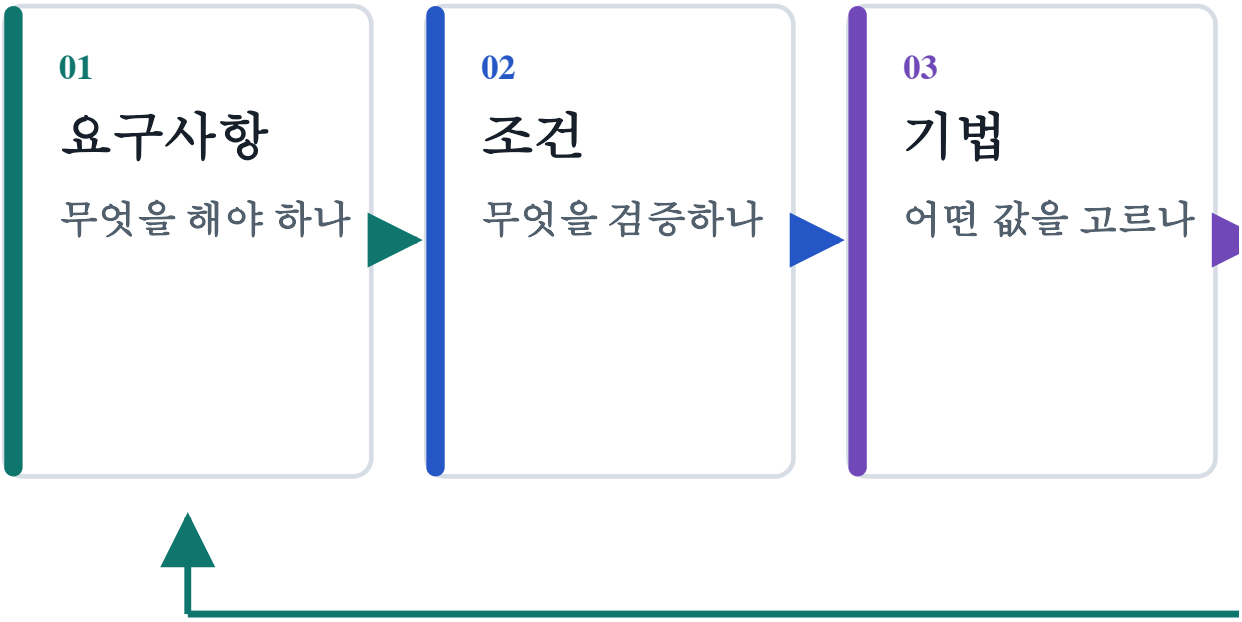
M10-01 · GIBALJA VISUAL MANUAL

요구사항에서 테스트 케이스 만들기

한 문장 목표: 요구사항 문장을 바로 클릭 절차로 옮기지 않고, 판정할 조건과 대표값을 설계 기법으로 고른 뒤, 누가 실행해도 같은 결론을 내리는 테스트 케이스와 추적 증거로 바꿉니다.

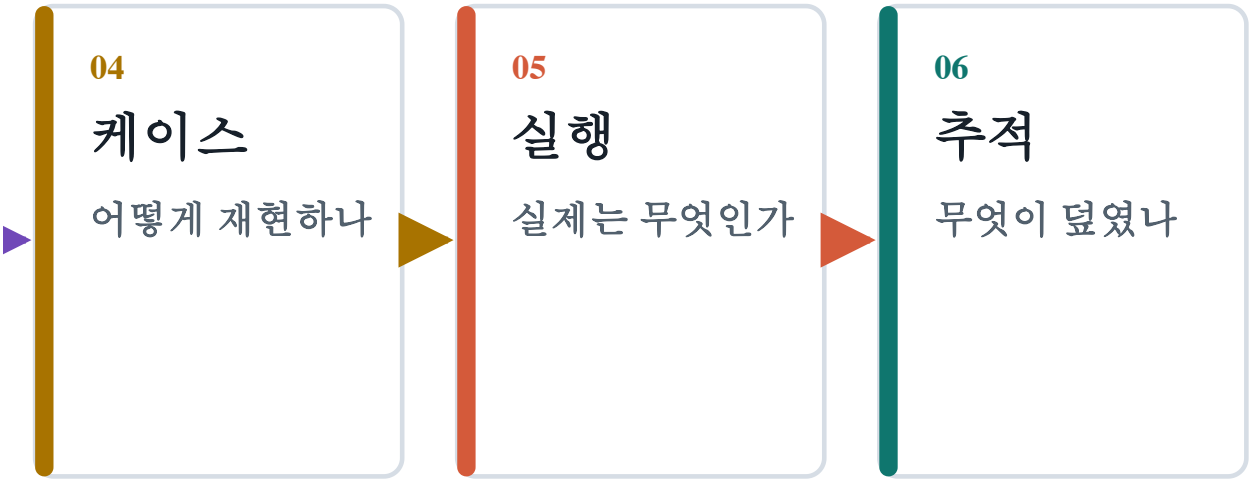
그림 1. 요구사항에서 테스트 증거까지 여섯 단계. 좋은 테스트는 REQ→조건→기법→TC→결과→결함을 양방향으로 연결합니다.

문장을 바로 절차로 옮기지 않고 검증 조건과 설계 기법을 거칩



변경된 요구사항은 영향받는

니다.



케이스와 회귀로 돌아옵니다

0. 한눈에 보는 두 번째 지도

그림 2. “한번 눌러 봤다”는 경험이고, version·시작 상태·입력·oracle·실제 결과·증거가 있어야 테스트 계약입니다.

다른 사람이 같은 조건에서 같은 판정을 내릴 수 있어야 합니다

A

예시만 있는 테스트

로그인해 본다
예약 버튼을 눌러 본다
잘 되면 통과
데이터·기대값·version 없음

VERSI

.

B

재현 가능한 테스트

REQ·TC ID와 version

ONED

사전 조건·입력·행동

관찰 가능한 기대 결과

실제 결과·판정·evidence

난이도	그림 먼저	개념·판정	실습	셀프 테스트	최종 산출물
Level 2	30분	70분	80분	15분	요구사항·케이스·RTM·release evidence

요구사항이 “적절히”, “빠르게”, “가능하면”으로 쓰여 있으면 테스트를 많이 만들어도 판정은 흔들립니다. 반대로 조건을 명확히 해도 대표값 선택 근거가 없으면 중요한 경계와 조합이 빠집니다. 이 매뉴얼은 문장을 해석하는 단계, 값을 고르는 단계, 실행과 판정을 기록하는 단계를 분리합니다. 그래야 테스트 실패가 제품 결함인지, 잘못된 요구사항·케이스·환경·기대값인지 진단할 수 있습니다.

0.1 첫 두 장에서 기억할 여덟 문장

```
요구사항 문장은 테스트 절차가 아니라 test basis다.
먼저 무엇을 검증할지 test condition으로 분해한다.
대표값은 감이 아니라 설계 기법으로 고른다.
좋은 case는 precondition·action·observable expected를 가진다.
Oracle은 “정상 화면”이 아니라 비교 가능한 값·상태·부작용이다.
FAIL은 곧 제품 defect가 아니므로 계약·case·환경·oracle을 먼저 본다.
요구사항과 결과는 양방향으로 추적해야 변경 영향을 찾을 수 있다.
수정된 실패는 regression case와 evidence package로 남긴다.
```

1. 이 PDF를 공부하는 방법

1.1 1회차 · 그림 16장만 읽기 · 30분

그림의 제목과 캡션만 읽습니다. 다음 연결을 말할 수 있으면 됩니다.

```
test basis → testable feature → test condition
→ EP·BVA·decision table·state transition
→ case·data·oracle → run·verdict·defect
→ traceability·coverage → release·regression
```

1.2 2회차 · 테스트 설계 스튜디오 세 variant · 50분

실습 생성기를 실행합니다.

```
./02_Labs/G10_Test_Quality_Security/L10-01_create-requirement-test-practice.sh
```

세 variant를 순서대로 실행합니다.

```
boundary-bug-v1
  → rule-bug-v2
  → candidate-v3
```

첫 variant에서는 경계값의 힘을, 두 번째에서는 결정표·상태전이·멱등성 회귀를, 세 번째에서는 16/16 통과와 6/6 추적 gate를 봅니다.

1.3 3회차 · 내 기능에 적용 · 115분

단계별 실습서를 따라 요구사항·테스트 케이스·추적성 기록서를 채웁니다. 낯선 표현은 요구사항·테스트 케이스 설계 용어집에서 찾습니다.

1.4 읽는 순서보다 중요한 손의 순서

먼저 하지 않을 것	먼저 할 것
화면 클릭 step부터 작성	REQ의 대상·상태·입력·조건·결과 질문
모든 값을 무작정 나열	처리 결과가 같은 partition 식별
대표 중간값만 실행	경계 바로 안·밖 선택
if 문을 머릿속으로 기억	결정표의 feasible column 열거
최종 화면만 확인	code·state·quantity·side effect 비교
FAIL을 곧바로 개발자에게 전달	case·environment·oracle·evidence 진단

2. 학습 outcome과 경계를 고정합니다

2.1 이번 실습의 합성 시스템

```
actor: YEONCORE-LAB 합성 교육 운영자
target: synthetic_session_booking
requirements: 6
test_cases: 16
techniques:
  - example
  - equivalence partitioning
  - boundary value analysis
  - decision table
  - state transition
  - regression
variants:
  - boundary-bug-v1
  - rule-bug-v2
  - candidate-v3
decisions:
  - defects_detected
  - release_ready
```

외부 API·실제 예약·고객 데이터·결제·비용이 없습니다. 고정된 합성 규칙을 Python 표준 라이브러리로 실행하므로 학습자는 결과가 변하는 이유를 요구사항과 구현 규칙에서 직접 추적할 수 있습니다.

2.2 이전·다음 매뉴얼과의 경계

연결	여기서 가져오는 것	이번 매뉴얼이 더하는 것
M07-02 작업 분해·완료 조건	작은 작업과 완료 조건	완료 조건을 대표 테스트 케이스와 oracle로 바꾸기
M09-05 AI 응답 평가	case·expected·grader·회귀	결정론적 제품 요구사항에 EP·BVA·결정표·상태전이 적용
M10-02 정상·예외·권한·보안 시나리오	다음 단계	이번에는 요구사항 한 건을 재현 가능한 case로 바꾸는 기본기를 고정
M12-02 요구사항 작성	이후 전체 요구 명세	여기서는 주어진 요구사항의 testability를 검토하고 질문으로 되돌려 보내기

이번 매뉴얼은 성능 부하 모델, 침투 테스트, 법률 적합성, 실제 결제 검증을 완료했다고 주장하지 않습니다. M10-02에서 정상·예외·권한·보안 시나리오 검수로 범위를 넓힙니다.

2.3 테스트가 증명하는 것과 증명하지 않는 것

증명 가능한 주장	이 테스트만으로 증명하지 못하는 주장
명시된 조건에서 관찰 결과가 기대와 같음	모든 가능한 입력과 운영 상황에서 결함이 없음
선택한 partition·boundary·rule·transition을 덮음	실제 사용자 분포 전체를 대표함
특정 build·환경·data에서 재현됨	다른 browser·network·timezone에서도 같음
추적된 요구사항에 대한 evidence가 있음	추적되지 않은 요구나 암묵적 기대까지 만족함

3. Test basis에서 증거까지 여섯 단계를 분리합니다

ISTQB CTFL v4.0.1은 test analysis를 test basis에서 testable features와 prioritized test conditions을 찾는 활동으로, test design을 그 conditions에서 test cases와 testware를 만드는 활동으로 구분합니다. 이 구분을 지키면 “무엇을 검증할까”와 “어떤 값으로 실행할까”가 섞이지 않습니다.

단계	핵심 질문	산출물
1. 요구사항	무엇을 해야 하나	REQ·owner·version·rationale
2. 조건	무엇을 검증하나	positive·negative test conditions
3. 기법	어떤 값을 고르나	partition·boundary·rule·transition
4. 케이스	어떻게 재현하나	precondition·data·action·expected
5. 실행	실제는 무엇인가	actual·verdict·run evidence
6. 추적	무엇이 덮였나	RTM·coverage·defect·change impact

3.1 문장을 바로 step으로 옮길 때 생기는 문제

```
요구사항: 사용자는 세션을 적절한 인원으로 예약할 수 있다.
```

이를 곧바로 “예약 화면을 열고 2명을 입력해 버튼을 누른다”로 옮기면 다음이 빠집니다.

- 누가 예약할 수 있는가.
- 세션 상태는 무엇인가.

- 적절한 인원은 어떤 범위·형식인가.
- 세션이 닫혔고 정원도 부족하면 어느 오류가 먼저인가.
- 실패했을 때 잔여석과 예약 수는 그대로인가.
- 성공은 어떤 code·state·ID·수량으로 관찰하는가.

