓰는 것'과 '사실인 줄 알고 쓰는 것'의 차이.
- ★ Problem Space 질문이 적게 나온 그룹에게 '왜 안 나왔을까?'를 물어보세요. 수량 비교 자체가 학습 소재입니다.

1.3. 0. 전강 연결 (5분)

1강이므로 워크샵 전체 지도를 보여줍니다.

```
오늘 우리가 8강에 걸쳐 할 것:

"민준이 가져온 저 한 문장을 어떻게 다루어야 하는가"

— 빠르게 만들기 시작하는 것?
— 아니면 다른 무언가가 먼저 필요한가?
```

1.4. 1. 오프닝 도발 (10분) — "30초 실험"

슬라이드나 말 없이, 종이를 나눠주고 말합니다.

> "민준이 방금 이렇게 말했습니다.

> '팀 협업 기능 강화해주세요. 알림이랑 업무 배정 쪽이요. 다음 달 데모.'

> 지금 이 순간 머릿속에 떠오르는 것을 30초 안에 적어주세요."

30초 후 몇 명에게 읽어달라고 합니다.

전형적으로 나오는 것들:

```
"WebSocket으로 실시간 알림"
"Kafka 써서 이벤트 처리하면"
"알림 테이블 따로 만들고"
"Firebase Cloud Messaging이"
"업무 배정은 assignee 컬럼 추가하면"
```

이것을 칠판에 적어두고 말합니다.

> "여기 적힌 것들, 전부 Solution Space입니다.

> Problem Space에 있는 것이 하나라도 있나요?"

짧은 침묵. 이것이 이 강의 전체 논점입니다.

1.5. 2. 개념 설명 (20분)

개념 1 — Einstellung Effect (10분)

익숙한 해결책이 있으면 더 나은 해결책이 보이지 않는 현상. 경험이 많을수록 이 효과가 더 강하게 작동합니다. 3-4년차가 특히 위험한 이유를 명시적으로 이야기하세요.

개념 2 — Presumptive Architecture (5분)

모든 개발자는 무의식적인 아키텍처 디폴트를 가지고 있습니다. 이것 자체는 나쁘지 않습니다. '가정인 줄 알고 쓰는 것'과 '사실인 줄 알고 쓰는 것'의 차이를 강조하세요.

개념 3 — Problem Space vs Solution Space (5분)

두 공간 모두 반드시 거쳐야 합니다. 핵심: 이동이 _의식적_이어야 한다. 자동으로 넘어가는 것이 문제입니다.

1.6. 소그룹 공유 가이드

```
비교 포인트
──────────────

"같은 요구사항인데 Problem/Solution 분류가 다른가?"
  → 그 차이 자체가 해석자 간 모호함의 실례입니다.

"Presumptive Architecture가 나쁜 건가요?"
  → 아닙니다. 가정인 줄 알고 쓰는 것이 핵심입니다.

"Problem Space 질문이 적게 나왔다"
  → 왜 안 나왔을까요? 어떤 종류의 질문이 막혔나요?
```

1.7. 5. 디브리프 가이드 (10분)

```
예상 저항과 대응
────────────────

"분류가 애매한 것들이 있었다"
  → 좋습니다. 그 경계 지점이 다음 강의 주제입니다.

"Problem Space 질문이 생각보다 많이 안 나왔다"
  → 왜 안 나왔을까요? 어떤 종류의 질문이 막혔나요?

"다 알고 있다고 생각했는데 막상 쓰니 모르는 게 많았다"
  → 이것이 오늘의 핵심 체험입니다.

"Presumptive Architecture가 나쁜 건가요?"
  → 아닙니다. 가정인 줄 알고 쓰는 것과
     사실인 줄 알고 쓰는 것의 차이입니다.
```

Chapter 2. 모호함의 해부학 — 요구사항을 분해하는 언어

2.1. 시간 배분

순서	내용	시간
	0. 전강 연결	5분
	1. 오프닝 도발 — 번역 실험	10분
	2. 개념 설명 (FAC 트라이앵글, 모호함 출처)	20분
	3. 고정 케이스 실습 — 워크시트 A-2	30분
	4. 현업 케이스 슬롯 — 워크시트 B-2	20분
	5. 디브리프	10분
	6. 과제 카드 #2 배포	5분
	합계	100분

2.2. 강의 운영 주의사항

- ★ '번역 실험'에서 다른 답들이 나왔을 때, '틀린 것이 있나요?'라고 묻고 잠시 기다리세요. 침묵이 학습 순간입니다.
- ★ Attributes 칸이 비어있는 것이 자연스럽습니다. 캐내야만 나온다는 것을 강조하세요.
- ★ Constraints 칸이 가장 어려울 것입니다. '말하지 않은 전제를 찾는 것'임을 안내하세요.
- ★ '문서가 아니라 문서화가 전부다' — 화이트보드 작업 자체가 팀 정렬 과정임을 짚어주세요.

2.3. 강의 진행 상세

2.4. 0. 전강 연결 (5분)

칠판이나 슬라이드에 1강 과제 카드의 핵심을 꺼냅니다.

```
지난 강에서:
"나는 지금 문제를 이해한 건가, 패턴을 인식한 건가?"

1강 과제에서 가져온 것 수거:
"이번 주에 받은 요구사항, Problem Space 질문이
 몇 개 나왔나요?" → 2-3명 짧게 공유

이번 강에서:
질문을 만들 수 있다는 것은 좋습니다.
그런데 어떤 질문이 효과적인가?
모호함이 어디 있는지 알아야 좋은 질문이 나옵니다.
```

2.5. 1. 오프닝 도발 (10분)

"번역 실험"

슬라이드나 칠판에 문장 하나를 띄웁니다.

> "사용자가 더 쉽게 로그인할 수 있게 해주세요."

수강생들에게 묻습니다.

> "이 문장이 의미하는 것을 구체적으로 적어주세요.
>
> 이걸 개발 태스크로 만든다면 어떤 태스크가 생기나요?
>
> 1분, 혼자 작업해주세요."

1분 후, 3-4명에게 읽어달라고 합니다.

전형적으로 나오는 것들:

```
A: "소셜 로그인(Google, Kakao) 추가"
B: "로그인 폼 UI 개선, 입력 필드 줄이기"
C: "비밀번호 찾기 프로세스 간소화"
D: "자동 로그인 / Remember me 기능"
E: "로그인 에러 메시지 개선"
```

이것들을 칠판에 나란히 적고 말합니다.

> "같은 문장을 읽었는데, 다섯 가지 다른 태스크가 나왔습니다.
>
> 이 중에 틀린 것이 있나요?"

잠시 침묵.

> "없습니다. 전부 옳을 수 있고, 전부 틀릴 수도 있습니다.
>
> 왜냐하면, 원래 문장이 이 중 어떤 것을 의미하는지
>
> 아직 아무도 모르기 때문입니다.
>
> 이것이 모호함입니다."

2.6. 2. 개념 설명 (20분)

2.6.1. 개념 1 — 모호함의 세 가지 출처 (8분)

Weinberg는 요구사항의 모호함이 세 가지 다른 곳에서 온다고 설명합니다.

① 언어적 모호함 (Linguistic Ambiguity) 단어 자체가 측정 기준을 품고 있지 않은 경우.

```
"빠른 응답"          → 얼마나 빠른? 200ms? 2초?
"사용하기 편리한"    → 누구에게? 어떤 상황에서?
"안전한"             → 어떤 위협으로부터? 어떤 수준으로?
"더 좋게"            → 무엇 기준으로?
```

이런 단어들은 말하는 사람 머릿속에 기준이 있습니다. 하지만 그 기준이 전달되지 않았습니다.

② 해석자 간 모호함 (Interpreter Ambiguity) 같은 단어인데 사람마다 다른 것을 상상하는 경우.

```
"알림 강화"
  BizDev가 상상한 것:    고객사 팀장 PC에 팝업
  백엔드 개발자:          이메일 / 슬랙 webhook
  프론트엔드 개발자:      인앱 뱃지, 빨간 점
  사용자(팀원):           카톡 오면 안 되나?
```

같은 단어로 대화하고 있지만, 전혀 다른 것을 이야기하고 있습니다. 이 상태로 개발이 시작되면 어떤 일이 생기는지는 다들 경험이 있을 겁니다.

③ 생략된 맥락의 모호함 (Missing Context Ambiguity) 말하지 않은 전제가 있는 경우.

```
"지금처럼 동작해야 해요"      → 지금이 어떻게 동작하는지 아는가?
"당연히 그건 안 되겠죠"       → 무엇이 당연한가?
"기존 방식을 유지하면서"      → 기존 방식이 무엇인가?
"빠르게 만들어주세요"         → 얼마나 빠르게? 무엇을 포기하고?
```

이런 전제들은 말하는 사람에게는 너무 당연해서 말하지 않습니다. 하지만 듣는 사람에게는 알 수가 없습니다.

2.6.2. 개념 2 — Functions / Attributes / Constraints 트라이앵글 (12분)

Weinberg는 모든 요구사항이 실제로는 세 가지 다른 범주로 이루어져 있다고 봅니다.

```
Attributes ──────── Constraints
(어떻게 해야 하는가)  (건드릴 수 없는 것)
```

범주	정의	전형적 표현	잘 빠지는 함정
Functions	시스템이 수행하는 동작	"결제한다", "알림을 보낸다", "배정한다"	여기만 받아 적고 끝낸다
Attributes	그 동작의 품질 기준	"200ms 이내", "99.9% 가용성", "모바일에서도"	나중에 재작업의 원인이 된다
Constraints	해결책이 지켜야 할 조건	"PCI-DSS 준수", "예산 2천만원", "기존 DB 변경 불가"	설계 다 하고 나서야 발견한다

왜 이 구분이 중요한가?

Function만 들은 경우: "알림을 보낸다" → 어떤 방법으로든 보내면 된다고 생각 → 이메일로 구현 → 나중에 "실시간이어야 한다"는 Attribute가 나옴 → 재작업

Attribute를 먼저 들은 경우: "실시간으로, 모든 플랫폼에, 1초 이내에" → Attribute가 Architecture를 결정한다 → 처음부터 WebSocket 고려

Constraint를 나중에 발견한 경우: 설계를 마쳤는데 → "아 참, 고객사 네트워크가 WebSocket을 막고 있어요" → 전면 재설계

결론: Attribute와 Constraint는 캐내지 않으면 나오지 않습니다. BizDev도, 사용자도, 대부분 Function 언어로 이야기합니다. 엔지니어가 질문으로 꺼내야 합니다.

"문서가 아니라 문서화가 전부다" (Weinberg)

Weinberg의 가장 중요한 통찰 중 하나를 짧게 전달합니다.

> "The document is nothing; the documenting is everything."

요구사항 문서를 만드는 목적은 _문서를 저장하는 것_이 아닙니다. 문서를 만드는 _과정에서 팀이 같은 멘탈 모델을 갖게 되는 것_이 목적입니다.

FAC 트라이앵글을 채우는 행위 자체가 팀의 이해를 정렬시킵니다. 결과물보다 대화가 중요합니다.

2.7. 3. 고정 케이스 실습 (30분)

2.7.1. 소그룹 공유 가이드

소그룹 공유 (8분)

3-4인 그룹으로 PART 2의 결과를 비교합니다.

퍼실리테이터 가이드:

```
비교 포인트
──────────

"같은 칸에 다른 것을 적은 경우가 있는가?"
  → 그 차이 자체가 해석자 간 모호함의 실례입니다.

"누군가 Attribute를 Function으로 적은 경우가 있는가?"
  → 왜 그렇게 분류했는지 이야기해보게 합니다.

"Constraints 칸이 가장 비어있는 경향이 있는가?"
  → 대부분 그렇습니다. Constraint는 가장 캐내기 어렵습니다.
    "왜 이건 생각이 안 났을까?"를 메타적으로 물어보세요.
```

전체 공유 (7분)

각 그룹에서 "가장 위험한 공백"으로 고른 것을 칠판에 모읍니다. 서로 다른 것을 골랐다면 — 왜 다른지가 토론의 핵심입니다.