3.2 Test condition은 case보다 추상적입니다

수준	예
requirement	좌석 수가 1 미만 또는 4 초과면 저장 없이 거절한다
condition	하한 바로 밖의 값이 거절되고 부작용이 없어야 한다
coverage item	lower outside = 0
test case	OPEN·remaining 10에서 seats=0 요청
expected	code invalid·remaining 10·count 0

Condition은 “무엇을”이고 case는 “어떤 값과 절차로”입니다. 한 condition에 여러 케이스가 필요할 수 있고, 한 케이스가 여러 관련 요구사항을 함께 확인할 수도 있습니다.

3.3 단계별 review gate

gate	통과 질문
requirement gate	ID·owner·version·관찰 결과가 있는가
condition gate	positive·negative·risk가 식별됐는가
design gate	값 선택이 기법과 coverage item으로 설명되는가
case gate	다른 사람이 같은 시작 상태와 oracle을 재현하는가
execution gate	actual·verdict·evidence·build가 기록됐는가
trace gate	orphan·중복·변경 영향이 보이는가

4. 요구사항을 판정 가능한 원자로 쪼갭니다

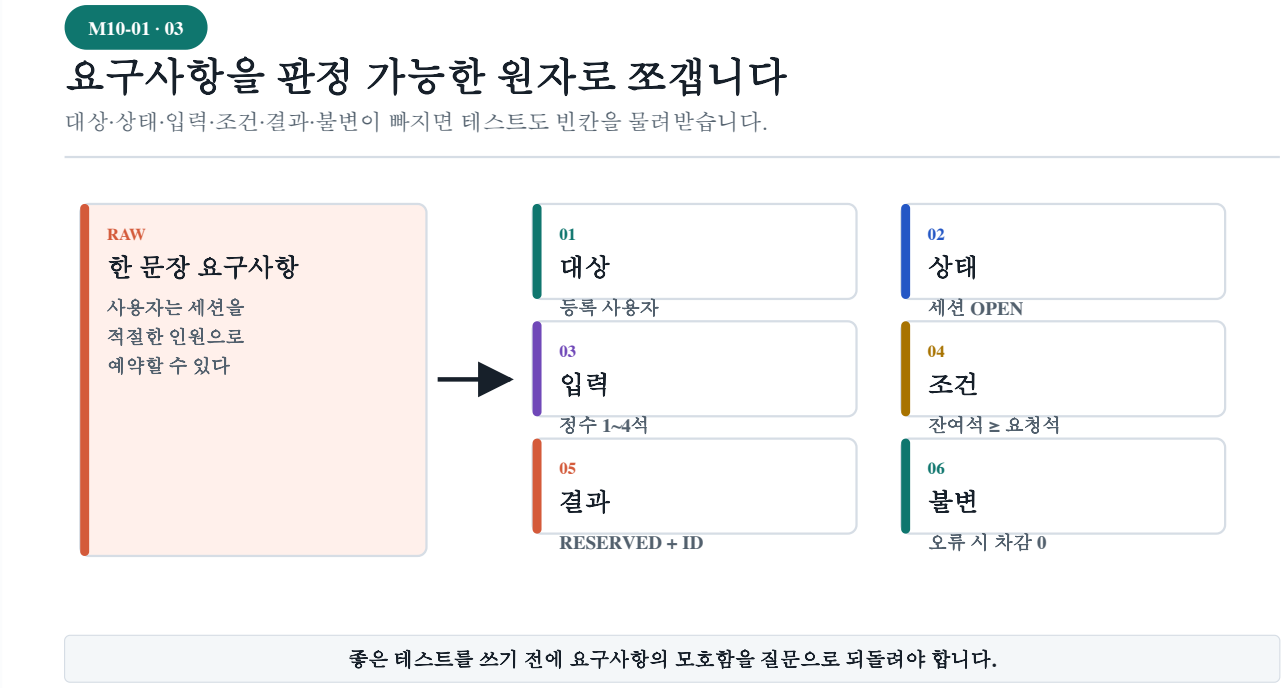


그림 3. 대상·상태·입력·조건·결과·불변식 중 하나가 비면 테스트도 같은 자리에 빈칸을 물려받습니다.

[illegible]

대상·상태·입력·조건·결과·불변이 빠지면 테스트도 빈칸을 물려

RAW

한 문장 요구사항

사용자는 세션을
적절한 인원으로
예약할 수 있다

01
대상
등록 사용자

03
입력
정수 1~4석

05
결과
RESERVED +

받습니다.

02

상태

세션 OPEN

04

조건

잔여석 ≥ 요청석

06

불변

오류 시 차감 0

ㅏ ID

4.1 여섯 원자

원자	질문	합성 예약 예
대상 WHO	누가·무엇이 행동하는가	등록 사용자·한 세션
상태 WHEN	어떤 시작 상태·시간인가	session=OPEN
입력 VALUE	값·형식·단위·범위는	integer seats 1~4
조건 ORDER	어떤 조건과 우선순위인가	세션 상태를 정원보다 먼저 판정
결과 OBSERVE	무엇을 외부에서 보는가	code·state·remaining·count
불변 INVARIANT	실패 뒤 무엇이 그대로인가	capacity delta 0·예약 0

4.2 한 문장에 여러 의무가 있으면 나눕니다

다음 요구사항은 예약·알림·결제 세 의무를 섞습니다.

```
예약이 성공하면 결제를 처리하고 알림을 빠르게 전송한다.
```

원자화한 후보:

REQ	한 의무	별도 oracle
REQ-A	예약 성공 시 예약 상태를 RESERVED로 만든다	state·reservation ID
REQ-B	RESERVED 예약의 결제 성공 시 CONFIRMED로 전이한다	payment code·state
REQ-C	CONFIRMED 뒤 정해진 시간 안에 알림 event를 기록한다	event·timestamp

원자화는 요구를 쪼개기 위한 형식 놀이가 아닙니다. 서로 다른 실패 원인·owner·검증 방법을 분리하는 작업입니다.

4.3 비기능 요구사항도 관찰 가능한 값으로 바꿉니다

모호한 표현	판정 가능한 후보
빠르게 응답한다	승인된 환경에서 p95 ≤ [값] ms
안전하게 저장한다	[암호화 범위]·[key 관리]·[접근 role] 충족

모호한 표현	판정 가능한 후보
사용하기 쉽다	대표 과업 성공률·오류율·시간 threshold
충분히 기록한다	필수 audit fields·보존 기간·누락률

값을 임의로 채우지 않습니다. 제품·보안·운영 owner가 threshold와 측정 방법을 승인해야 합니다.

5. 모호한 표현을 다섯 질문으로 거릅니다

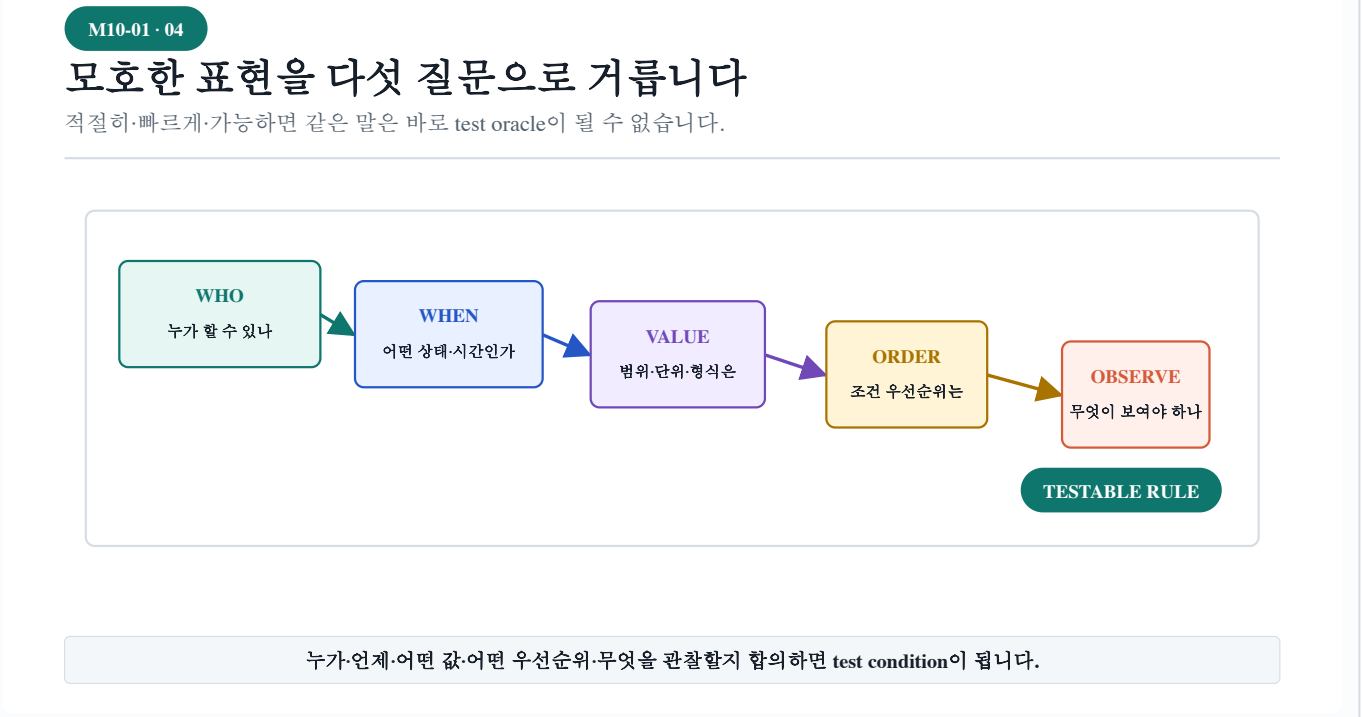
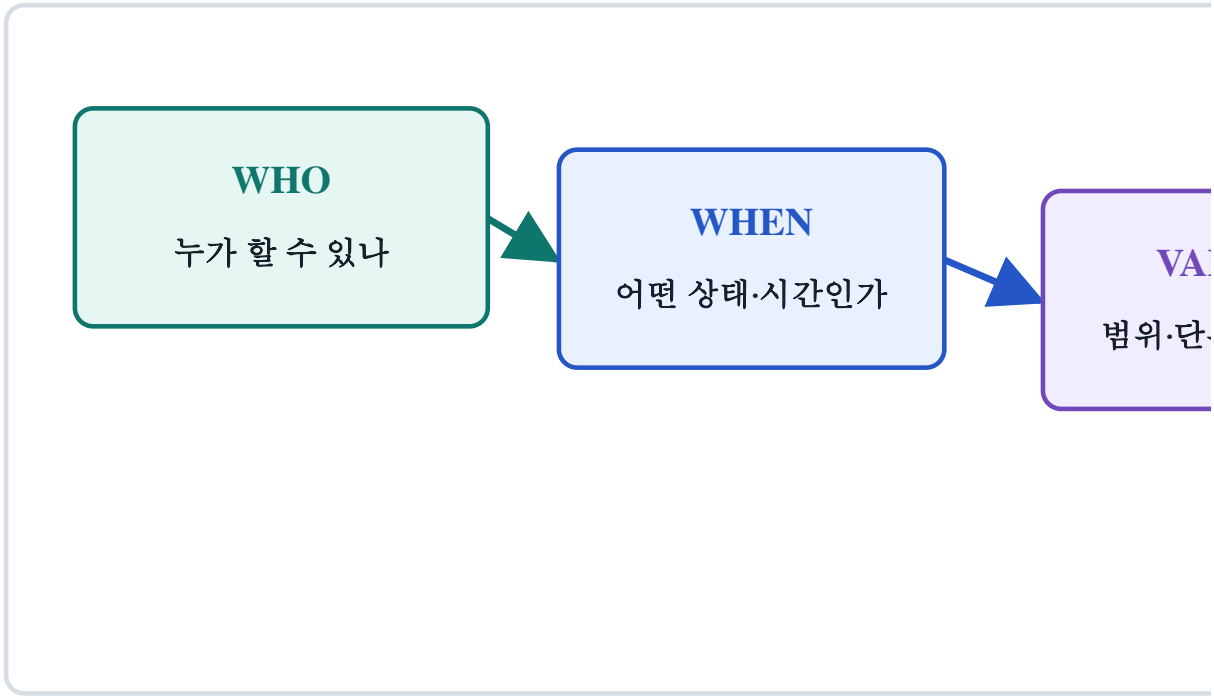