2.8. 4. 현업 케이스 슬롯 (20분)

2.8.1. 디브리프 가이드

2.9. 5. 디브리프 (10분)

퍼실리테이터 가이드 — 예상 저항과 대응

```
"Attribute와 Function 경계가 헷갈려요"
──────────────────────────────────────

정상입니다. 경계는 실제로 맥락에 따라 달라집니다.
"실시간"은 Attribute일 수도, 핵심 Function일 수도 있습니다.
중요한 건 분류 자체가 아니라, 그 항목에 대해 팀이
같은 것을 이해하고 있는가입니다.

"Constraint 칸이 잘 안 채워져요"
────────────────────────────────

Constraint는 가장 캐내기 어렵습니다.
당사자가 "당연한 것"으로 여겨서 말하지 않기 때문입니다.
다음 강(3강)에서 이것을 끌어내는 질문 기법을 다룹니다.

"FAC를 다 채우고 나면 개발을 시작해도 되나요?"
──────────────────────────────────────────────

"?"가 전부 "확인됨"이 되면 시작해도 됩니다.
하지만 어떤 "?"는 물어봐도 답이 안 나올 수 있습니다.
그건 5강(모호함을 안고 가기)에서 다룹니다.
지금은 일단 "무엇을 모르는지"를 아는 것이 목표입니다.
```

```
"이걸 매번 다 해야 하나요? 시간이 너무 걸릴 것 같아요"
_____________________________________________________

처음엔 시간이 걸립니다. 하지만 목표는 이 도구를
대화 안에서 자연스럽게 쓰는 것입니다.
'FAC 테이블을 채우는 행위'보다 '이 세 가지를
머릿속에서 자동으로 체크하는 습관'이 최종 목표입니다.
```

Chapter 3. 질문하는 자가 설계한다 — Context-Free Questions

3.1. 시간 배분

순서	내용	시간
	0. 전강 연결	5분
	1. 오프닝 도발 — 두 종류의 질문	10분
	2. 개념 설명 (Context-Free Q 3카테고리, Black Box)	20분
	3. 고정 케이스 실습 — 인터뷰 롤플레이 + 워크시트 A-3	30분
	4. 현업 케이스 슬롯 — 워크시트 B-3	20분
	5. 디브리프	10분
	6. 과제 카드 #3 배포	5분
	합계	100분

3.2. 강의 운영 주의사항

- ★ 인터뷰 롤플레이 전에 민준 역할 카드를 해당 수강생에게만 보여주세요. 비대칭 정보가 핵심입니다.
- ★ Meta-Question을 쓴 팀에서 '야간 조' 같은 예상 밖 정보가 나오면 전체 공유로 꺼내세요. 이 경험이 Meta-Question의 효과를 증명합니다.
- ★ '어색하다'는 저항이 나오면: '자연스러운 대화 안에서 이 정보를 얻는 것이 목표'임을 환기시키세요.
- ★ Black Box의 예외 케이스를 많이 찾은 팀이 있다면 어떤 질문을 했는지 역추적하게 하세요.

3.3. 강의 진행 상세

3.4. 0. 전강 연결 (5분)

```
지난 강에서:
요구사항은 Functions / Attributes / Constraints로
이루어져 있다.
그리고 "?" 칸이 "확인됨"보다 많은 것이 정상이다.

2강 과제에서 가져온 것 수거:
"Attributes나 Constraints 칸을 채우려 할 때
 어떤 질문이 막혔나요?" → 2-3명 짧게 공유

이번 강에서:
그 빈 칸을 채우는 질문을 어떻게 만드는가.
좋은 질문과 나쁜 질문이 있는가?
```

```
있다면 무엇이 다른가?
```

3.5. 1. 오프닝 도발 (10분)

"두 종류의 질문 실험"

수강생에게 시나리오를 줍니다.

> 여러분은 방금 BizDev에게 이 말을 들었습니다.

> "대시보드를 좀 더 직관적으로 만들어주세요."

>

> 이제 BizDev에게 질문 하나를 할 수 있습니다.

> 어떤 질문을 하겠습니까? 적어주세요. (1분)

1분 후, 칠판에 나온 질문들을 두 그룹으로 분류합니다. 수강생에게 미리 알리지 않고, 분류하면서 패턴을 찾게 합니다.

전형적으로 나오는 질문들:

```
그룹 A (좁히는 질문)
──────────────────────────────────────────

"어떤 메뉴를 먼저 보이게 할까요?"
"차트 색상을 바꿀까요?"
"로딩 속도를 개선할까요?"
"레이아웃을 어떻게 바꾸면 좋을까요?"

그룹 B (여는 질문)
──────────────────────────────────────────

"지금 대시보드에서 가장 불편한 점이 무엇인가요?"
"대시보드를 주로 어떤 상황에서 사용하시나요?"
"직관적이 된다면 어떤 모습이어야 할까요?"
"지금 대시보드로 해결 못 하고 있는 것이 있나요?"
```

분류 후 말합니다.

> "그룹 A는 이미 해결책을 가정하고 있습니다.

> '차트 색상을 바꿀까요?'는 문제가 차트 색상이라고 전제합니다.

> 그룹 B는 아직 문제 공간에 있습니다.

>

> Weinberg는 그룹 B 같은 질문을 특별히 이름 붙였습니다.

> Context-Free Questions — 맥락(해결책)을 전제하지 않는 질문."

3.6. 2. 개념 설명 (20분)

3.6.1. 개념 1 — Context-Free Questions (12분)

Weinberg가 _Exploring Requirements_에서 제시한 개념입니다. Context-Free Questions는 어떤 도메인, 어떤 제품에도 적용 가능한 _범용 질문_입니다. 특정 해결책을 전제하지 않기 때문에, 문제 공간을 좁히지 않고 탐색할 수 있습니다.

세 카테고리로 나뉩니다.

① Process Questions — 어떻게 일어나고 있는가

현재 상황과 흐름을 이해하는 질문입니다.

```
"지금 이 문제를 어떻게 해결하고 있나요?"
"이 작업을 하는 사람이 누구인가요? 몇 명인가요?"
"이 문제가 가장 심각하게 느껴지는 순간은 언제인가요?"
"이게 해결되지 않으면 어떤 일이 생기나요?"
"지금 임시방편으로 쓰고 있는 방법이 있나요?"
```

특히 마지막 질문 — "지금 임시방편으로 뭘 하고 있나요?" — 은 Weinberg가 가장 강력하다고 강조하는 질문입니다. Workaround는 진짜 니즈의 단서입니다. 사람들은 말로는 표현 못 하는 것을 행동으로 이미 하고 있습니다.

② Product Questions — 무엇이어야 하는가

시스템에 대한 기대와 경계를 이해하는 질문입니다.

```
"이 시스템이 반드시 해야 하는 것은 무엇인가요?"
"절대 해서는 안 되는 것은 무엇인가요?"
"어떤 상태가 되면 '잘 됐다'고 할 수 있나요?"
"지금 없는 것 중, 있으면 가장 좋겠다는 것은?"
"누가 이것의 성공 여부를 판단하나요?"
```

"절대 해서는 안 되는 것"은 특히 중요합니다. 요구사항 대화에서 거의 나오지 않지만, 이것이 나중에 Constraint로 발목을 잡는 경우가 많습니다.

③ Meta-Questions — 질문에 대한 질문

Weinberg가 종종 가장 강력하다고 말하는 카테고리입니다.

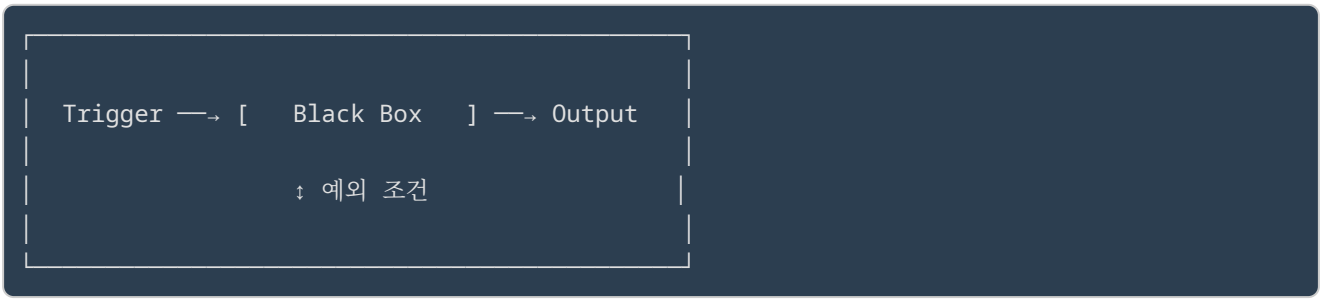
```
"제가 이런 질문을 해도 괜찮겠습니까?"
"제가 놓친 것이 있을까요?"
"제가 이해한 것이 맞는지 확인해도 될까요?"
"이 주제에 대해 더 잘 아는 사람을 연결해주실 수 있나요?"
"제 질문 중 잘못된 방향으로 가고 있는 것이 있나요?"
```

Meta-Question이 강력한 이유: 상대방에게 _대화의 방향을 교정할 권한_을 줍니다. 엔지니어가 잘못된 가정을 가지고 있을 때, 상대방이 이것을 자연스럽게 수정할 수 있게 됩니다.

3.6.2. 개념 2 — Black Box Test (8분)

Context-Free Questions로 충분히 탐색했다면, 다음 단계는 시스템의 _경계_를 합의하는 것입니다.

Weinberg는 이것을 _Black Box Test_로 설명합니다. 시스템 내부를 어떻게 구현할지 논의하기 전에, 외부에서 보이는 것만으로 먼저 합의하는 방법입니다.



Black Box Template:

항목	질문	설명
이름	이 기능/시스템을 한 마디로 부른다면?	이름을 합의하는 것 자체가 첫 번째 정렬
Trigger	언제 이 기능이 시작되는가?	어떤 사건이 이 기능을 발동시키는가
Input	무엇이 들어오는가? 누가 넣는가?	사람, 데이터, 시스템 이벤트
Output	무엇이 나와야 하는가? 누구에게?	사용자 화면, 데이터, 외부 시스템
예외	이 기능이 실패하거나 동작 안 하는 조건	가장 자주 빠뜨리는 부분

Black Box Test가 유용한 이유: * 내부 구현 논쟁 없이 경계를 먼저 합의할 수 있다 * "이 경우는 어떻게 됩니까?"라는 예외 케이스 질문이 자연스럽게 나온다 * BizDev도, 사용자도 이해할 수 있는 언어로 요구사항을 확인할 수 있다

3.6.3. 짧은 연결 — Jobs-to-be-Done (보충 개념, 3분)

Christensen의 관점을 한 줄로 전달합니다.

> "사람들은 제품을 사지 않는다. 어떤 Job을 완수하기 위해 제품을 '고용'한다."

"알림 기능을 만들어달라"는 것은 _기능 요청_입니다. "팀장이 배정했는데 팀원이 모르는 상황을 없애달라"는 것은 _Job_입니다.

같은 Job을 완수하는 방법은 여러 가지일 수 있습니다. 알림이 아닐 수도 있습니다. Context-Free Questions는 기능 요청 뒤에 있는 Job을 찾는 도구입니다.

3.7. 3. 고정 케이스 실습 (30분)

3.7.1. 소그룹 공유 가이드

소그룹 공유 (5분)

같은 역할 카드를 쓴 팀들이 모여 Black Box를 비교합니다.

퍼실리테이터 가이드:

```
비교 포인트
──────────

"같은 인터뷰 대상인데 Black Box가 얼마나 다른가?"
  → 질문의 방향이 달랐기 때문입니다.
    어떤 질문이 어떤 발견을 만들었는지 역추적해보세요.

"예외 케이스를 가장 많이 찾은 팀은 어떤 질문을 했는가?"
  → "절대 해서는 안 되는 것"과 "이 기능이 실패하는 조건"을
    물어본 팀이 예외를 더 많이 발견합니다.

"Meta-Question을 쓴 팀에서 어떤 일이 생겼는가?"
  → '제가 놓친 게 있나요?'가 가장 새로운 정보를 끌어냅니다.
    이 경험을 전체 공유에서 꺼내세요.
```

3.8. 4. 현업 케이스 슬롯 (20분)

3.8.1. 워크시트 B-3 | 내 현업 케이스: Context-Free Questions 적용

```
┌──────────────────────────────────────────────────────────────┐


==== 디브리프 가이드
```

```
=== 5. 디브리프 (10분)

_퍼실리테이터 가이드 — 예상 저항과 대응_
```

"Context-Free Questions가 너무 원론적인 것 같아요. 현업에서 이렇게 물어보면 이상하지 않나요?"

——— 맞습니다. 이 질문들을 그대로 읽는 것이 목표가 아닙니다. 이 카테고리를 머릿속에 가지고, 자연스러운 대화 안에서 이 정보를 얻는 것이 목표입니다. 승현이 민준에게 "Process Questions를 드리겠습니다"라고 하지 않았던 것처럼.

"BizDev이 모른다고 하면 어떻게 해요?"

——— '모른다'는 답도 정보입니다. '누가 알 것 같으세요?'가 다음 질문입니다. 그리고 아무도 모른다면 — 그것 자체가 5강의 주제입니다. 본질적 모호함일 수 있습니다.

"Black Box의 예외 케이스가 너무 많아서 멈춰요."

———— 좋은 신호입니다. 예외를 많이 발견할수록 나중에 놀라는 일이 줄어듭니다. 다 해결하려고 하지 않아도 됩니다. '지금 알고 있다'는 것이 중요합니다.

"인터뷰를 잘 못 하겠어요. 너무 어색해요."

——— Meta-Question 하나만 기억해도 됩니다. '제가 놓친 것이 있을까요?' 이 한 질문이 가장 많은 것을 끌어냅니다.

```
'''



'''


== 지도를 그리기 전에 — 문제의 구조와 Domain Model

=== 시간 배분

[cols="1,1,1", options="header"]
|===
| 순서 | 내용 | 시간
|  | 0. 전강 연결 | 5분
|  | 1. 오프닝 도발 — 두통약 비유 | 10분
|  | 2. 개념 설명 (레이어 3개, Problem Framing, Domain Model) | 20분
|  | 3. 고정 케이스 실습 — 워크시트 A-4 | 30분
|  | 4. 현업 케이스 슬롯 — 워크시트 B-4 | 20분
|  | 5. 디브리프 | 10분
|  | 6. 과제 카드 #4 배포 | 5분
|  | *합계* | *100분*
|===
```

```
=== 강의 운영 주의사항

- ★ 레이어 분류에서 정답을 주지 마세요. '우리가 어느 레이어를 고치려는가'의 합의가 목표입니다.
- ★ Domain Model 스케치 시 State(상태) 설계가 가장 많이 갈립니다. '배정됨'과 '확인됨'의 차이를
팀마다 다르게 그렸다면 전체 비교로 꺼내세요.
- ★ '근본 원인까지 항상 해결해야 하나?'는 중요한 질문입니다. '의식적 선택'임을 강조하세요. 5강 Spike와
7강 Risk의 복선입니다.
- ★ Domain Model이 '문서'가 아니라 '팀 어휘 정렬 과정'임을 반복해서 짚어주세요.

=== 강의 진행 상세

=== 0. 전강 연결 (5분)
```

지난 강에서: Context-Free Questions로 FAC의 빈 칸을 채웠다. Black Box로 시스템의 경계를 합의했다.

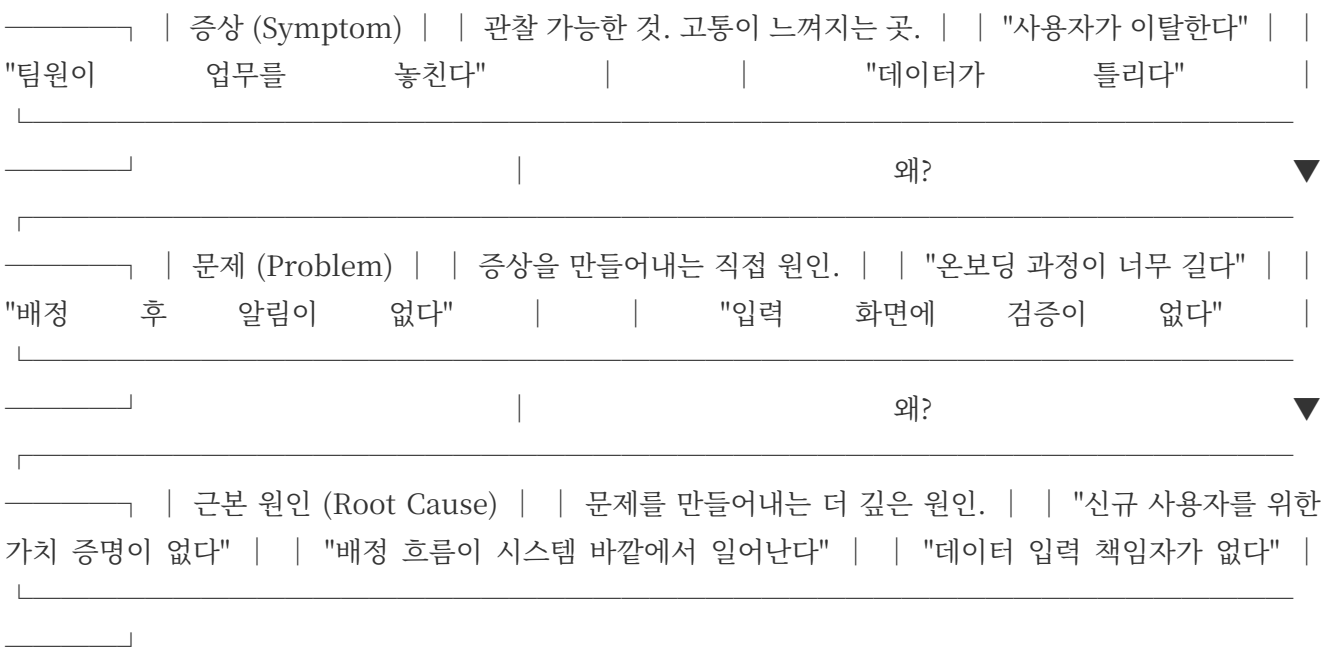
3강 과제에서 가져온 것 수거: "실제로 질문을 써봤을 때 어떤 일이 있었나요?" → 2-3명 짧게 공유 → 특히 "예상 밖의 발견"이 있었다면 꺼내기

이번 강에서: 충분히 물었다. 충분히 들었다. 그런데 — 우리는 올바른 문제를 이해하고 있는가? 물어봤다고 문제를 이해한 것은 아닐 수 있다.

```
'''

=== 1. 오프닝 도발 (10분)

_"두통약 비유"_

슬라이드나 말로 짧게 씁니다.

[quote]
____
_어떤 사람이 매일 아침 두통을 호소합니다._
____
[quote]
____
_병원에 갔더니 의사가 두통약을 처방해줬습니다._
____
[quote]
____
_약을 먹으면 두통이 가라앉습니다._
____
[quote]
____
_그런데 다음 날 아침, 두통이 또 옵니다._
____

[quote]
____
```

```
_두통은 증상입니다._

____
[quote]

____
_알고 보니 원인은 – 매일 아침 수면 부족과 카페인 과다였습니다._

____

[quote]

____
_두통약은 증상을 해결했지만, 문제를 해결하지 않았습니다._

____

잠시 후 묻습니다.

[quote]

____
"소프트웨어 개발에서 이런 일을 해본 적 있나요?

____
[quote]

____
요청받은 것을 만들었는데, 얼마 후 비슷한 요청이 또 왔던 경험."

____

손을 들게 합니다. 대부분 들 것입니다.

[quote]

____
"그것이 증상을 고쳤기 때문입니다.

____
[quote]

____
이번 강은 – 증상과 문제와 근본원인을 구분하는 방법,

____
[quote]

____
그리고 문제를 충분히 이해했다는 것을 어떻게 아는가입니다."

____

'''

=== 2. 개념 설명 (20분)

==== 개념 1 – 문제의 세 레이어 (10분)

모든 문제 상황에는 세 개의 레이어가 있습니다.
```

표면 | ▼




```
왜 구분이 중요한가?
```

증상 레이어에서 해결하면: 두통약 → 증상은 잠시 없어짐 → 문제가 남아있어 증상 재발

문제 레이어에서 해결하면: 알림 추가 → 이 문제는 해결됨 → 근본 원인이 남으면 다른 문제 발생

근본 원인 레이어에서 해결하면: 배정 흐름 전체를 시스템으로 끌어들임 → 여러 증상이 동시에 사라짐

```
하지만 항상 근본 원인까지 파야 한다는 뜻은 아닙니다. _어느 레이어에서 해결할 것인지는 의식적인 선택_이어야 합니다. 지금 당장의 데모를 위해 증상 레이어에서 해결하기로 결정하는 것은 괜찮습니다 — 그것이 결정인 줄 알고 있다면.

'''

_Problem Framing — 어떻게 보느냐에 따라 해결책이 달라진다_

같은 현상을 어떻게 프레임하느냐에 따라 Problem Space의 모양이 달라집니다.
```

현상: "팀원들이 업무를 놓친다"

프레임 A (알림 문제로 보면): → 알림 시스템 강화 → 기술적 해결책이 나온다

프레임 B (커뮤니케이션 문제로 보면): → 배정 프로세스 재설계 → 조직/프로세스 해결책이 나온다

프레임 C (가시성 문제로 보면): → 팀장이 팀원의 현재 업무를 볼 수 있게 → 대시보드 해결책이 나온다

프레임 D (책임 명확성 문제로 보면): → 배정 확인(acknowledgement) 기능 → 다른 종류의 기술 해결책이 나온다

```
이 중 어떤 프레임이 "맞는가"는 지금은 모릅니다. 하지만 _한 가지 프레임만 있다는 것이 가장 위험합니다._ 여러 프레임이 가능하다는 것을 인식하는 것이 첫 번째 훈련입니다.

'''
```

```
==== 개념 2 — Domain Model vs Design Model (Fairbanks) (10분)

Fairbanks는 소프트웨어를 둘러싼 모델을 세 가지로 구분합니다.
```

Domain Model ←—— Problem Space에 속한다 | | (이것이 Bridge) ▼ Design Model ←—— Solution Space의 시작점 | ▼ Code Model ←—— 실제 구현

```
_Domain Model — 현실 세계의 개념들_

이 시스템이 다루는 _실세계의 사람, 행위, 규칙, 상태_입니다.

팀오더 케이스에서 Domain Model의 요소들:
```

사람(Actor): 팀장 / 팀원 / 고객사 관리자 행위(Action): 업무 배정 / 업무 확인 / 업무 완료 / 긴급 배정 상태(State): 미배정 / 배정됨 / 확인됨 / 진행중 / 완료 / 지연 규칙(Rule): 야간 긴급은 즉시 전달 / 미확인 시 재알림 / 마감일 리마인더 물건(Object): 업무 / 팀 / 라인 / 마감일

```
_Design Model — 만들 소프트웨어의 구조_

ERD, API 설계, 시스템 다이어그램. 여기서부터 Solution Space입니다.

대부분의 엔지니어가 하는 실수:
```

요구사항 수신 | | ← 여기를 건너뜀 ▼ Domain Model (현실 이해) | | ← 여기도 건너뜀 ▼ Design Model (ERD, API) ← 여기서 시작함

```
Domain Model을 건너뛰면 무슨 일이 생기는가?

설계한 것이 현실의 개념과 어긋납니다. 개발 중에 "이 케이스는 어떻게 처리하죠?"라는 질문이 반복됩니다. 그 질문들은 대부분 Domain Model 레이어에 있는 것들입니다.

_Domain Model이 충분히 안정됐을 때가 Design Model로 넘어갈 시점입니다._

어떻게 아는가? 팀이 같은 도메인 어휘를 쓰고 있을 때. "업무"와 "태스크"를 혼용하지 않을 때. "배정됨"과 "확인됨"이 구분될 때.

'''

=== 3. 고정 케이스 실습 (30분)


==== 소그룹 공유 가이드

_소그룹 공유 (7분)_

각 그룹의 Domain Model을 칠판에 나란히 붙이거나 그립니다.
```

```
퍼실리테이터 가이드:
```

비교 포인트 ———————————— "상태(State)를 얼마나 다르게 그렸는가?" → State 설계가 가장 많이 갈립니다. '배정됨'과 '확인됨'을 같은 것으로 본 팀 vs 다른 것으로 본 팀. 이 차이가 나중에 DB 설계, API 설계를 완전히 바꿉니다.