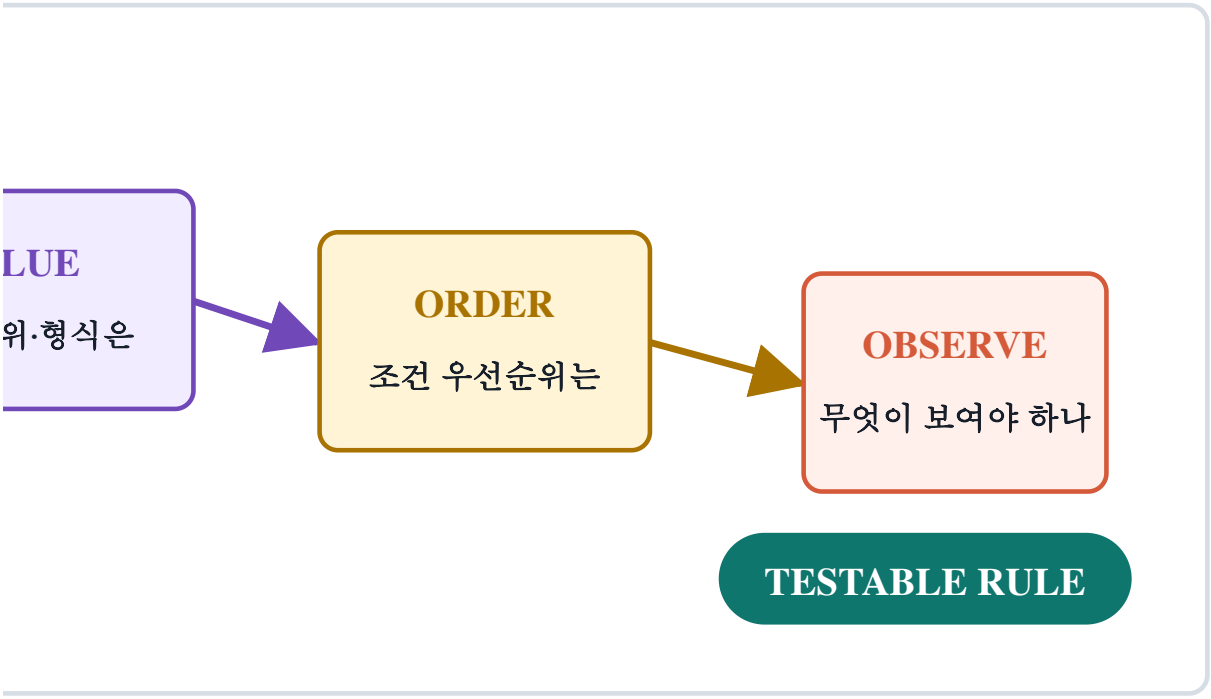
그림 4. “적절히”라는 단어를 기대값으로 쓰지 말고 누가·언제·어떤 값·어떤 우선순위·무엇을 관찰할지 합의합니다.

[illegible]

적절히·빠르게·가능하면 같은 말은 바로 test oracle이 될 수 없습



·니다.



5.1 모호성은 테스트 작성자가 조용히 해석하지 않습니다

테스터가 빈칸을 임의로 메우면 테스트는 숨은 요구사항이 됩니다. 실패 뒤 “원래 그 뜻이 아니었다”는 논쟁이 생깁니다. 질문·결정·결정자·날짜·영향 케이스를 남겨야 합니다.

모호한 표현	질문	잘못된 조용한 가정
적절한 인원	최소·최대·정수·0·문자열은	2명만 시험
빠르게	어느 환경의 어떤 percentile인가	내 노트북 체감
오류를 알린다	code·message·field·상태는	빨간 문구면 통과
먼저 처리한다	동시에 맞는 조건의 우선순위는	구현 순서를 기준으로 삼음
중복을 막는다	동일성 key·payload·시간 범위는	버튼 두 번 클릭만 시험

5.2 좋은 질문은 선택지를 좁힙니다

```
나쁜 질문: 이 요구사항이 맞나요?

좋은 질문: seats=0과 seats=5는 모두 SEAT_COUNT_INVALID인가,
그리고 두 경우 모두 remaining과 reservation_count가 그대로인가?
```

좋은 질문은 대상 값, 기대 결과, 부작용을 함께 제시합니다. 답변이 곧 acceptance criterion과 oracle이 됩니다.

5.3 질문이 해결될 때까지 case 상태를 구분합니다

상태	의미	실행 가능 여부
draft	분석 중이며 expected가 바뀔 수 있음	참고 실행만
blocked-by-requirement	owner 결정 없이는 판정 불가	release evidence 제외
reviewed	기법·oracle 동료 검토 완료	실행 가능
approved	requirement owner와 기준 합의	gate 사용 가능
retired	요구·기능이 폐기됨	이력만 유지

6. 테스트 케이스 한 줄에 판정 근거를 묶습니다

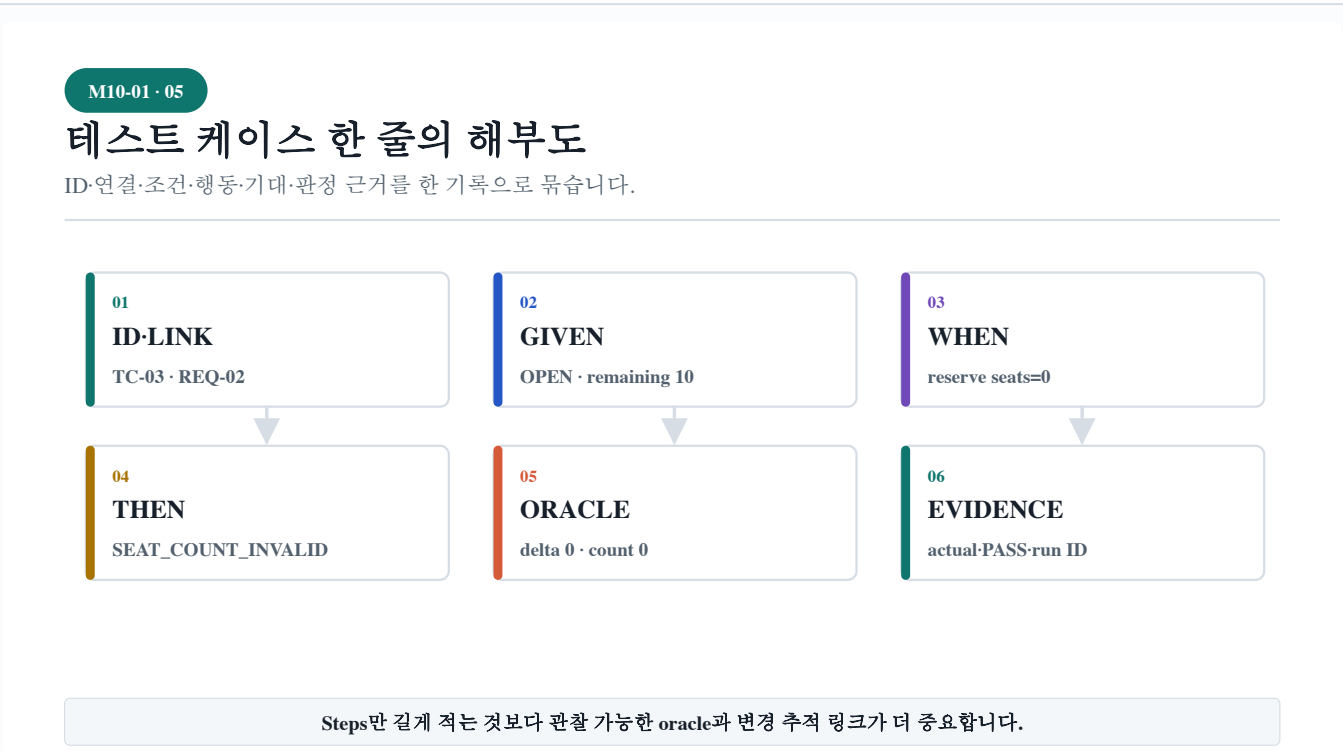


그림 5. Steps 길이보다 ID·연결·시작 상태·한 사건·관찰 결과·판정 증거가 완전한지가 중요합니다.

ID·연결·조건·행동·기대·판정 근거를 한 기록으로 묶습니다.

01

ID·LINK

TC-03 · REQ-02

02

GIVEN

OPEN · remaining

04

THEN

SEAT_COUNT_INVALID

05

ORACLE

delta 0 · count 0

10

03

WHEN

reserve seats=0

06

EVIDENCE

actual·PASS·run ID

6.1 최소 case schema

field	질문	나쁜 예	좋은 예
ID·link	왜 검사하나	예약 테스트	TC-03 · REQ-02
precondition	어디서 시작하나	화면을 연다	OPEN·remaining 10·count 0
test data	어떤 값을 쓰나	잘못된 값	seats=0 · fixture v1
action	어떤 사건인가	여러 버튼 클릭	reserve 한 번
expected	무엇이 보여야 하나	오류가 난다	code·state·remaining·delta·count
oracle	왜 그것이 맞나	상식	REQ-02 v1.0 acceptance
evidence	무엇으로 다시 보나	기억	run ID·actual fields·trace

6.2 제목은 조건과 기대를 드러냅니다

약한 제목	강한 제목
예약 오류 테스트	0석 요청은 저장 없이 SEAT_COUNT_INVALID
결제 테스트	결제 실패 시 RESERVED 상태 유지
취소 확인	CANCELLED 상태의 재취소는 INVALID_TRANSITION

제목만 훑어도 coverage map이 보여야 합니다. “테스트 1”, “오류 확인” 같은 이름은 변경 영향과 중복 탐지를 어렵게 합니다.

6.3 행동은 한 가지 핵심 사건으로 제한합니다

한 케이스 안에서 등록·로그인·예약·결제·취소를 모두 하면 어느 단계가 원인인지 알기 어렵습니다. 필수 setup은 fixture·API·Given으로 준비하고, When은 판정하려는 한 사건으로 둡니다.

6.4 Expected는 화면 문자열보다 넓습니다

결과 층	예약 실패 예
사용자-visible	오류 code·field message
domain state	예약 상태 생성 안 됨

결과 층	예약 실패 예
quantity	remaining 변화 0
persistence	reservation count 0
integration side effect	결제·알림 event 0
audit	실패 이유와 run 식별자 기록

7. 동등분할로 모든 값 대신 처리 방식을 봅니다

그림 6. 모든 값을 다 넣는 대신 같은 결과가 예상되는 집합을 만들고, valid와 invalid partition마다 대표값을 고릅니다.

같은 결과가 예상되는 값의 집합을 동등분할로 만듭니다.

같게 처리될 값의 집합마다 ㅊ

유효 1~4	4 초과
대표값 2	대표값 5

최소 하나의 대표를 고릅니다

7.1 동등분할의 세 계약

1. Partition은 서로 겹치지 않습니다.
2. Partition은 비어 있지 않습니다.
3. 같은 partition 안의 값은 같은 처리 결과가 예상됩니다.

좌석 수가 정수 1~4만 허용될 때:

PART	규칙	valid	대표값	expected
P1	비정수	no	`two`	SEAT_COUNT_INVALID
P2	x < 1인 정수	no	0	SEAT_COUNT_INVALID
P3	1 ≤ x ≤ 4인 정수	yes	3	RESERVED
P4	x > 4인 정수	no	5	SEAT_COUNT_INVALID

7.2 “같은 처리”를 결과 code만으로 판단하지 않습니다

두 입력이 모두 오류 code를 내도 한쪽은 잔여석을 차감하고 다른 쪽은 그렇지 않다면 같은 partition이 아닐 수 있습니다. Code·state·quantity·side effect를 함께 봅니다.

7.3 Partition을 너무 크게 합치지 않습니다

다음은 모두 invalid처럼 보이지만 검증 경로가 다를 수 있습니다.

입력	잠재 처리 차이
`null`·field missing	schema required 검사
빈 문자열	parsing·normalization
문자열 `2`	type coercion 정책
실수 2.5	numeric type·integer rule
정수 0	lower range rule
정수 5	upper range rule

처리와 위험이 다르면 별도 partition으로 나눕니다. 반대로 결과와 경로가 같다면 불필요하게 수십 개 케이스로 늘리지 않습니다.

7.4 여러 입력의 partition 조합

입력 field가 늘면 모든 조합이 폭발합니다. 먼저 각 field의 중요한 partition을 찾고, business rule로 상호작용하는 조합은 결정표나 pairwise 같은 후속 기법으로 다룹니다. “모든 조합”을 무작정 약속하지 않습니다.

7.5 동등분할 self-check

- valid·invalid가 모두 있는가.
- 값 없음·형식 오류·범위 오류를 구분했는가.
- 대표값 선택 이유를 적었는가.
- 같은 partition이라는 결과 근거가 있는가.
- 각 partition이 적어도 한 TC와 연결됐는가.