"Rule을 얼마나 찾았는가?" → 야간 긴급 규칙을 넣은 팀이 있는가? 없는 팀은 — 왜 빠졌는지 돌아보게 합니다.

"지금 그린 Domain Model이 현재 설계(있다면)와 얼마나 다른가?" → 괴리가 큰 것이 나중에 재작업 포인트입니다.

```
'''

=== 4. 현업 케이스 슬롯 (20분)

==== 워크시트 B-4 | 내 현업 케이스: 레이어 분석 + Domain Model
```

3.8.2. 디브리프 가이드

3.9. 5. 디브리프 (10분)

퍼실리테이터 가이드 — 예상 저항과 대응

```
"증상/문제/근본원인 경계가 애매해요."
————————————————————————————————————

맞습니다. 이 경계는 관점에 따라 달라집니다.
'배정 후 연락이 없다'는 사람에 따라
증상이 될 수도, 문제가 될 수도 있습니다.
중요한 것은 레이어의 '정확한 분류'가 아니라
"우리가 어느 레이어를 고치려고 하는가"를
팀이 같이 인식하는 것입니다.

"Domain Model을 항상 그려야 하나요?
 작은 기능에도 해야 하나요?"
————————————————————————————————————

도구를 항상 꺼낼 필요는 없습니다.
목표는 'Domain Model 문서'가 아니라
'도메인 어휘가 팀 안에서 정렬되는 것'입니다.
슬랙 대화에서 "배정됨이랑 확인됨이 다른 거죠?"라고
자연스럽게 물어보는 것도 Domain Model 작업입니다.

"근본원인까지 해결하는 게 항상 좋은 건 아닌가요?"
————————————————————————————————————————

맞습니다. 이것은 5강과 7강의 주제이기도 합니다.
데모까지 1개월이면 증상 레이어 해결이 현실적일 수 있습니다.
중요한 것은 — 그것이 선택인 줄 알고 있는 것.
"우리는 지금 증상을 고치고 있다.
 근본 원인은 다음 분기에 다룬다."라는 결정.
```

Chapter 4. 안개 속에서 걷는 법 — 모호함을 안고 가기

4.1. 시간 배분

순서	내용	시간
	0. 전강 연결	5분
	1. 오프닝 도발 — 두 가지 모르는 것	10분
	2. 개념 설명 (R/E 구분, Cunningham, Spike, Schön) ★ 이 강은 25분	25분
	3. 고정 케이스 실습 — 워크시트 A-5	30분
	4. 현업 케이스 슬롯 — 워크시트 B-5	15분
	5. 디브리프	10분
	6. 과제 카드 #5 배포	5분
	합계	100분

4.2. 강의 운영 주의사항

- ★ 이 강은 1-4강의 전제를 뒤집습니다. 수강생이 불편해할 수 있습니다. 그 불편함이 정상임을 인정하세요.
- ★ '그냥 만들자'와 Spike의 차이를 명확히 하세요. Spike는 목적(학습)과 결과(버릴 것)를 알고 시작합니다.
- ★ R/E 분류가 틀려도 괜찮음을 강조하세요. Spike를 하다 보면 명확해집니다.
- ★ Technical Debt의 원래 의미(Cunningham)를 잘못 이해하는 수강생이 많습니다. '나쁜 코드'가 아니라 '의도적 인식론적 선택'임을 충분히 설명하세요.
- ★ '지금 결정이 꼭 필요한가?' 체크 — 이 판단 자체가 시니어 역량임을 짚어주세요.

4.3. 강의 진행 상세

강의 성격	**이 강은 1-4강의 전제를 뒤집는 전환점. 개념 설명에 충분한 시간을 줄 것.**

4.4. 0. 전강 연결 (5분)

```
지금까지의 여정:
  1강: 나는 Solution Space로 너무 빨리 넘어가고 있었다
  2강: 요구사항은 Functions / Attributes / Constraints다
  3강: Context-Free Questions로 빈 칸을 채운다
  4강: 증상/문제/근본원인을 구분하고, Domain Model을 그린다
```

```
공통 전제:
  "모호함은 → 질문으로 → 해소한다"

이번 강은 이 전제에 균열을 냅니다.

4강 과제에서 가져온 것 수거:
"팀원에게 Domain Model을 보여줬을 때
 다르게 이해한 부분이 있었나요?" → 2-3명 짧게 공유
```

4.5. 1. 오프닝 도발 (10분)

"두 가지 모르는 것 실험"

수강생에게 두 가지 상황을 줍니다.

> 상황 A

> 고객사 BizDev이 "알림이 얼마나 빨리 가야 하는지"를 모른다고 합니다.

> 고객사 실제 사용자(팀장)에게 물어보면 알 수 있을 것 같습니다.

>

> 상황 B

> 팀이 "이 알림 기능을 사용자들이 실제로 쓸 것인지"를 모릅니다.

> 아무도 아직 이런 기능을 써본 적이 없고, 고객사도 확신이 없습니다.

두 상황에 대해 묻습니다.

> "두 상황 모두 '모른다'입니다.

> 같은 방법으로 해결할 수 있을까요?

> 아니면 다른 방법이 필요할까요?

> 30초 안에 판단해보세요."

30초 후 손을 들게 합니다. "같은 방법"과 "다른 방법"으로 나뉠 것입니다.

> "왜 다르다고 생각했나요? 왜 같다고 생각했나요?"

이 토론이 오늘 강의의 출발점입니다.

> "상황 A는 물어보면 알 수 있는 모호함입니다.

> 상황 B는 아무도 모르는 모호함입니다.

> 물어봐도 답이 없습니다.

>

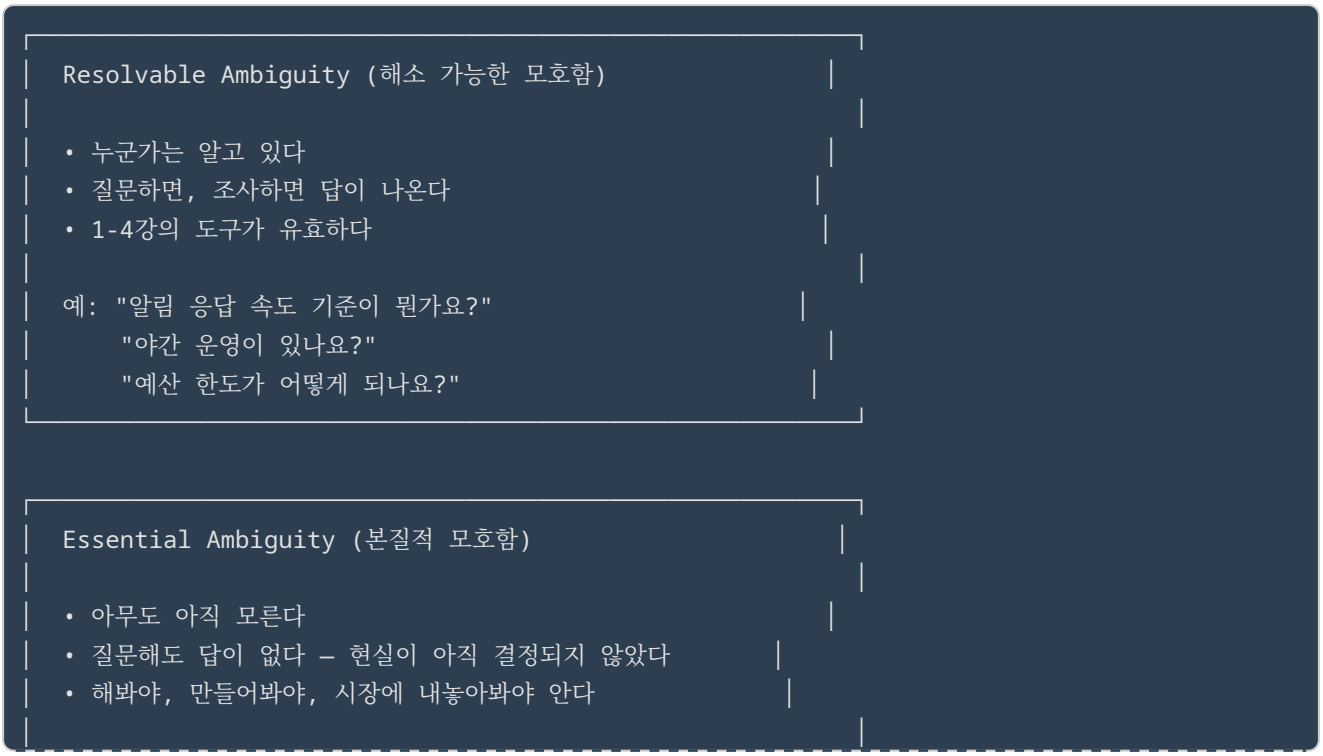
> 1-4강은 상황 A를 위한 도구였습니다.

> 오늘은 상황 B를 다룹니다."

4.6. 2. 개념 설명 (25분)

4.6.1. 개념 1 — 모호함의 두 종류 (10분)

모호함에는 근본적으로 다른 두 가지 종류가 있습니다.



```
│  예: "사용자가 이 기능을 실제로 쓸까요?"                    │
│      "이 방식이 우리 팀의 워크플로우에 맞을까요?"           │
│      "어떤 알림 형식이 이 환경에서 효과적일까요?"           │
└──────────────────────────────────────────────────────────┘
```

이 두 가지를 구분하지 못하면 두 가지 실수가 생깁니다.

실수 1 — Essential Ambiguity를 Resolvable처럼 다룬다: 답이 없는 질문을 계속 던집니다. 결정을 미룹니다. 완벽한 정보를 기다리다가 시간을 씁니다. "더 조사하면 알 수 있을 것 같아서요."

실수 2 — Resolvable Ambiguity를 Essential처럼 다룬다: 물어보면 알 수 있는 것을 그냥 가정합니다. "이 정도면 대충 맞겠지." 나중에 재작업합니다.

진단 질문 하나:

> "이것을 알기 위해 내가 할 수 있는 행동이 있는가?"

>

> 있다면 → Resolvable. 그 행동을 하면 됩니다.

> 없다면 (또는 행동해도 불확실하다면) → Essential.

> Cunningham 전략이 필요합니다.

4.6.2. 개념 2 — Cunningham의 접근: 가장 단순한 것 (10분)

Ward Cunningham은 소프트웨어 개발의 여러 핵심 아이디어를 만든 사람입니다. Wiki를 발명했고, XP(Extreme Programming)의 핵심 실천들을 설계했습니다.

Cunningham이 던진 질문 하나:

> "What is the simplest thing that could possibly work?"

이 질문은 자주 오해됩니다. "최소 기능을 만들어라"가 아닙니다.

정확한 의미:

> "이 모호함을 드러내줄 가장 작은 행동은 무엇인가?"

Essential Ambiguity 앞에서 Cunningham의 전략은 — _완성하려는 것이 아니라 배우려는 것_입니다. 가장 작고 빠른 행동을 통해, 현실로부터 피드백을 받습니다. 그 피드백이 모호함을 줄여줍니다.

Spike Solution (XP):

Spike는 생산 코드를 쓰기 전에, _특정 모호함을 탐색하기 위한 버려지는 코드_입니다.

```
목적:  이 기술이 이 요구사항을 충족하는가?
       이 방식이 이 환경에서 동작하는가?

방법:  최대한 단순하게, 빠르게, 버릴 것을 알고 만든다
       프로덕션 품질 불필요

결과:  Yes/No 또는 새로운 질문
       코드가 아니라 학습이 산출물
```

팀오더 케이스에서의 예:

> "사용자가 백그라운드 알림을 실제로 받는가?"

> → 문서를 읽어서 확인하는 것 vs

> `테스트 앱으로 직접 디바이스에 보내보는 것`

이 질문은 문서 조사로도 어느 정도 답이 나옵니다. 하지만 "야간에 공장 현장에서 태블릿 화면이 꺼진 상태에서 받히는가?"는 실제로 해봐야 압니다. 이것이 Spike입니다.

Technical Debt — 원래 의미 (Cunningham)

Cunningham이 Technical Debt를 처음 이야기했을 때의 맥락을 많은 사람들이 모릅니다.

그는 "나쁜 코드"에 대한 비유로 이 말을 한 것이 아니었습니다.

Cunningham의 원래 의미:

> "지금 우리가 이해한 것보다 더 정교한 설계가 필요하지만,

> `지금 당장 그것을 할 수 없다.`

> `의도적으로 단순하게 만들고,`

> `학습이 쌓이면 리팩토링한다.`

> `이 차이가 '부채'다."`

핵심: _의도적인 인식론적 선택_입니다.

"지금 모르는 채로 만들고, 알게 되면 고친다" — 이것이 부채를 지는 것입니다. 이것은 게으름이 아닙니다.

Essential Ambiguity 앞에서의 성숙한 전략입니다.

반면, 몰라서 아무렇게나 만드는 것은 부채가 아닙니다. 그건 그냥 나쁜 코드입니다.

4.6.3. 개념 3 — Reflection-in-Action (Schön) + 세 가지 진단 질문 (5분)

철학자 Donald Schön은 전문가의 실천을 연구하면서 중요한 것을 발견했습니다.

전문가는 행동 전에 완벽한 계획을 세우지 않습니다. 행동하면서 생각하고, 행동의 결과로 문제를 더 깊이 이해합니다. 그는 이것을 Reflection-in-Action이라고 불렀습니다.

설계도를 다 그리고 집을 짓는 건축가가 아니라, 흙을 만지면서 구조를 이해하는 도예가의 이미지입니다.

이것은 미숙함이 아닙니다. 복잡한 문제에 대한 성숙한 인식론입니다.

세 가지 진단 질문 — 통합 정리:

```
이 모호함 앞에서 나는 어떻게 할 것인가?

질문 1: 이것을 알기 위해 내가 할 수 있는 행동이 있는가?
         있다면 → Resolvable → Weinberg 전략 (질문, 인터뷰)

질문 2: 행동해도 여전히 불확실하다면?
         → Essential → Cunningham 전략 (Spike, 최소 행동)

질문 3: 지금 이 결정이 꼭 필요한가?
         필요하지 않다면 → 결정을 미루는 것이 전략
         (Fairbanks의 "Just Enough" 정신)
```

4.7. 3. 고정 케이스 실습 (30분)

4.7.1. 소그룹 공유 가이드

소그룹 공유 (5분)

퍼실리테이터 가이드:

```
비교 포인트
──────────

"같은 모호함을 R로 분류한 팀과 E로 분류한 팀이 있는가?"
  → 왜 다르게 봤는지가 핵심입니다.
    실제로 행동이 가능한데 E로 분류한 것은 없는가?
    반대로, 아무도 모르는데 R로 분류한 것은 없는가?

"Spike를 설계할 때 가장 막힌 것은 무엇인가?"
  → '가장 단순한'의 기준을 잡는 것.
    "이것보다 더 단순하게 할 수 있는가?"를
```

```
    반복해서 묻도록 유도합니다.

"'지금 결정 안 해도 된다'는 칸을 활용했는가?"
  → 이 판단 자체가 중요한 역량입니다.
    모든 모호함을 지금 해소해야 한다는 강박에서 벗어나는 것.
```

4.8. 4. 현업 케이스 슬롯 (20분)

4.8.1. 워크시트 B-5 | 내 현업 케이스: 모호함 재분류

4.8.2. 디브리프 가이드

4.9. 5. 디브리프 (10분)

퍼실리테이터 가이드 — 예상 저항과 대응

```
"Essential Ambiguity가 있으면 그냥 만들면 되는 건가요?
 막 만들자는 얘기예요?"
──────────────────────────────────────────────────────────
아닙니다. 핵심 차이가 있습니다.

  그냥 만드는 것:       모호함을 인식하지 못하고 만든다
  Spike:                  모호함을 인식하고, 그것을 탐색하기 위해
                          의도적으로 최소한으로 만든다

Spike는 목적이 있습니다 — 학습. 그리고 버릴 것을 압니다.
목적 없이 만드는 것은 Spike가 아닙니다.

"Resolvable과 Essential을 잘못 판단하면 어떻게 되나요?"
──────────────────────────────────────────────────────────
괜찮습니다. Spike를 하다 보면 Resolvable인지 Essential인지
더 명확해집니다. 진단이 완벽하지 않아도 됩니다.
시작하는 것이 더 중요합니다.

"Technical Debt를 의도적으로 지는 게 좋은 건가요?"
──────────────────────────────────────────────────────
모든 상황에서 좋은 건 아닙니다.
핵심은 — '지고 있다는 것을 알고 있는가'입니다.
모르고 지는 부채와, 알고 지는 부채는 다릅니다.
알고 있다면 언제 갚을지 계획할 수 있습니다.

"Weinberg는 모호함을 해소하라고 했는데,
 Cunningham은 안고 가라고 합니다. 뭐가 맞아요?"
──────────────────────────────────────────────────
둘 다 맞습니다. 다른 상황의 이야기입니다.
Weinberg: 해소 가능한 모호함 → 해소하라
```

```
Cunningham: 해소 불가능한 모호함 → 안고 걸어라
오늘 배운 진단 능력이 그 판단을 하는 도구입니다.
```

Chapter 5. 이 문제, 어디서 본 것 같은데 — 유사 추론

5.1. 시간 배분

순서	내용	시간
	0. 전강 연결	5분
	1. 오프닝 도발 — 같은 문제 다른 도메인	10분
	2. 개념 설명 (유사 추론, 환원, 비즈니스 리서치)	20분
	3. 고정 케이스 실습 — 워크시트 A-6	30분
	4. 현업 케이스 슬롯 — 워크시트 B-6	20분
	5. 디브리프	10분
	6. 과제 카드 #6 배포	5분
	합계	100분

5.2. 강의 운영 주의사항

- ★ 유사 추론의 핵심은 '해결책 복사'가 아니라 '원리 역수입'입니다. 혼동하는 수강생에게 반복해서 환기시키세요.
- ★ 같은 문제에서 전혀 다른 도메인의 유사 사례가 나왔을 때 — 둘 다 맞다고 해주세요. 차이에서 오히려 더 많이 배웁니다.
- ★ 비즈니스 리서치에서 '기능 목록'이 아니라 'Trade-off'를 읽는 것임을 강조하세요. '이 서비스는 무엇을 포기했는가?'를 묻게 하세요.
- ★ '경험이 쌓이는 방식'이 이 능력임을 짚어주세요. 훈련할 수 있다는 것.

5.3. 강의 진행 상세

5.4. 0. 전강 연결 (5분)

```
지금까지의 여정:
  1강: Solution Space로의 조기 도약 인식
  2강: FAC로 요구사항 해부
  3강: Context-Free Questions로 탐색
  4강: 증상/문제/근본원인, Domain Model
  5강: Resolvable vs Essential, Spike

이번 강은 Problem Space 탐색의 마지막 층입니다.
문제를 충분히 이해했다면 – 이제 묻습니다.
"이런 종류의 문제, 세상 어딘가에서 이미 풀리지 않았나?"
```

```
5강 과제에서 가져온 것 수거:
"해봐야 알았던 경험을 찾아봤을 때
 그 모호함은 R이었나요, E였나요?" → 2-3명 짧게 공유
```

5.5. 1. 오프닝 도발 (10분)

"같은 문제, 다른 도메인 실험"

슬라이드에 두 가지 상황을 나란히 놓습니다.

```
상황 A                              상황 B
──────────────────────────────      ──────────────────────────────

배달 앱                              소셜 미디어 피드
주문이 들어오면 가장 가까운          팔로잉한 계정이 너무 많아
배달원에게 자동 배정해야 한다.       모든 게시물을 보여줄 수 없다.
모든 배달원이 이미 바쁠 수도         어떤 것을 먼저 보여줄 것인가?
있다.
```

수강생에게 묻습니다.

> "이 두 문제가 같은 종류의 문제처럼 보이는 사람 있나요?

> `어떤 면에서 비슷하다고 생각하시나요?"`

잠시 후 힌트를 줍니다.

> "둘 다 — 한정된 자원(배달원, 화면 공간)에

> `다수의 요청(주문, 게시물)을 어떻게 배분할 것인가의 문제입니다.`

> `도메인은 다르지만 _구조_가 같습니다."`

이어서 말합니다.

> "팀오더의 알림 우선순위 문제도 같은 구조입니다.

> `한정된 주의력(팀원의 주의)에`

> `다수의 알림을 어떻게 전달할 것인가.`

>

```
Polya는 이것을 물었습니다.
```

```
'Do you know a related problem?'
```

```
이미 누군가 비슷한 구조의 문제를 풀어놓았다면,
```

```
그 해결책에서 배울 수 있습니다."
```

5.6. 2. 개념 설명 (20분)

5.6.1. 개념 1 — 유사 추론 (Analogical Reasoning) (10분)

Polya가 _How to Solve It_에서 가장 강조한 것 중 하나입니다.

> "Do you know a related problem?"

문제를 처음 보는 것처럼 대할 필요가 없습니다. 이 문제의 _구조_가 이미 알려진 다른 문제와 닮았다면, 그 해결책의 논리를 가져올 수 있습니다.

유사 추론의 세 단계:

```
1단계: 구조 추출
  이 문제의 핵심 구조는 무엇인가?
  (인물, 행위, 관계, 제약을 추상화)

2단계: 유사 문제 탐색
  이 구조와 닮은 문제가 다른 도메인에 있는가?

3단계: 원리 역수입
  유사 문제의 해결 원리를 이 문제에 적용한다
  (해결책 자체가 아니라 원리를)
```

팀오더 케이스 적용 예:

```
1단계 구조 추출:
  "여러 수신자에게 메시지를 전달하는데,
   수신자의 상태(온라인/오프라인/절전)가 불확실하고,
   메시지의 긴급도가 다르다"

2단계 유사 문제 탐색:
```

```
  • 병원 응급실 트리아지 — 환자 긴급도에 따른 순서
  • 항공 관제 — 여러 항공기의 우선순위 통신
  • 이메일 스팸 필터 — 중요도에 따른 분류
  • TCP 재전송 — 패킷 미수신 시 재시도 메커니즘
  • 트위터 알림 배치 — 중요 알림은 즉시, 나머지는 묶음

3단계 원리 역수입:
  TCP 재전송 원리:
    "전달 확인이 안 오면 재전송한다"
    → 팀원 알림 확인(acknowledgement)이 없으면 재알림
  트리아지 원리:
    "긴급도에 따라 채널을 다르게 쓴다"
    → 야간 긴급은 별도 채널(SMS/전화), 일반은 앱 푸시
```

중요한 주의점: 해결책을 복사하는 것이 아닙니다. 원리를 가져오는 것입니다. TCP를 그대로 쓰는 것이 아니라, "확인 없으면 재시도"라는 논리 구조를 가져옵니다.