8. 경계값의 안쪽과 바깥쪽을 나란히 찌릅니다

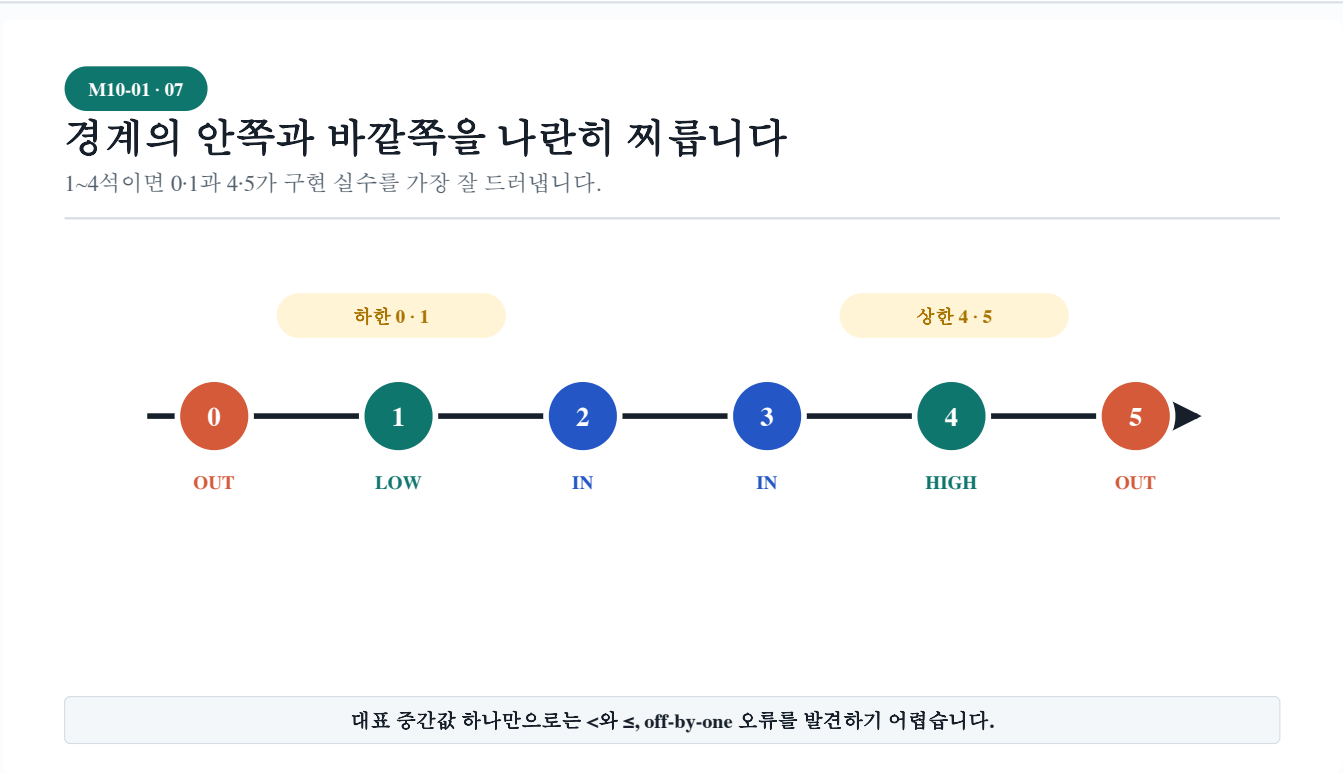
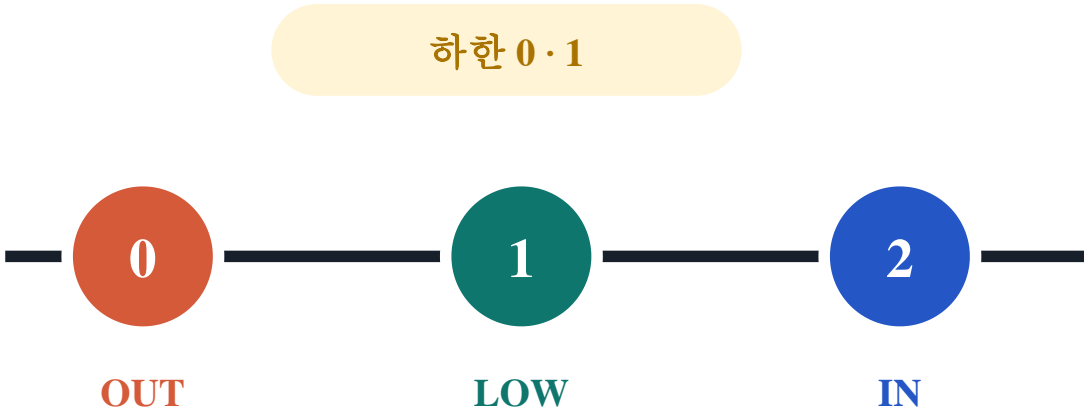


그림 7. 1~4석이면 0·1과 4·5를 나란히 실행해야 <·≤·off-by-one 구현 실수를 드러낼 수 있습니다.

1~4석이면 0·1과 4·5가 구현 실수를 가장 잘 드러냅니다.



상한 4 · 5

8.1 대표 중간값은 경계를 증명하지 않습니다

`seats=2`가 성공해도 구현이 0~5를 허용하는지 알 수 없습니다. 실습의 `boundary-bug-v1`은 정확히 그런 오류를 넣었습니다. TC-02는 통과하지만 TC-03과 TC-06은 실패합니다.

8.2 2-value와 3-value 관점

ISTQB CTFL v4.0.1은 boundary value analysis에서 2-value와 3-value 접근을 설명합니다. 맥락에 따라 경계값과 가장 가까운 이웃을 고르거나, 경계와 양쪽 이웃을 함께 고릅니다.

규칙	기본 probe
1 ≤ seats ≤ 4	0·1·4·5
length < 100	99·100, 위험하면 98·99·100
age ≥ 14	13·14
start ≤ now < end	start 직전·정확히 start·end 직전·정확히 end

8.3 경계는 숫자에만 있지 않습니다

domain	경계 예
문자열	최소·최대 길이, 빈 문자열, Unicode normalization
날짜	월말·윤년·DST·자정·timezone
목록	0개·1개·page size·마지막 page
금액	0·최소 결제·한도·소수점·반올림
권한	만료 직전·정확히 만료·직후
저장 용량	limit 바로 아래·정확히 limit·바로 위

8.4 Inclusive와 exclusive를 문서에 적습니다

`1~4`는 일상어로는 모호할 수 있습니다. 수식과 예를 함께 적습니다.

```
valid: integer x where 1 <= x <= 4
invalid examples: 0, 5, 2.5, "2", null
```

기대 결과도 함께 승인해야 구현과 테스트가 같은 경계를 봅니다.

8.5 경계값 self-check

- lower·upper 양쪽을 식별했는가.
- 경계 포함·제외가 명확한가.
- 바로 안과 바로 밖 값을 선택했는가.
- 수치 type·단위·반올림을 고정했는가.
- 경계 실패 뒤 상태·수량 불변식을 검사하는가.

9. 결정표의 열로 조건 조합을 외부화합니다

그림 8. 머릿속 if 분기를 표의 열로 꺼내면 가능한 조합·우선순위·누락·충돌이 눈에 보입니다.

[illegible]

분기 수가 늘수록 기억에 의존하지 말고 feasible rule을 열거합니

조건·행동	R1
세션 OPEN	T
잔여석 충분	T
RESERVED	X
CAPACITY_INSUFFICIENT	–
SESSION_NOT_OPEN	–

다.

	R2	R3	R4
	T	F	F
	F	T	F
	–	–	–
	X	–	–
	–	X	X

9.1 결정표의 구조

구성	뜻
condition rows	결과를 결정하는 사실·상태·입력
action rows	각 조합에서 일어나야 할 결과
rule columns	한 조건 조합과 그 기대 행동
feasible 표시	실제로 만들 수 있는 조합인지
priority	여러 조건이 맞을 때 먼저 적용할 규칙
TC link	해당 column을 실행하는 케이스

9.2 예약 rule을 열로 읽기

조건·행동	R1	R2	R3	R4
session OPEN	T	T	F	F
remaining 충분	T	F	T	F
RESERVED	X	-	-	-
CAPACITY_INSUFFICIENT	-	X	-	-
SESSION_NOT_OPEN	-	-	X	X

R4는 두 실패 조건이 동시에 맞습니다. 요구사항이 우선순위를 정하지 않으면 구현·테스트마다 다른 오류를 고를 수 있습니다. 결정표가 요구사항 결함을 드러내는 지점입니다.

9.3 각 feasible column은 coverage item입니다

ISTQB guidance에서 실행 가능한 결정표 column은 coverage item으로 볼 수 있습니다. 최소 기준은 각 feasible rule을 하나 이상의 case로 덮는 것입니다. 하지만 위험이 큰 열은 여러 데이터 값이나 추가 oracle이 필요할 수 있습니다.

9.4 Don't care로 줄이되 의미를 숨기지 않습니다

세션이 닫혀 있으면 잔여석 조건과 무관하게 `SESSION_NOT_OPEN`이라면 `remaining 충분`을 `–`로 접을 수 있습니다. 단, 이것은 명시된 우선순위가 있을 때만 가능합니다.

9.5 결정표가 찾는 요구사항 결함

냄새	의미
행동이 빈 column	가능한 조합의 expected 누락
같은 조건에 두 행동	rule 충돌 또는 우선순위 누락
동일한 column 반복	중복 rule
만들 수 없는 조합	domain constraint를 문서화해야 함
너무 많은 column	조건을 단계별 표로 분해할 필요

9.6 결정표 self-check

- 결과를 바꾸는 조건만 넣었는가.
- feasible·infeasible 근거가 있는가.
- 행동 없는 가능한 조합이 없는가.
- 동시에 참인 조건의 우선순위를 정했는가.
- 실행 가능한 열마다 TC가 있는가.

10. 상태 테스트는 화면이 아니라 이동을 봅니다

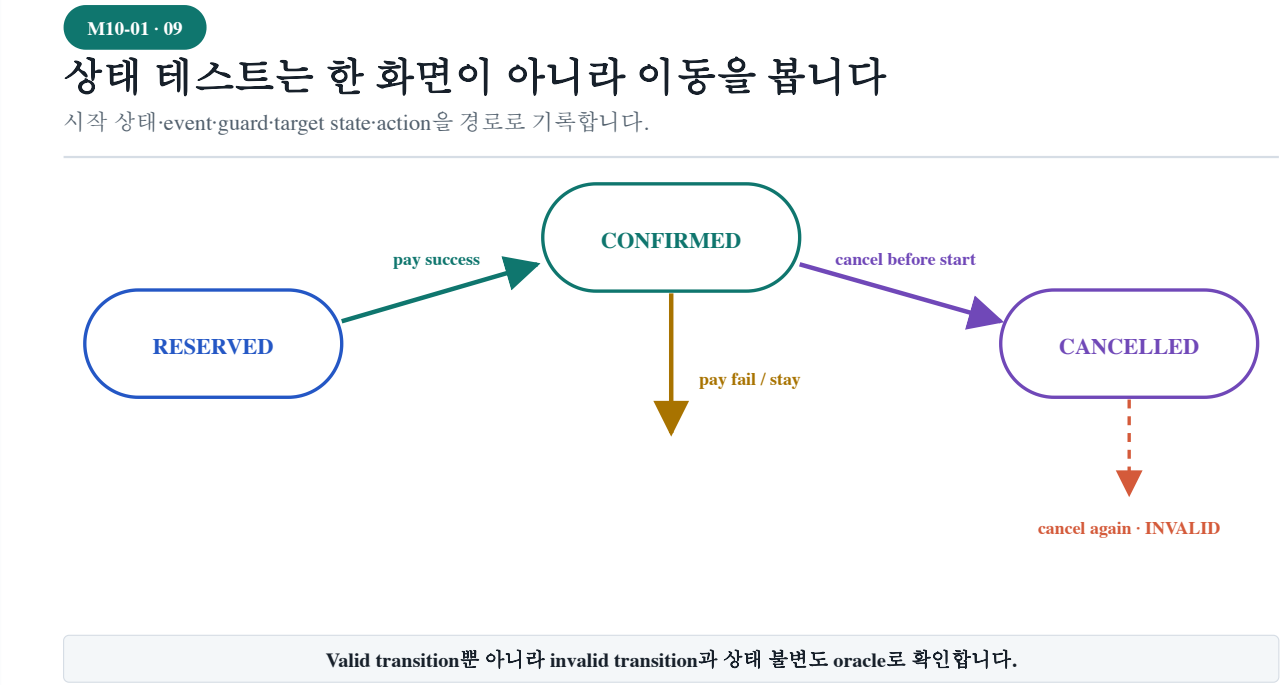
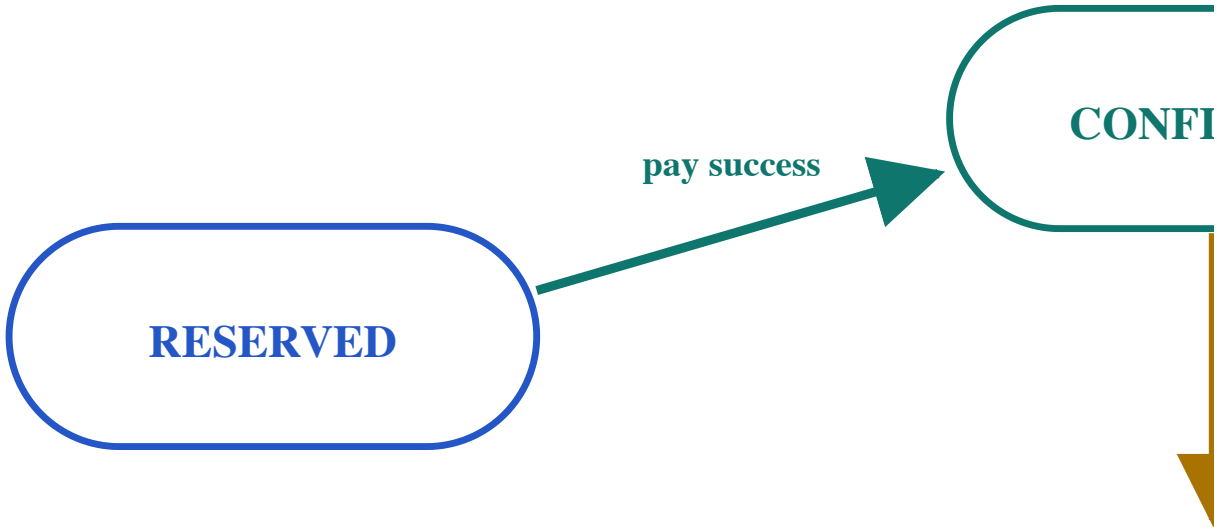
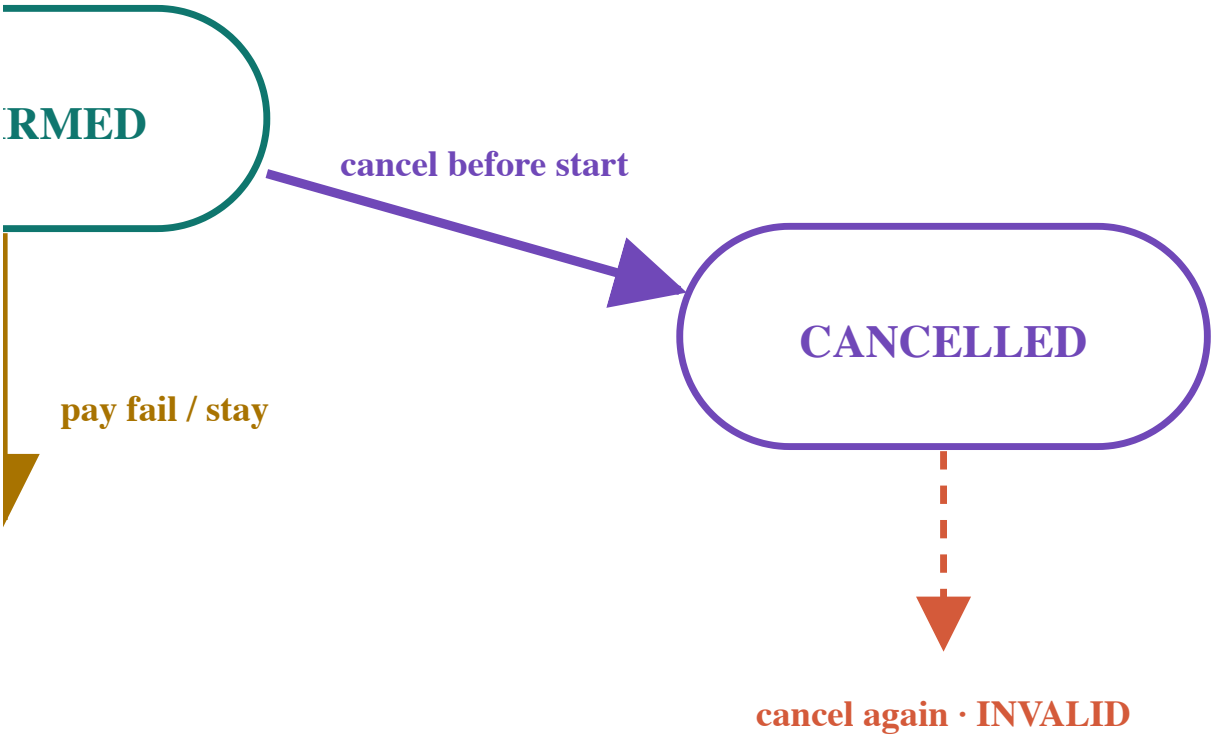


그림 9. 시작 state·event·guard·action·target state를 경로로 적고, valid transition뿐 아니라 invalid transition과 불변식을 확인합니다.

시작 상태·event·guard·target state·action을 경로로 기록합니다.




RMED
cancel before start
CANCELLED
pay fail / stay
cancel again · INVALID

10.1 상태전이의 다섯 요소

요소	예약 예
start state	RESERVED
event	pay(success=true)
guard	결제 가능한 예약
transition action	결제 승인 기록
target state	CONFIRMED

10.2 Valid transition만 보면 부족합니다

시작	event	expected code	목표·유지 state	중요 이유
RESERVED	pay success	PAYMENT_ACCEPTED	CONFIRMED	정상 이동
RESERVED	pay fail	PAYMENT_FAILED	RESERVED	실패 뒤 유지
CONFIRMED	cancel before start	CANCELLED	CANCELLED	허용 이동
CANCELLED	cancel again	INVALID_TRANSITION	CANCELLED	금지 이동·불변식

금지된 event가 성공처럼 보이거나 상태를 바꾸지 않는지 검사해야 합니다. 오류 message만 맞고 state가 틀리면 다음 행동 전체가 오염됩니다.

10.3 State coverage와 transition coverage는 다릅니다

모든 state를 한 번 방문해도 모든 transition을 실행한 것은 아닙니다. 특히 같은 state에 여러 event가 있거나, 한 transition의 guard가 여러 값이면 transition·rule coverage를 따로 봅니다.

10.4 연속 경로가 필요한 경우

단일 transition이 모두 맞아도 조합에서 결함이 날 수 있습니다.

```
RESERVED → CONFIRMED → CANCELLED
RESERVED → payment fail → RESERVED → payment success → CONFIRMED
```

중요한 lifecycle은 transition pair와 end-to-end path를 추가합니다. 모든 경로를 무한히 만들지 말고 위험·빈도·과거 결함으로 고릅니다.

10.5 상태전이 self-check

- initial·final state가 있는가.
- event와 guard를 구분했는가.
- valid·invalid transition을 모두 식별했는가.
- 실패 뒤 state·quantity invariant가 있는가.
- 도달 불가능하거나 빠져나올 수 없는 state가 없는가.

11. Given·When·Then을 짧은 실행 계약으로 씁니다

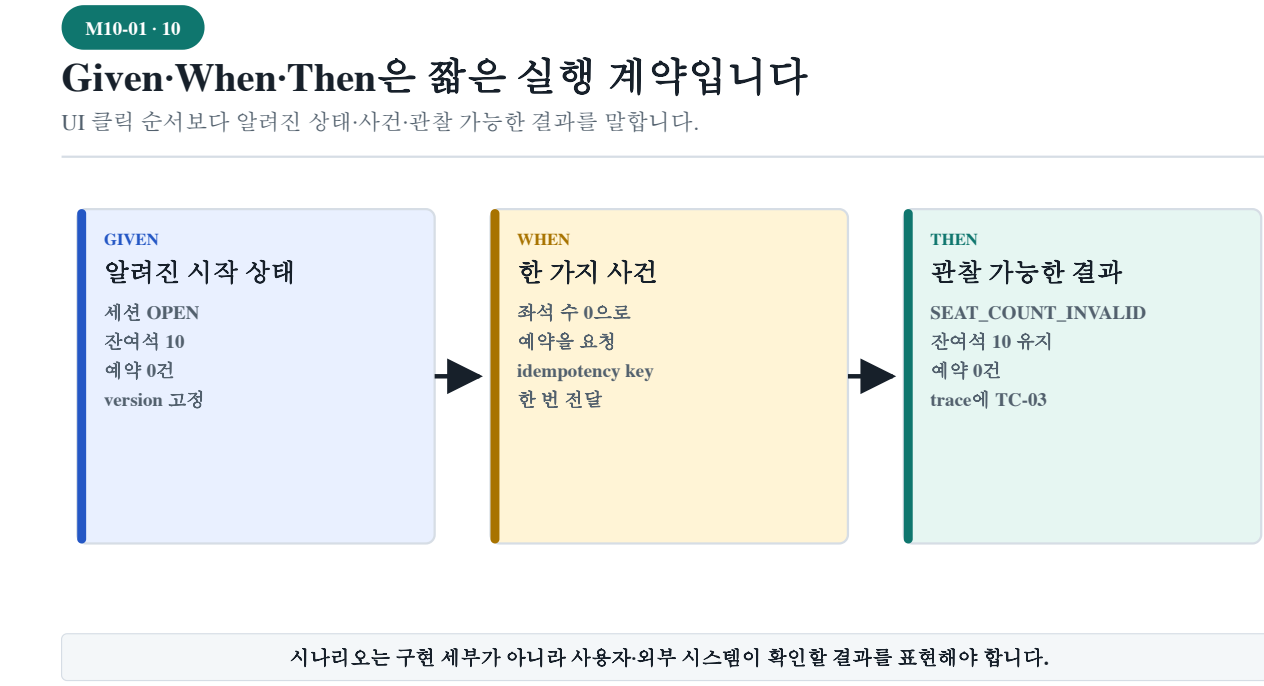


그림 10. UI 클릭 순서보다 알려진 상태·한 사건·사용자와 외부 시스템이 볼 수 있는 결과를 표현합니다.

Given-When-Then은 테스트 문장이 아니라 상태 계

UI 클릭 순서보다 알려진 상태·사건·관찰 가능한 결과를 말합니

GIVEN

알려진 시작 상태

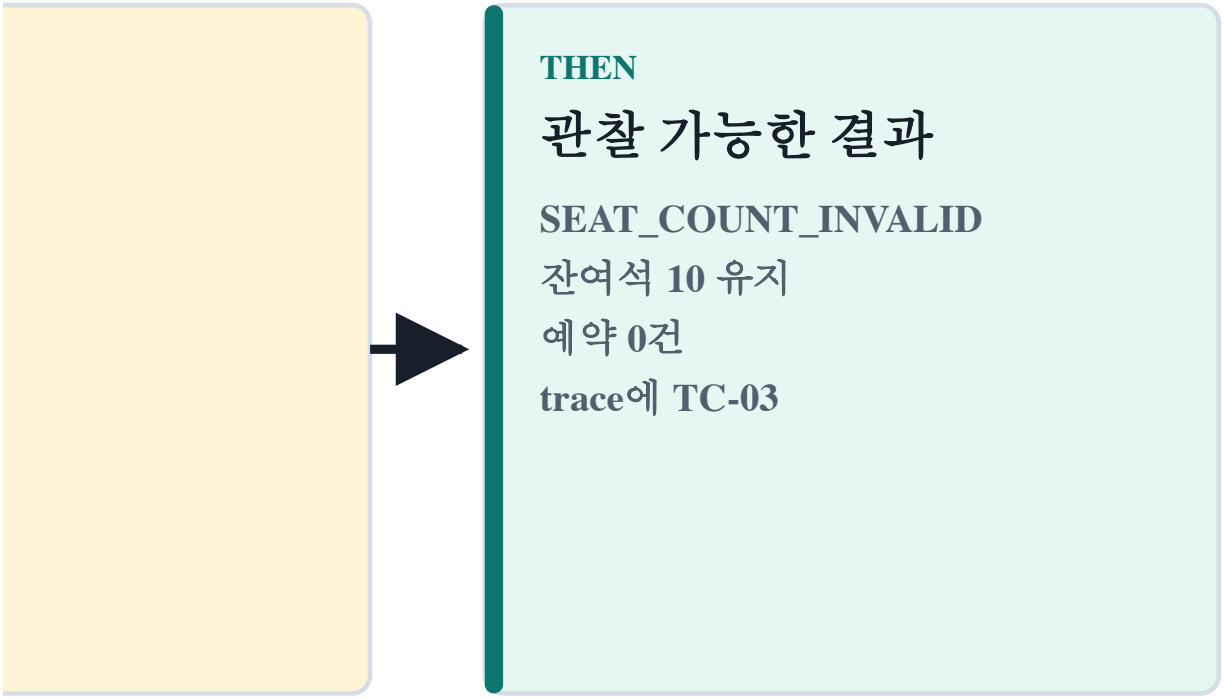
세션 OPEN
잔여석 10
예약 0건
version 고정

WHEN

한 가지 사건

좌석 수 0으로
예약을 요청
idempotency key
한 번 전달

다.