5.6.2. 개념 2 — 환원 (Reduction) (5분)

환원은 복잡한 문제를 이미 알려진 _문제 유형_으로 매핑하는 것입니다.

```
"이건 결국 어떤 종류의 문제인가?"
```

자주 쓰이는 문제 유형 어휘:

문제 유형	핵심 구조	전형적 예
Matching 문제	A 집합과 B 집합을 최적으로 연결	배달원-주문, 구직자-채용
Trust 문제	모르는 사람 간의 신뢰 구축	중고거래, 숙박 공유
Cold Start 문제	데이터 없는 초기 상태에서 추천	신규 사용자, 새 제품
Visibility 문제	한쪽이 다른 쪽의 상태를 모름	팀 업무 상황, 물류 위치
Rate Control 문제	처리 속도보다 요청이 많음	API throttling, 알림 피로
Consistency 문제	여러 곳의 데이터가 달라짐	분산 시스템, 멀티 디바이스

팀오더 케이스: Visibility 문제 + Rate Control 문제의 결합

팀장은 팀원의 확인 상태를 모르고(Visibility), 알림이 너무 많으면 무시된다(Rate Control).

이 두 가지 문제 유형을 알고 있다면 — 각 유형에서 검증된 해결 패턴이 있습니다.

5.6.3. 개념 3 — 비즈니스 리서치 관점: Trade-off를 읽는다 (5분)

경쟁 서비스나 유사 서비스를 조사할 때, 단순히 기능을 나열하는 것은 별로 도움이 안 됩니다.

Cunningham의 관점에서 보면 — 다른 서비스는 이미 _Essential Ambiguity를 행동으로 탐색한 결과_를 제품 안에 담고 있습니다.

읽어야 할 것은 _그들이 선택한 Trade-off_입니다.

```
비즈니스 리서치 질문 프레임:

"이 서비스는 무엇을 최우선으로 했는가?"
"무엇을 포기했는가?"
"왜 이 방식을 선택했을까?"
"어떤 사용자/환경에 최적화되어 있는가?"
"어떤 상황에서 이 설계가 실패하는가?"
```

예: 슬랙의 알림 설계 * 최우선: 메시지 전달 확실성 + 알림 커스터마이징 * 포기한 것: 단순함 (설정이 너무 복잡함) * 최적화된 환경: 데스크탑 중심 지식 근로자 * 실패하는 상황: 현장 근무자, 알림 설정을 안 하는 사용자

팀오더의 사용자는 현장 근무자입니다. 슬랙의 설계가 왜 그들에게 안 맞는지가 여기서 보입니다.

5.7. 3. 고정 케이스 실습 (30분)

5.7.1. 소그룹 공유 가이드

소그룹 공유 (10분)

퍼실리테이터 가이드:

```
비교 포인트
_____________
"팀마다 다른 유사 사례를 찾았는가?"
  → 같은 문제에서 전혀 다른 도메인의 유사 사례가 나올 수 있습니다.
    병원 트리아지를 찾은 팀과 TCP 재전송을 찾은 팀이 있다면
    둘 다 맞습니다. 그리고 둘에서 끌어낸 원리가 다를 것입니다.
    그 차이를 칠판에 나열해보세요.

"원리를 역수입할 때 막힌 부분은?"
  → "해결책을 복사하려 했는가, 원리를 가져오려 했는가?"를
    확인합니다. 전자는 대부분 "우리 상황에 안 맞아요"로 끝납니다.

"비즈니스 리서치에서 Trade-off를 읽는 것이
 단순한 기능 비교와 어떻게 달랐는가?"
  → 이 질문을 꺼내서 리서치 관점의 차이를 언어화하게 합니다.
```

5.8. 4. 현업 케이스 슬롯 (20분)

5.8.1. 워크시트 B-6 | 내 현업 케이스: 유사 추론 적용

5.8.2. 디브리프 가이드

5.9. 5. 디브리프 (10분)

퍼실리테이터 가이드 — 예상 저항과 대응

```
"유사 사례를 찾는 게 시간이 너무 걸릴 것 같아요."
──────────────────────────────────────────────────

처음엔 그렇습니다. 하지만 이것은 훈련입니다.
몇 번 하다 보면 문제를 들으면서 자동으로
"이건 어디서 본 구조인데"가 떠오릅니다.
그 순간이 3-4년차에서 5-6년차로 넘어가는 변화 중 하나입니다.

"유사 사례가 틀릴 수도 있지 않나요?"
──────────────────────────────────────

그렇습니다. 유사 추론은 틀릴 수 있습니다.
하지만 '틀렸다'는 것을 발견하는 것도 학습입니다.
"우리 문제는 TCP와 달리 응답 확인을 강제할 수 없다"는 발견이
오히려 문제를 더 깊이 이해하게 만들 수 있습니다.

"이미 알고 있는 것만 유사 사례로 쓸 수 있지 않나요?
 모르는 도메인에서 찾는 건 어떻게 해요?"
──────────────────────────────────────────────────

두 가지 방법이 있습니다.
첫째, 문제 유형 어휘로 검색해보세요.
"Visibility problem solution" "notification fatigue pattern" 같은 검색.
둘째, 팀 안에서 서로 다른 경험을 가진 사람이 다른 유사 사례를
알 수 있습니다. 이것이 다양한 배경의 팀이 유리한 이유입니다.
```

Chapter 6. 어디에 공을 들여야 하는가 — Risk-Driven 설계

6.1. 시간 배분

순서	내용	시간
	0. 전강 연결	5분
	1. 오프닝 도발 — 두 팀의 이야기	10분
	2. 개념 설명 (Quality Attributes, Risk-Driven 3단계, Cunningham 연결)	22분
	3. 고정 케이스 실습 — 워크시트 A-7	30분
	4. 현업 케이스 슬롯 — 워크시트 B-7	20분
	5. 디브리프 + 단순화 선순환	10분
	6. 과제 카드 #7 배포	3분
	합계	100분

6.2. 강의 운영 주의사항

- ★ '완벽하게 설계해야 한다'는 압박이 가장 강한 수강생에게 이 강이 가장 중요합니다.
- ★ Engineering Risk와 PM Risk 혼동이 자주 일어납니다. '일정이 빠듯하다'는 PM Risk임을 명확히 하세요.
- ★ PART 4 '지금 안 해도 된다'는 칸을 채우는 것이 어려운 수강생이 있을 것입니다. '영원히 안 한다'가 아니라 '언제 다시 볼지 계획하는 것'임을 강조하세요.
- ★ 마지막에 '설계 단순화의 선순환'을 꼭 짚어주세요. 이것이 8강으로의 연결 고리입니다.

6.3. 강의 진행 상세

6.4. 0. 전강 연결 (5분)

```
지금까지의 여정:
  1-4강: Problem Space를 깊이 탐색하는 도구들
  5강:   모호함의 두 종류와 Spike
  6강:   유사 추론으로 이미 풀린 구조 찾기

이제 Solution Space로 본격 진입합니다.
6강까지는 "무엇을 만들 것인가"를 이해했다면,
7강부터는 "어떻게 만들 것인가"입니다.

그런데 — "어떻게"에 얼마나 공을 들여야 하는가?
모든 것을 완벽하게 설계해야 하는가?
```

```
6강 과제에서 가져온 것 수거:
"유사 사례에서 역수입한 원리를
 내 케이스에 적용해봤을 때 어떤 변화가 있었나요?"
→ 2-3명 짧게 공유
```

6.5. 1. 오프닝 도발 (10분)

"두 팀의 이야기"

슬라이드나 말로 짧게 제시합니다.

> 팀 A

> 새 기능을 시작하면서 3주 동안 아키텍처 문서를 작성했습니다.

> 모든 컴포넌트, 모든 API, 모든 데이터 흐름을 그렸습니다.

> 개발을 시작했을 때, 이미 두 가지 주요 가정이 틀렸습니다.

> 팀 B

> 새 기능을 시작하면서 "일단 만들자"며 바로 코딩했습니다.

> 3주 후 — 성능 문제가 터졌습니다.

> 아키텍처를 전면 재작업해야 했습니다.

수강생에게 묻습니다.

> "어떤 팀이 맞게 한 건가요?"

잠시 후:

> "둘 다 틀렸습니다.

> 팀 A는 너무 많이 설계했고,

> 팀 B는 너무 적게 설계했습니다.

>

> Fairbanks는 이 두 극단 사이의 중간 경로를 물었습니다.

> '얼마나 설계해야 하는가?'

>

> 그의 답은 — 리스크에 비례해서."

6.6. 2. 개념 설명 (22분)

6.6.1. 개념 1 — Quality Attributes vs Functionality (7분)

먼저 핵심 구분을 정확히 잡아야 합니다.

```
Functionality (기능)
─────────────────────
"시스템이 무엇을 하는가"
예: 알림을 보낸다, 업무를 배정한다, 목록을 보여준다

→ 스프린트마다 쌓을 수 있습니다
→ 나중에 추가하기 상대적으로 쉽습니다


Quality Attributes (품질 속성, the "-ilities")
──────────────────────────────────────────────
"시스템이 얼마나 잘 하는가"
예: 성능(Performance), 가용성(Availability), 보안(Security),
    확장성(Scalability), 변경용이성(Modifiability)

→ 아키텍처 결정으로 확보됩니다
→ 나중에 바꾸면 재작업이 매우 비쌉니다
→ 기능과 달리 서로 Trade-off 관계입니다
```

왜 이 구분이 중요한가:

```
기능을 나중에 추가하는 비용:    선형적
Quality Attribute를 나중에 추가하는 비용: 기하급수적
```

일관성(Consistency)을 나중에 추가하려면 — DB 스키마, API 구조, 캐시 전략을 다 바꿔야 할 수 있습니다. 처음부터 고려했다면 설계 결정 하나로 끝날 수 있는 것입니다.

팀오더 케이스에서 Quality Attributes:

Quality Attribute	팀오더에서의 의미	방치했을 때의 리스크
Reliability	야간 긴급 알림이 반드시 도달해야 함	라인 스탑
Performance	알림 전달 지연이 허용 범위 내여야 함	팀원이 늦게 인지
Modifiability	알림 채널을 나중에 추가할 수 있어야 함	이메일/SMS 추가 시 전면 재작업
Usability	현장에서 확인하기 쉬워야 함	알림을 받아도 행동 안 함

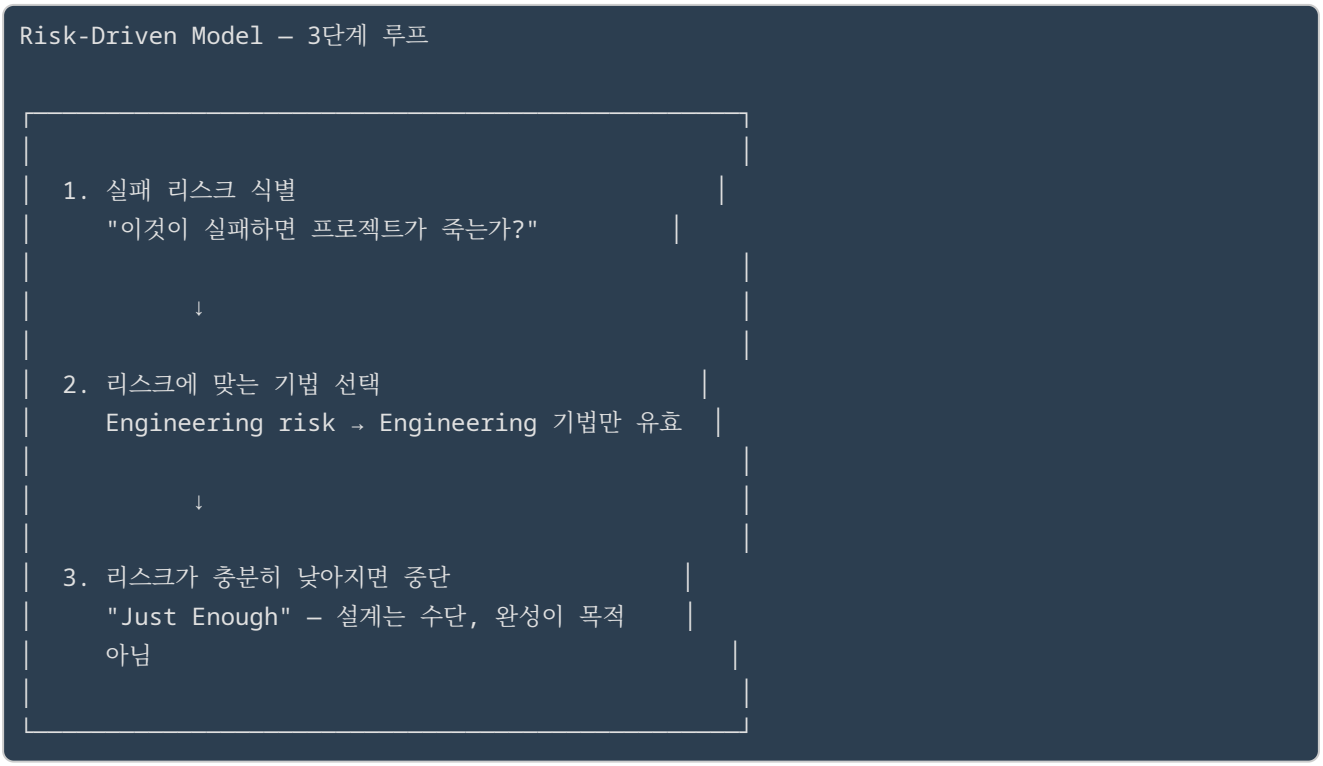
Weinberg의 Attributes(2강)가 여기서 다시 등장합니다. 2강에서 캐내지 못한 Attributes가 7강에서 설계 리스크가 됩니다.