11.1 세 절의 역할

keyword	역할	피해야 할 것
Given	알려진 초기 상태·data·permission·version	실행 순서에 의존하는 긴 setup
When	결과를 일으키는 한 사건	여러 업무 행동을 한 줄에 섞기
Then	관찰 가능한 결과와 불변식	내부 함수 이름·구현 추측

Cucumber Gherkin Reference는 Given을 알려진 초기 context, When을 event/action, Then을 관찰 가능한 outcome을 표현하는 데 사용합니다. Scenario는 business rule의 구체 예이며, 구현 세부보다 행동을 설명해야 오래 유지됩니다.

11.2 예약 경계 실패 예

```
Given session=OPEN, remaining=10, reservation_count=0
And requirements_version=session-booking-requirements-1.0.0
When the user requests seats=0 once
Then code=SEAT_COUNT_INVALID
And state=NONE
And remaining=10, capacity_delta=0, reservation_count=0
```

11.3 Scenario Outline은 표를 읽을 때 씁니다

```
Scenario Outline: 좌석 수 경계 판정
  Given session=OPEN and remaining=10
  When seats=<seats>로 예약한다
  Then code=<code> and remaining=<remaining>

  Examples:
    | seats | code               | remaining |
    | 0     | SEAT_COUNT_INVALID | 10        |
    | 1     | RESERVED           | 9         |
    | 4     | RESERVED           | 6         |
    | 5     | SEAT_COUNT_INVALID | 10        |
```

Outline이 coverage 설계를 대신하지 않습니다. 먼저 BVA로 0·1·4·5를 선택한 뒤 표현 형식으로 Outline을 씁니다.

11.4 좋은 Then은 관찰 가능하고 구체적입니다

약한 Then	강한 Then
정상적으로 처리된다	code=RESERVED·state=RESERVED
오류를 보여 준다	code=SEAT_COUNT_INVALID·field=seats
데이터가 그대로다	remaining=10·count=0·event count=0
결제가 실패한다	code=PAYMENT_FAILED·state=RESERVED

11.5 Scenario independence

각 scenario는 다른 scenario의 실행 결과에 의존하지 않아야 합니다. 공통 Given은 fixture로 준비하고, case마다 reset·cleanup을 정의합니다. 순서 의존성은 flaky test의 주요 원인입니다.

12. Test data·환경·설정을 case의 일부로 둡니다

같은 case라도 build·feature flag·timezone·seed가 다르면 actual이 달라질 수 있습니다. 재현성은 step을 자세히 쓰는 것보다 실행 조건을 version으로 고정하는 데서 나옵니다.

12.1 최소 environment manifest

구성요소	기록 예
product	build·commit·container image
runtime	OS·Python·browser·device
database	schema·migration·fixture version
configuration	feature flags·policy·environment variables
dependencies	external service stub·API version
time	timezone·clock fixture·expiry 기준
randomness	random seed·generated data version

12.2 실제 데이터보다 목적에 맞는 합성 데이터부터

실습은 실제 고객 데이터가 필요하지 않습니다. 경계·조합·상태를 드러내는 최소 합성 fixture가 더 안전하고 재현 가능합니다. 실제 운영 분포를 검증할 때는 승인·비식별화·최소화·보존·삭제·접근 통제를 별도로 설계합니다.

12.3 Test data도 ID와 owner가 필요합니다

field	이유
DATA ID·version	같은 fixture를 다시 찾기
purpose·related TC	왜 필요한 값인지 설명
source·synthetic status	개인정보·권한 경계 확인
create·reset procedure	독립 실행 보장
expected state	setup 오류와 제품 오류 구분
owner·retention	유지·삭제 책임

12.4 외부 의존성 대역을 구분합니다

대역	용도	주의
stub	정해진 응답 반환	호출 검증은 제한적
mock	예상 호출·인수·횟수 검증	구현 세부에 과결합 가능
fake	단순하지만 동작하는 구현	실제 시스템과 차이 기록
simulator	복잡한 protocol·상태 모사	정확성·version 유지 비용

대역 통과는 실제 integration 통과가 아닙니다. 별도 integration contract와 실제 환경 검증 단계를 둡니다.

13. Oracle은 “정상 화면”보다 정확한 비교 기준입니다

그림 11. 표면 code가 같아도 remaining·count가 다르면 실패입니다. Expected와 actual을 field 단위로 비교합니다.

결과 code뿐 아니라 state·수량·부작용·시간을 함께 비교합니다.

EXPECTED

예상 결과

code=RESERVED

state=RESERVED

remaining=8

count=1

→

ACTUAL

실제 결과

code=RESERVE

state=RESERVE

remaining=6

count=2

D
D

ORACLE

표면 code는 같지만

capacity·count 불일치

TC-15 FAIL

중복 부작용 발견

13.1 Oracle의 역할

Test oracle은 실제 결과가 올바른지 판정하는 기준입니다. 요구사항·business rule·승인된 reference·불변식·독립 계산·metamorphic relation 등이 oracle이 될 수 있습니다.

oracle source	적합한 상황	위험
requirement·acceptance	명시된 규칙	요구 자체가 모호·잘못될 수 있음
approved reference	계산·응답 기준	reference version 노후화
independent implementation	복잡 계산 교차검증	같은 결함을 공유할 수 있음
invariant	상태·수량 보존	세부 결과 전체는 판정 못 함
domain expert	해석이 필요한 판단	합의·가용성·편향
production baseline	변경 전 회귀 비교	기존 결함을 정답으로 고정할 위험

13.2 Code 하나만 같으면 통과가 아닙니다

멱등성 TC-15의 expected와 잘못된 actual:

field	expected	actual	verdict
code	RESERVED	RESERVED	같음
state	RESERVED	RESERVED	같음
remaining	8	6	실패
reservation count	1	2	실패
reservation ID	same	different	실패

사용자는 같은 성공 message를 볼 수 있지만 중복 청구·중복 좌석 차감이 발생합니다. Oracle은 표면 message 뒤의 business state를 봐야 합니다.

13.3 Expected를 실행 전에 고정합니다

실행 뒤 actual을 보고 expected를 맞춰 쓰면 테스트가 현재 구현을 승인하는 기록으로 변합니다. Requirement owner와 oracle source를 먼저 고정하고, 변경이 필요하면 case version과 승인 이력을 남깁니다.

13.4 Tolerance를 명시합니다

대상	모호한 기대	판정 가능한 기대
시간	약 2초	p95 ≤ 2.0s, 측정 구간·환경 명시
실수	거의 같음	absolute error ≤ 0.001
순서	비슷한 순서	stable sort key·tie rule
텍스트	같은 뜻	허용 claim·금지 claim·review protocol

Tolerance도 요구사항입니다. 테스트 작성자가 결과를 본 뒤 편의대로 넓히지 않습니다.

13.5 Oracle problem과 대안

AI 생성 응답·대규모 계산·복잡 simulation처럼 정확한 정답 하나를 알기 어려울 수 있습니다. M09-05의 rubric·grader·사람 calibration을 사용하거나 다음 대안을 조합합니다.

- 입력 변환 전후에 유지돼야 할 metamorphic relation.
- 안전·형식·범위처럼 결정적으로 검사 가능한 invariant.
- 작은 사례의 독립 계산·전문가 gold set.
- 기준 version과 후보의 pairwise review.
- 불확실할 때 `inconclusive`와 사람 검토.

14. 요구사항·케이스·결과·결함을 양방향으로 잇습니다

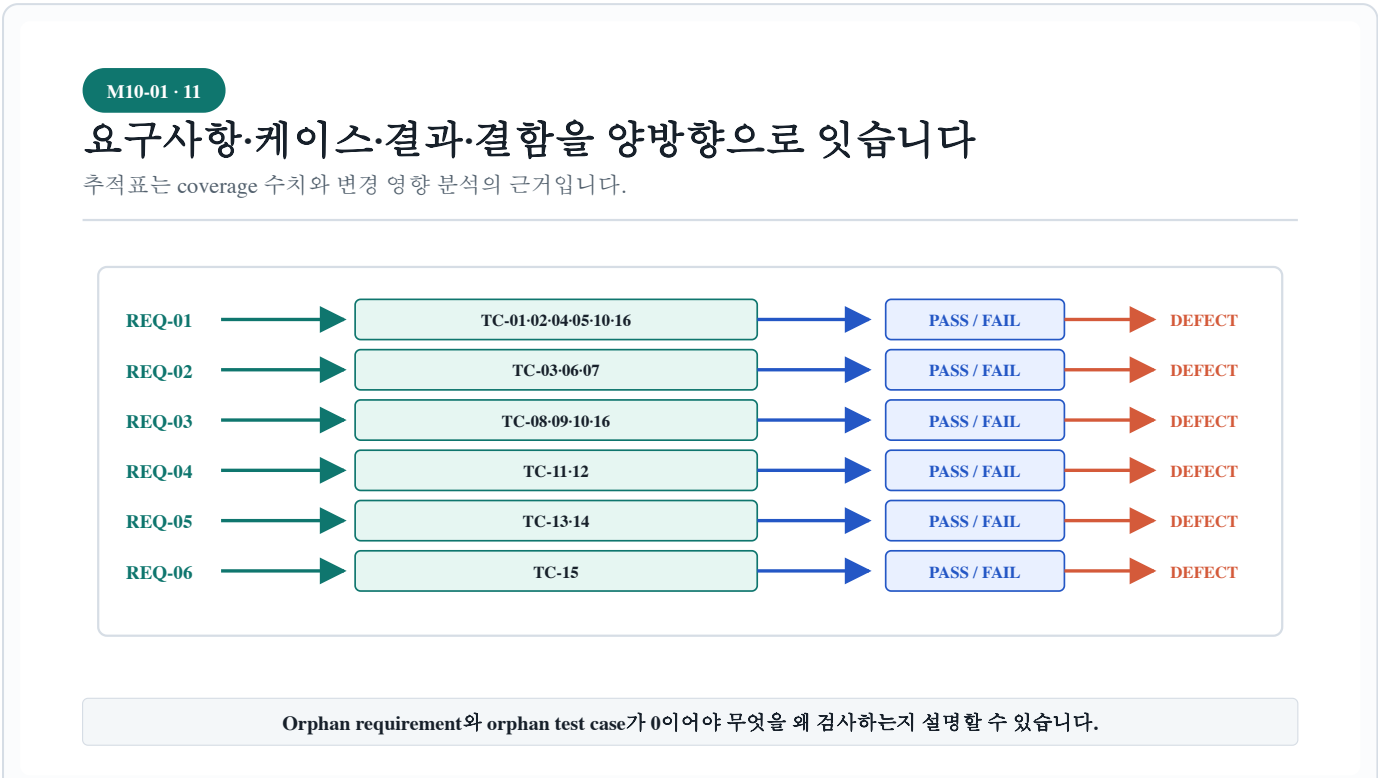
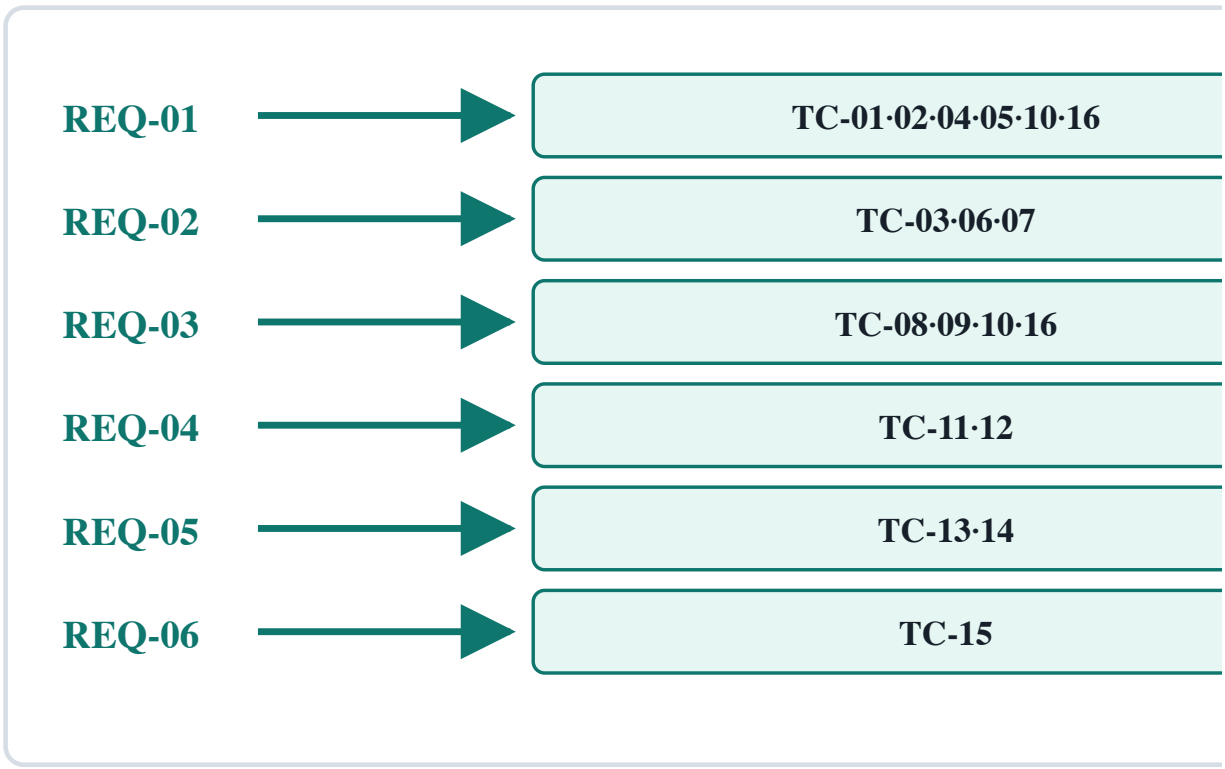
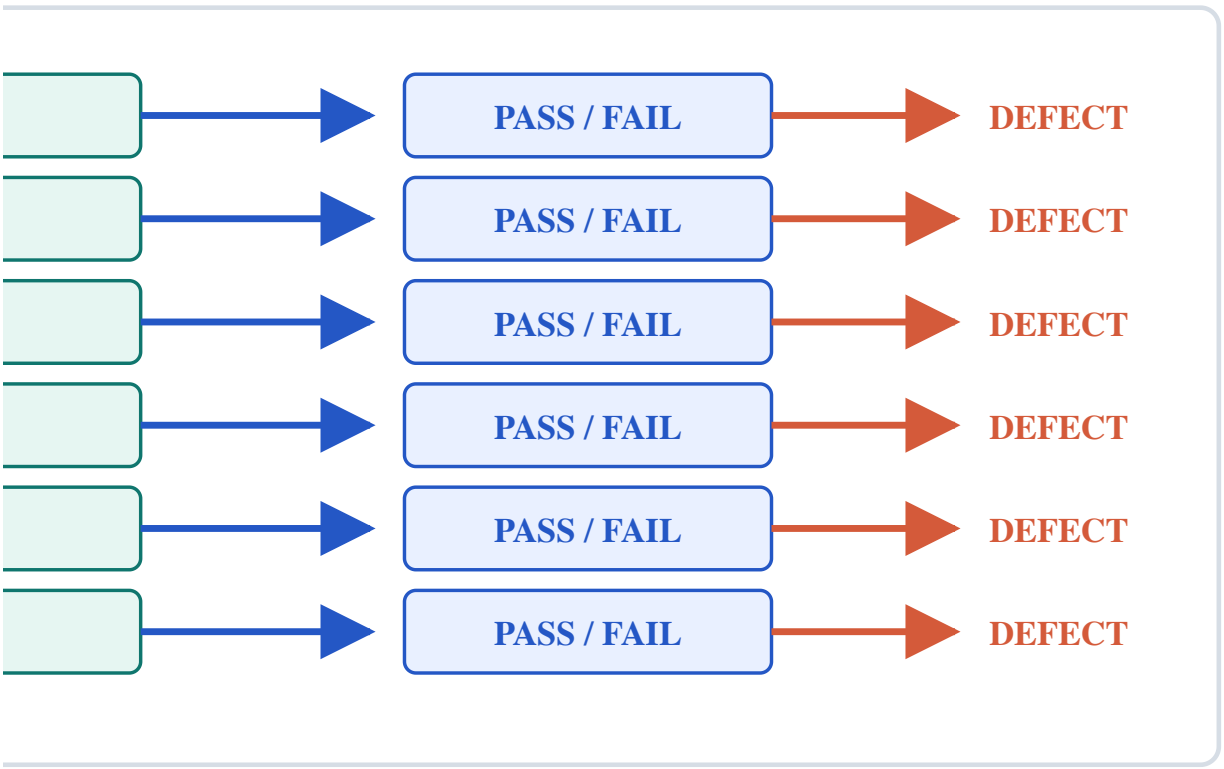