6.6.2. 개념 2 — Risk-Driven Model (Fairbanks) (10분)

Fairbanks의 핵심 질문:

> "How much software architecture should you do?"

그의 답: 리스크에 비례해서.



Engineering Risk vs Project Management Risk

이 구분이 3-4년차에게 가장 실용적인 부분입니다.

```
──────────────────────────────  ──────────────────────────────────
시스템 분석/설계/구현의 문제    일정, 팀 규모, 우선순위의 문제

"성능이 요구사항을 못 맞춘다"  "3주 안에 못 끝낼 것 같다"
"데이터 일관성을 보장 못 한다" "팀원이 이 기술을 모른다"
"보안 취약점이 있다"            "요구사항이 계속 바뀐다"

→ Engineering 기법으로 해소     → PM 기법으로 해소
  (프로토타입, 부하 테스트,       (일정 조정, 교육,
   threat modeling 등)             프로세스 개선 등)
```

리스크 유형 ≠ 기법 유형이면 무효:

성능 리스크에 클래스 다이어그램을 그리는 것은 무효입니다. 성능 리스크는 부하 테스트나 queuing analysis 같은 기법이 필요합니다. 일정 리스크에 아키텍처 다이어그램을 그리는 것도 무효입니다.

리스크 식별 실용 체크리스트:

```
아래 항목 중 "YES"가 있으면 Engineering Risk로 간주

□ 이 시스템의 응답 시간 요구사항이 까다로운가?
□ 데이터 일관성이 중요한 시스템인가?
□ 보안 민감 데이터를 다루는가?
□ 트래픽이 급격히 증가할 가능성이 있는가?
□ 여러 팀이 같은 컴포넌트를 건드리는가?
□ 특정 컴포넌트 장애가 전체 장애로 이어지는가?
□ 나중에 기능 추가/변경이 많이 예상되는가?
□ 외부 시스템/서비스와의 통합이 복잡한가?
```

6.6.3. 개념 3 — Cunningham과의 연결 (5분)

Risk-Driven Model과 5강의 Cunningham이 만나는 지점을 짚어줍니다.

Fairbanks의 "리스크가 낮아지면 중단"은 Cunningham의 "What is the simplest thing that could possibly work?"와 같은 정신입니다.

```
리스크가 낮은 영역:
  정교한 설계를 하는 것이 오히려 낭비
  → Cunningham: Simplest Thing
  → Fairbanks: 설계 중단

리스크가 높은 영역:
  설계를 충분히 하지 않는 것이 실패의 원인
  → Fairbanks: 리스크에 맞는 기법 선택
  → Cunningham: Spike로 빠르게 탐색
```

결론: "얼마나 설계할 것인가"의 기준은 "얼마나 완성하고 싶은가"가 아니라 "어디에 실패 리스크가

있는가"입니다.

6.7. 3. 고정 케이스 실습 (30분)

6.7.1. 소그룹 공유 가이드

소그룹 공유 (8분)

퍼실리테이터 가이드:

```
비교 포인트
──────────

"팀마다 상위 3개 리스크가 다른가?"
  → 같은 시스템인데 리스크 우선순위가 다르다면
    어떤 맥락 차이가 그 판단을 만들었는가?
    데모 관점 vs 실서비스 관점의 차이일 수 있습니다.

"PM Risk와 Engineering Risk를 혼동한 경우가 있는가?"
  → 가장 자주 섞이는 것:
    "1개월 안에 못 만들 것 같다" → PM Risk
    "야간 알림 전달을 보장 못 할 것 같다" → Engineering Risk
    이 둘에 대응하는 기법이 완전히 다름을 짚어줍니다.

"PART 4 — '지금 안 해도 된다'는 결정이 어려웠는가?"
  → 어려웠다면 — 왜 어려웠는가?
    "완벽하게 설계해야 한다"는 압박과 어떻게 타협하는가?
    이것이 이 강의 핵심 인식 전환입니다.
```

6.8. 4. 현업 케이스 슬롯 (20분)

6.8.1. 워크시트 B-7 | 내 현업 케이스: Risk 식별 + 설계 에너지 배분

6.8.2. 디브리프 가이드

6.9. 5. 디브리프 (10분)

퍼실리테이터 가이드 — 예상 저항과 대응

```
"리스크를 미리 다 알 수는 없잖아요."
──────────────────────────────────

맞습니다. 완전히 알 수 없습니다.
하지만 두 가지를 기억하세요.

첫째, 알 수 있는 리스크에 집중하는 것이 시작입니다.
알 수 없는 리스크는 운영 중에 발견하고 대응합니다.
```

```
둘째, 6강의 유사 추론이 여기서 도움이 됩니다.
비슷한 구조의 시스템에서 자주 나오는 리스크가 있습니다 —
'Prototypical Risk'입니다. 같은 유형의 문제에서
반복되는 리스크 패턴을 알면 blind spot이 줄어듭니다.

"Quality Attributes를 어떻게 수치로 정의해요?"
──────────────────────────────────────────────

2강 Attributes로 돌아옵니다.
"응답 시간"이라는 Quality Attribute를
"95th percentile 200ms 이하"로 수치화한 것이
3강의 Context-Free Questions로 끌어낸 것입니다.
수치가 없는 Attribute는 검증할 수 없습니다.

"리스크가 낮은 것을 지금 안 해도 된다는 게
 나중에 더 어려워지지 않나요?"
──────────────────────────────────────────────

그럴 수도 있습니다. 그래서 PART 4에서
'언제 다시 검토할 것인가'를 적게 했습니다.
"지금 안 한다"는 결정은 "영원히 안 한다"가 아닙니다.
리스크가 커지는 시점을 모니터링하고,
그 시점에 다시 투자하는 것입니다.
```

설계 단순화의 선순환

```
Risk-Driven 접근의 가장 중요한 결과를 하나 더 짚고 마무리합니다.

"지금 안 해도 된다"는 것이 명확해질수록,
"지금 해야 한다"는 것에 집중할 수 있습니다.

집중할 수 있으면 → 그것을 더 빠르게 완성할 수 있습니다.
더 빠르게 완성하면 → 더 일찍 사용자에게 전달할 수 있습니다.
더 일찍 전달하면 → 실제 피드백을 더 일찍 받을 수 있습니다.
피드백이 있으면 → 다음 무엇을 만들지를 더 정확하게 알 수 있습니다.

Problem Space 탐색과 Risk-Driven 설계가 느리게 만드는 것처럼
보이지만, 실제로는 이 선순환을 가능하게 합니다.

물론 미룬 것들은 언젠가 돌아옵니다.
하지만 그때는 — 핵심이 이미 사용자 손에 있습니다.
그리고 검증된 핵심 위에 확장하는 것은,
검증 없이 처음부터 모든 것을 만드는 것보다 훨씬 안전합니다.
```

Chapter 7. 승현의 지도 — 해결책의 검증과 통합

7.1. 시간 배분

순서	내용	시간
	0. 전강 연결 — 8강 전체 여정 요약	5분
	1. 오프닝 도발 — 두 팀의 데모	10분
	2. 개념 설명 (Problem-Solution Fit, Pre-mortem)	15분
	3. 고정 케이스 실습 — 워크시트 A-8	30분
	4. 현업 케이스 통합 발표 — 워크시트 B-8	25분
	5. 디브리프 — 워크샵 전체 수렴	10분
	6. 마지막 카드 배포	5분
	합계	100분

7.2. 강의 운영 주의사항

- ★ 마지막 강입니다. 도구 사용보다 통합과 회고에 집중하세요.
- ★ PART 3 '1강과 지금을 비교'에서 '만들지 않아도 되는 것이 생겼다'에 체크한 사람을 반드시 꺼내세요. 이것이 워크샵 전체의 핵심 결론입니다.
- ★ Pre-mortem 실패 이유들을 1-7강 도구와 연결시키는 발문을 하세요. '이 실패 이유, 몇 강에서 쓴 도구로 잡을 수 있었을까요?'
- ★ 현업 케이스 발표는 강제하지 마세요. 자율적으로 진행하되, 발표자에게 충분한 피드백 시간을 주세요.
- ★ 마지막 카드 배포 후 짧게 침묵을 주세요. 8강의 여운을 느끼게 하세요.

7.3. 강의 진행 상세

7.4. 0. 전강 연결 (5분)

```
8강에 걸친 여정 요약:

Problem Space 탐색 도구들
  1강: Solution Space 조기 도약 인식
  2강: FAC로 요구사항 해부
  3강: Context-Free Questions + Black Box
  4강: 증상/문제/근본원인, Domain Model

모호함 다루기
  5강: Resolvable vs Essential, Spike
```

```
Solution Space 탐색 도구들
  6강: 유사 추론 + 비즈니스 리서치
  7강: Risk-Driven 설계 에너지 배분

이번 강:
  8강에서 배운 것을 들고 → 해결책을 검증한다
  그리고 돌아본다 —
  "문제를 깊이 이해한 것이 실제로 무엇을 바꿨는가?"

7강 과제에서 가져온 것 수거:
"가장 중요한 리스크에 대응 기법을 써봤을 때
 어떤 결과가 나왔나요?" → 2-3명 짧게 공유
```

7.5. 1. 오프닝 도발 (10분)

"두 팀의 데모"

> 팀 A

> 1개월간 빠르게 만들었습니다.

> 알림 기능, 업무 배정, 대시보드, 리포트 기능까지.

> 데모에서 고객사 팀장이 말했습니다.

> "이거… 저희가 원한 게 아닌 것 같아요."

> 팀 B

> 1개월간 세 가지만 만들었습니다.

> 업무 배정 알림, 수신 확인, 야간 긴급 채널.

> 데모에서 고객사 팀장이 말했습니다.

> "이거 바로 쓸 수 있겠네요."

수강생에게 묻습니다.

> "팀 B가 세 가지만 만든 것은 게으름인가요,

```
아니면 좋은 판단인가요?"
```

잠시 후:

> "팀 B는 문제를 깊이 이해했기 때문에

```
세 가지가 전부라는 것을 알았습니다.
```

```
팀 A는 문제를 이해하지 못한 채 만들었기 때문에
```

```
무엇이 필요한지 몰라서 많이 만들었습니다.
```

>

> 오늘 강의의 핵심입니다.

> 문제를 제대로 이해하면, 해결책은 오히려 단순해집니다.

> 무엇을 만들지가 명확해지는 동시에,

> 무엇을 만들지 않아도 되는지가 보입니다.

> 그리고 그 단순한 것을 먼저 전달할 수 있습니다."