그림 12. RTM은 coverage 퍼센트를 꾸미는 표가 아니라 변경 영향과 실패 이유를 설명하는 지도입니다.

[illegible]

추적표는 coverage 수치와 변경 영향 분석의 근거입니다.




PASS / FAIL
DEFECT
PASS / FAIL
DEFECT
PASS / FAIL
DEFECT
PASS / FAIL
DEFECT
PASS / FAIL
DEFECT
PASS / FAIL
DEFECT

NASA Systems Engineering Handbook은 요구사항 metadata에 고유 ID·rationale·출처·owner·verification method와 관련 책임을 두고, 상하위 요구와 검증 산출물 사이의 양방향 추적을 강조합니다. 소프트웨어에서도 요구사항·설계·코드·테스트·결과 링크가 변경 영향과 완전성 판단의 근거가 됩니다.

14.1 Forward와 backward 질문

방향	질문
forward	REQ-02를 어떤 conditions·cases가 검증하는가
forward	REQ-05가 바뀌면 어떤 state cases를 다시 실행하는가
backward	TC-15는 어떤 business rule 때문에 존재하는가
backward	이 defect를 고치면 어떤 requirement와 regression을 보호하는가

14.2 RTM의 최소 열

REQ	AC·COND	risk	technique·coverage	TC	latest result	defect·owner
REQ-02	invalid seats·no side effect	high	EP·BVA	TC-03·06·07	run-…	DEF-…
REQ-05	cancel valid·invalid	high	state transition	TC-13·14	run-…	booking-policy

14.3 Coverage 수치는 분모와 함께 씁니다

```
requirement coverage = linked requirements / in-scope requirements
condition coverage   = executed conditions / identified conditions
case execution       = executed cases / planned cases
rule coverage        = executed feasible rules / feasible rules
transition coverage  = executed critical transitions / identified critical transitions
```

`100%`만 쓰면 무엇을 100%로 정의했는지 알 수 없습니다. 분자·분모·범위·version을 함께 기록합니다.

14.4 Orphan은 두 방향에서 찾습니다

냄새	질문	가능 조치
orphan requirement	왜 연결된 case가 없는가	condition·case 추가 또는 범위 제외 승인
orphan test case	왜 이 case를 실행하는가	REQ·risk 링크 추가 또는 폐기

냄새	질문	가능 조치
duplicate case	같은 coverage item을 왜 반복하는가	위험 근거 추가 또는 통합
suspect link	REQ 변경 뒤 TC가 아직 유효한가	review·redesign·rerun

14.5 Trace link의 정확성이 coverage보다 먼저입니다

잘못 연결된 100%는 0%보다 위험할 수 있습니다. 링크가 실제 규칙·조건을 검증하는지 review합니다. 단지 같은 기능 이름이 있다는 이유로 연결하지 않습니다.

15. Risk로 설계·실행·자동화 순서를 정합니다

모든 케이스를 같은 깊이와 빈도로 실행할 수 없습니다. 실패 가능성·영향·사용 빈도·변경량·과거 결함을 함께 봅니다.

15.1 간단한 우선순위 모델

```
risk score = likelihood × impact
```

이 숫자는 토론을 대신하지 않습니다. 데이터 손실·보안·안전·법적 실패처럼 한 번의 영향이 큰 것은 별도 critical gate로 둡니다.

priority	예	실행 규칙
P0	권한 우회·중복 결제·데이터 손실	모든 후보·100% pass
P1	주 예약·경계·핵심 상태전이	merge·release 전
P2	희귀 표현·낮은 영향 조합	정기·시간 허용 시

15.2 테스트 기법도 위험에 따라 깊이를 바꿉니다

낮은 위험	높은 위험
partition 대표값 1개	여러 대표값·format·null 추가
2-value BVA	3-value·overflow·시간 경계 추가

낮은 위험	높은 위험
rule coverage	우선순위·연속 rule·오류 부작용 추가
transition coverage	invalid·pair·복구 경로 추가

15.3 실행 순서는 피드백 시간을 줄입니다

```
smoke·P0 → 변경 직접 영향 → high-risk boundary·rule
→ broader regression → 느린 end-to-end·환경 검증
```

초기에 빠르고 진단 가능한 실패를 찾으면 긴 suite 결과를 기다리기 전에 수정할 수 있습니다.

16. 실행 결과는 actual·verdict·evidence로 남깁니다

16.1 Run header가 먼저입니다

field	이유
run ID·operator·time	실행 사건 식별
requirement baseline	기대의 기준
test set version	어떤 case를 실행했는가
product build	무엇을 검증했는가
environment·flags	행동 차이 재현
data fixture·seed	시작 상태 재현

16.2 Verdict를 구분합니다

verdict	의미	다음 행동
PASS	모든 필수 oracle 충족	trace·evidence 보관
FAIL	하나 이상의 expected 불일치	진단·defect 후보

verdict	의미	다음 행동
BLOCKED	환경·data·dependency로 실행 불가	blocker 해결·재실행
NOT RUN	아직 실행하지 않음	coverage에서 분리
INCONCLUSIVE	oracle·증거 부족	requirement·review 보완

`BLOCKED`를 `FAIL`로 세면 제품 품질을 왜곡하고, `NOT RUN`을 `PASS`처럼 분모에서 빼면 거짓 coverage가 됩니다.

16.3 Evidence는 판정을 다시 계산할 수 있어야 합니다

evidence	최소 포함
structured result	TC·run·expected·actual·verdict
log	event ID·error code·필수 state, 민감정보 제거
screenshot	필요한 화면·시간·build, 개인 정보 마스킹
DB·state diff	before·after·query·schema version
request·response	승인된 field만, secret·token 제거

Screenshot 한 장만으로 숨은 state·quantity를 증명하기 어렵습니다. 반대로 로그만으로 사용자-visible 오류를 증명하기 어렵습니다. Claim에 맞는 evidence를 조합합니다.

16.4 Evidence 보안

실제 계정·token·결제·고객 원문을 무작정 캡처하지 않습니다. 데이터 최소화·masking·접근 통제·보존 기간·삭제 절차를 정합니다. Evidence도 보호해야 할 데이터입니다.