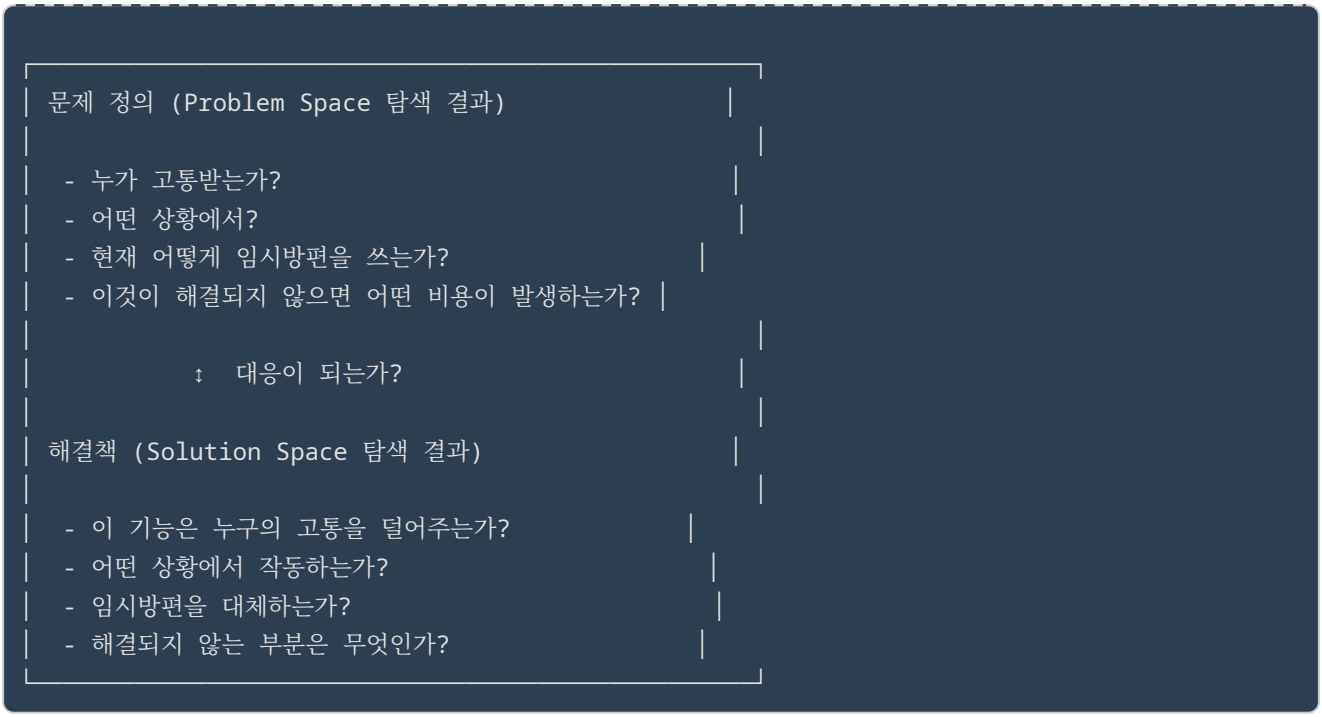
7.6. 2. 개념 설명 (15분)

7.6.1. 개념 1 — Problem-Solution Fit (7분)

해결책을 만들기 전에, 또는 만들면서 지속적으로 확인해야 할 것:

내가 정의한 문제와 내가 만든 해결책이 실제로 연결되어 있는가?

```
Problem-Solution Fit 체크:
```



가장 흔한 불일치:

```
문제: "팀원이 업무를 놓쳐서 라인이 멈춘다"
해결책: "예쁜 알림 UI + 읽음 표시 + 알림 히스토리 페이지"

→ 읽음 표시와 히스토리 페이지가 라인 스탑을 막는가?
  문제의 핵심(전달 보장)이 빠져 있다.
```

Fit 체크의 실용적 질문 세 가지:

1. **"이 기능이 없었다면 사용자는 어떻게 했을 것인가?"** → 임시방편이 그대로 남는다면, Fit이 안 된 것.
2. **"이 기능이 있어도 문제가 여전히 발생할 수 있는가?"** → 그렇다면 핵심을 놓친 것.
3. **"만들지 않은 것 중에 이 문제 해결에 필요한 것이 있는가?"** → 있다면 지금 해야 하는가, 나중에 해도 되는가를 결정.

7.6.2. 개념 2 — Pre-mortem: 역방향 시뮬레이션 (8분)

Gary Klein이 제안한 기법입니다.

일반적인 리뷰: "이 해결책이 왜 성공할 것인가?" Pre-mortem: "이 해결책을 도입했는데 6개월 후 실패했다고 가정하면, 이유는 무엇인가?"

왜 역방향이 더 효과적인가:

일반 리뷰	Pre-mortem
성공 편향이 생긴다	실패를 상상하므로 편향이 줄어든다
"잘 될 거야"	"실패했다면 왜?"

```
약점을 찾기 어렵다              약점이 자연스럽게 나온다
팀 분위기상 비판하기 어렵다    가상의 실패이므로 심리적 안전
```

Pre-mortem 수행 방법:

```
1. "이 해결책을 실제로 도입했다고 가정하세요."
2. "6개월이 지났습니다."
3. "실패했습니다. 어떤 이유로?"
4. 개인별로 3가지 이상 적기 (2분)
5. 소그룹 공유 → 가장 많이 나온 실패 이유 상위 3개 추출
6. 그 이유를 미리 대응할 수 있는가?
```

팀오더 케이스 Pre-mortem 예시:

```
"6개월 후 팀오더 알림 기능이 실패한 이유들"

• 야간 긴급 SMS 비용이 예상보다 많이 나와서 기능을 끔
• 팀원들이 알림을 너무 많이 받아서 끄기 시작함
• iOS 업데이트 후 백그라운드 알림 정책이 바뀜
• 팀장이 수신 확인 안 한 팀원에게 어떻게 해야 할지 몰라 안 씀
• 공장 시스템 방화벽이 FCM 차단함 (나중에 발견)
```

이 중 지금 미리 대응할 수 있는 것: * SMS 비용 → 월간 상한선 설정, 사용자가 직접 긴급 플래그 설정 * 알림 피로 → Rate Control 로직 (6강의 유사 추론) * 방화벽 → 고객사 네트워크 요구사항을 Constraint로 확인 (2강 FAC)

7.7. 3. 고정 케이스 실습 (30분)

7.7.1. 소그룹 공유 가이드

소그룹 공유 (8분)

퍼실리테이터 가이드:

```
이 강의 핵심 비교 포인트
_____________________

"PART 3에서 '만들지 않아도 되는 것이 생겼다'에
 체크한 사람이 있는가?"
  → 이것이 이 강의, 그리고 워크샵 전체의 핵심 결론입니다.
    그들에게 물어보세요:
    "무엇이 사라졌나요? 왜 사라져도 된다고 알게 됐나요?"

"해결책이 더 단순해졌는가?"
  → 더 단순해진 팀: 어느 강에서 그 단순화가 생겼는가?
    (2강 FAC? 4강 레이어 선택? 5강 Spike? 7강 Risk?)

"Pre-mortem에서 나온 실패 이유 중
```

```
 2강-4강의 도구로 미리 잡을 수 있는 것이 있는가?"
  → 이 연결이 보이면 워크샵 전체의 학습이 통합된 것입니다.
```

7.8. 4. 현업 케이스 슬롯 — 최종 통합 (25분)

8강의 현업 케이스 슬롯은 평소와 다릅니다. 개별 워크시트 작업이 아니라, 1강부터 8강까지 현업 케이스에 적용한 것을 통합 발표하는 시간입니다.

7.8.1. 워크시트 B-8 | 내 현업 케이스: 최종 통합 발표

7.8.2. 디브리프 가이드

7.9. 5. 디브리프 (10분)

예상되는 어려움과 대응

```
"기능이 너무 적은 것 같아서 불안했다"
  → 이것이 학습의 핵심 순간입니다. "딱 필요한 것만"이 왜
    칭찬인지를 팀오더 결과로 직접 보여주세요.

"워크시트 B가 완성되지 않은 것 같다"
  → 완성이 목표가 아닙니다. 8강을 거치며 문제 이해가
    어떻게 깊어졌는지의 흔적이 남은 것으로 충분합니다.

"도구를 실제로 쓸 수 있을지 모르겠다"
  → "하나만" 기억하게 하세요.
    요구사항 받으면 → 딱 한 번 멈추고 → 물어보는 것.
    나머지는 경험 속에서 자연스럽게 붙습니다.
```

퍼실리테이터 가이드 — 이 강의 핵심 수렴

```
8강 전체를 관통하는 하나의 메시지로 수렴합니다.

"문제를 제대로 이해하면, 해결책은 오히려 단순해진다."

이것이 이 워크샵 전체의 결론입니다.

1강에서 30초 만에 떠올린 것들을 기억하시나요?
그때의 해결책과 지금의 해결책이 다르다면,
그 차이가 어디서 왔는지 물어보세요.

2강 FAC에서 Attributes를 파악했기 때문에
불필요한 기능을 만들지 않아도 됐을 수 있습니다.

4강 레이어 선택에서 "증상이 아니라 문제를 고친다"고
결정했기 때문에 범위가 좁아졌을 수 있습니다.
```

```
5강 Spike에서 기술적 불확실성을 해소했기 때문에
잘못된 방향으로 2주를 쓰지 않았을 수 있습니다.

7강 리스크 식별에서 "지금 안 해도 된다"는 것을
알게 됐기 때문에 데모에 맞는 집중이 가능했을 수 있습니다.

이 모든 것이 Problem Space 탐색의 결과입니다.
더 오래 이해했기 때문에, 더 적게, 더 정확하게 만들 수 있었습니다.
```

7.10. 6. 워크샵 마무리 (5분)

과제 카드 #8 — 이번엔 과제가 아닙니다

```
┌──────────────────────────────────────────────┐
│ 마지막 카드                                  │
│                                              │
│ 과제가 없습니다.                             │
│                                              │
│ 대신, 이것 하나를 가지고 가세요.             │
│                                              │
│ 다음 번에 요구사항을 받았을 때,              │
│ Solution Space로 넘어가기 전에               │
│ 딱 한 번만 멈추고 물어보세요.                │
│                                              │
│ "나는 지금 무엇을 알고 있는가?               │
│  그리고 무엇을 아직 모르는가?"               │
│                                              │
│ 그 질문이 이 워크샵 전체입니다.              │
└──────────────────────────────────────────────┘
```

Chapter 8. 운영 팁 — 전체

8.1. 심리적 안전 확보

이 워크샵의 오프닝 도발들은 수강생이 자신의 기존 습관을 돌아보게 합니다. 비판하거나 틀렸다는 분위기가 만들어지면 학습이 닫힙니다. '지연의 선택이 틀린 게 아니다. 다른 방법이 있었다'는 프레임을 유지하세요.

8.2. 수강생이 막힐 때

워크시트에서 막히는 가장 흔한 지점:

막히는 지점	퍼실리테이터 개입 방법
FAC에서 Attributes 칸이 비어있다	'이것을 아는 게 왜 중요할까요?'를 물어보세요
Domain Model에서 State가 안 나온다	'이 개념이 가질 수 있는 상태 목록을 나열해보세요'를 주세요
R/E 분류가 어렵다	'지금 내가 취할 수 있는 행동이 있나요?'를 물어보세요
유사 사례가 안 떠오른다	'이 문제의 구조를 한 문장으로 추상화해보세요'를 먼저 하게 하세요
Risk가 너무 많이 나온다	'이것들 중 프로젝트를 죽일 수 있는 것은?'으로 좁혀주세요

8.3. 워크시트 B 관리

워크시트 B는 8강에 걸쳐 같은 케이스를 들고 다닙니다. 강 시작 시 'B 워크시트를 꺼내주세요'를 매번 안내하세요. 1강에서 케이스를 등록 못 한 수강생은 2강에서 등록해도 됩니다.

8.4. 과제 카드 연계

각 강의 시작 5분에 이전 강 과제 결과를 2-3명이 짧게 공유합니다. 강제하지 말고 자율적으로 운영하되, 흥미로운 사례가 있으면 꼭 꺼내세요. 이 공유가 워크샵의 '현장감'을 만드는 핵심입니다.