17. 실패한 테스트가 곧 제품 결함은 아닙니다

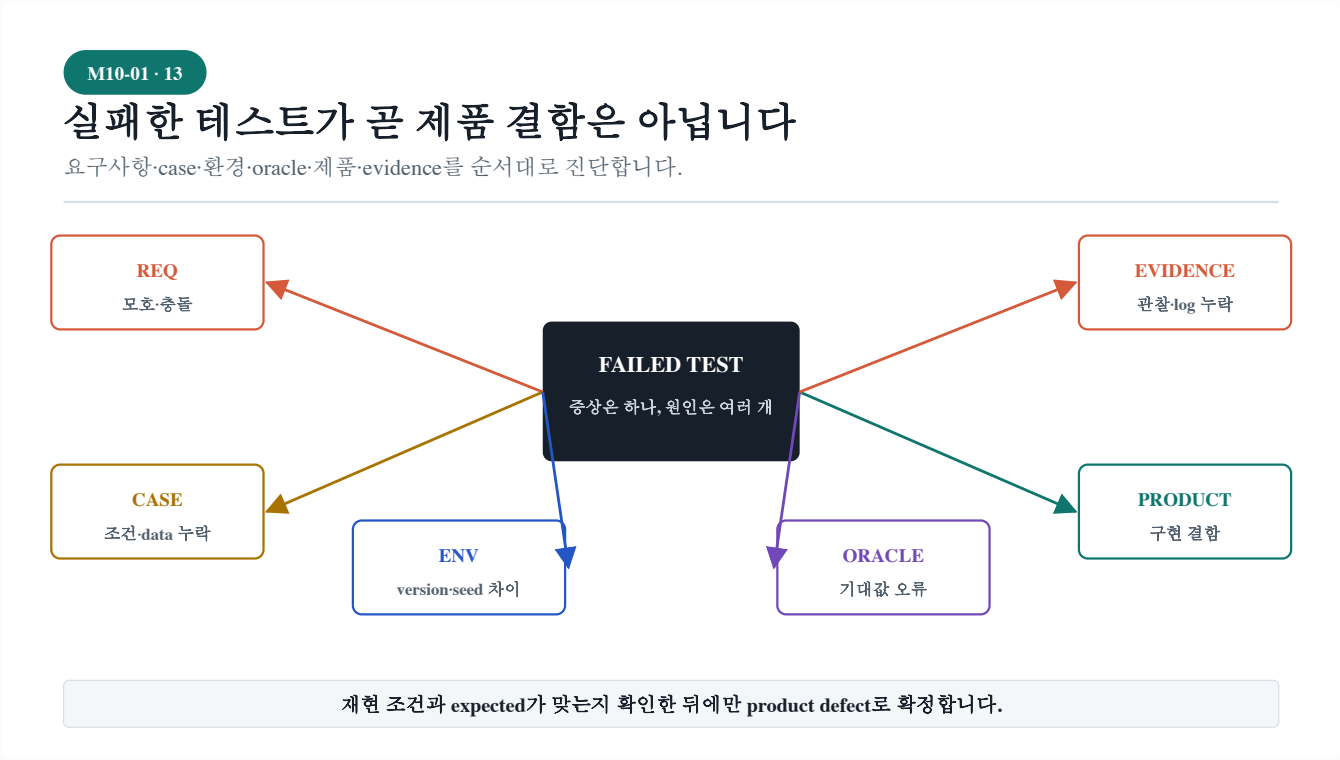
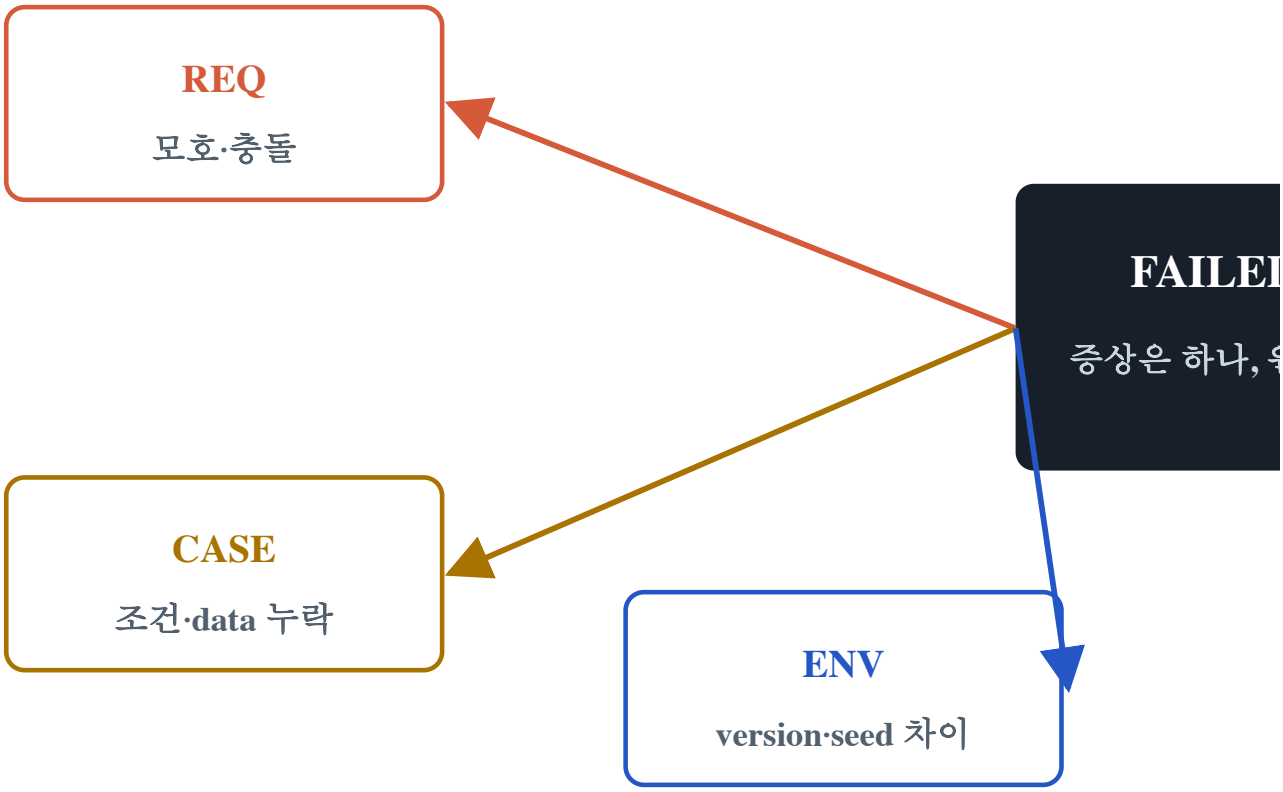
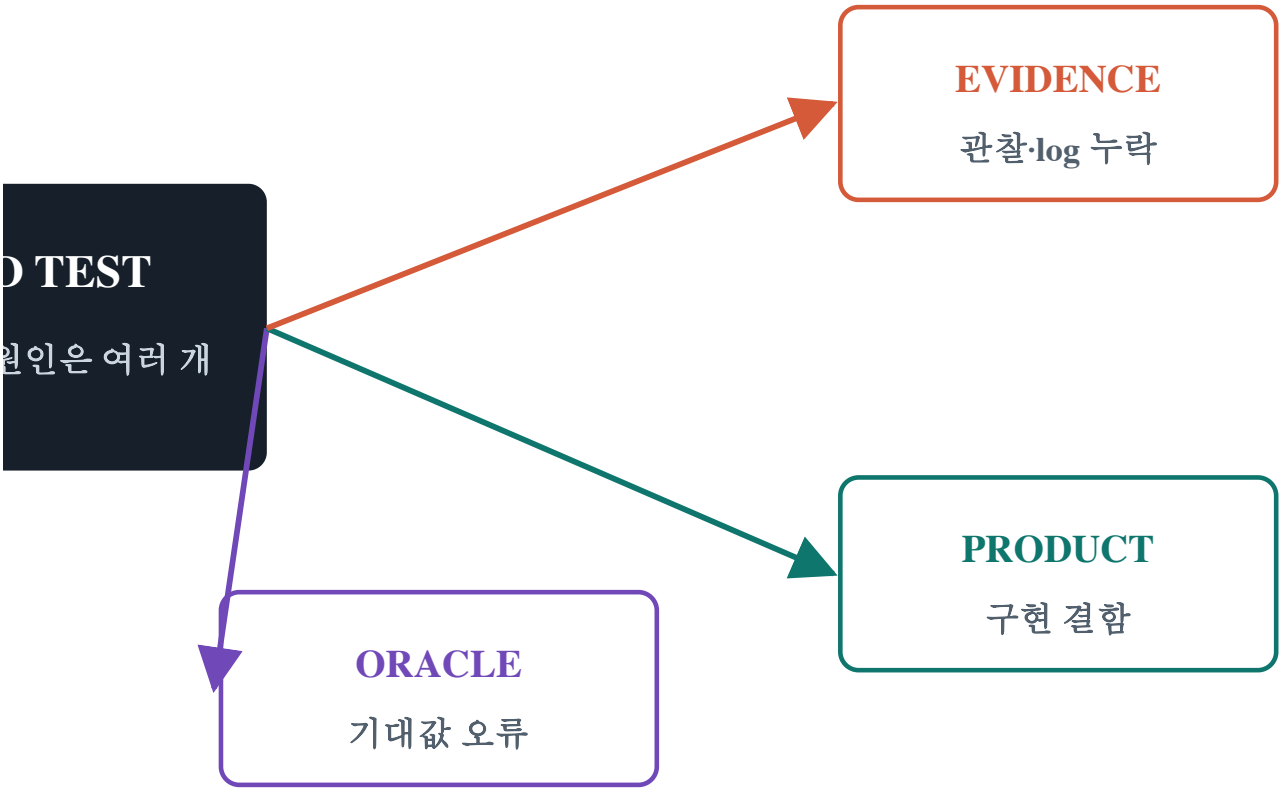


그림 13. 같은 FAIL 증상은 요구사항·케이스·환경·oracle·evidence·제품 중 여러 원인에서 생길 수 있습니다.

요구사항·case·환경·oracle·제품·evidence를 순서대로 진단합니다




EVIDENCE
관찰·log 누락
TEST
원인은 여러 개
PRODUCT
구현 결함
ORACLE
기대값 오류

17.1 여섯 단계 진단 순서

1. **Requirement:** 의미·version·우선순위·승인이 맞는가.
2. **Case:** precondition·data·action이 완전한가.
3. **Environment:** build·flag·dependency·clock·seed가 맞는가.
4. **Oracle:** expected와 tolerance가 승인 기준과 맞는가.
5. **Evidence:** actual을 충분히 관찰했는가.
6. **Product:** 같은 조건에서 구현 실패가 재현되는가.

17.2 원인별 예

분류	예	조치
requirement defect	닫힌 세션·정원 부족의 우선순위 없음	owner 결정·REQ version 변경
case defect	TC-03에 remaining 초기값 누락	case 수정·review
data defect	중복 key가 이미 다른 test에서 사용됨	fixture reset·isolation
environment defect	candidate 대신 rule-bug variant 실행	build·flag 수정
oracle defect	실패 후 remaining 감소를 정상으로 기대	acceptance·expected 수정 승인
product defect	0석이 실제 예약 record 생성	defect report·fix·regression

17.3 Defect report는 최소 재현 계약입니다

field	질문
linked REQ·TC·run	왜·어디서 발견했나
build·environment·data	무엇에서 다시 만들 수 있나
minimum reproduction	불필요한 단계를 제거했나
expected·actual diff	어떤 field가 다른가
severity·impact	사용자에게 어떤 결과인가
evidence	다시 검증 가능한가
owner·fix version	누가 언제 고치나

field	질문
confirmation·regression	수정과 재발 방지를 확인했나

17.4 증상과 root cause를 구분합니다

“예약이 두 개 보인다”는 symptom입니다. Root cause는 idempotency key 저장 시점, transaction, retry 정책, unique constraint 누락 등일 수 있습니다. 테스트는 재현 조건과 관찰 차이를 정확히 제공하고, 원인 분석은 관련 owner와 함께 합니다.

18. 수정된 실패를 regression 자산으로 바꿉니다

18.1 Confirmation과 regression은 다릅니다

활동	질문
confirmation testing	보고된 defect가 같은 조건에서 고쳐졌는가
regression testing	그 변경 때문에 기존 다른 행동이 깨지지 않았는가

TC-03·TC-06을 수정한 뒤 두 케이스만 재실행하면 confirmation입니다. 나머지 정상·rule·state·idempotency까지 다시 보면 regression입니다.

18.2 Regression case에 남길 것

- 사고·defect ID와 사용자 영향.
- 최소 재현 Given·When·Then.
- 과거 잘못된 actual과 수정 expected.
- fix version과 confirmation evidence.
- suite·trigger·owner.
- 제거·변경 조건.

18.3 자동화 후보를 고릅니다

자동화에 유리	수동·전문 검토가 유리
반복 빈도가 높음	탐색·새 요구 이해가 목적

자동화에 유리	수동·전문 검토가 유리
결과가 결정적	시각·사용성·해석이 큼
setup·data를 고정 가능	실제 device·외부 환경 변동이 큼
P0·regression 보호	요구가 빠르게 바뀌는 초기 단계
실행·판정 비용 절감	자동화 유지비가 가치보다 큼

자동화는 케이스 설계를 대신하지 않습니다. 잘못된 oracle을 빠르게 반복하면 더 빠르게 잘못된 확신을 만듭니다.

18.4 Flaky test를 통과율에 숨기지 않습니다

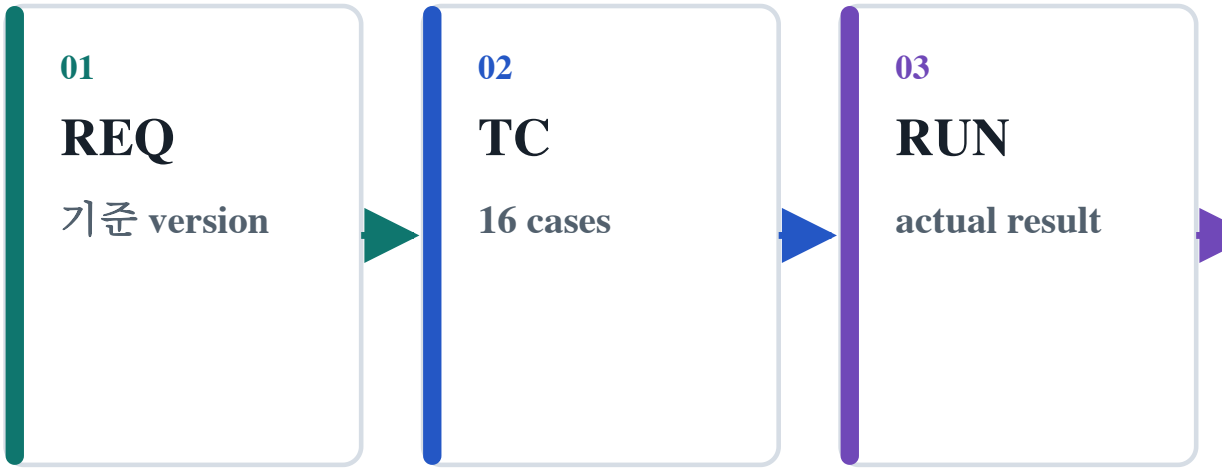
제품 변경 없이 PASS·FAIL이 바뀌면 time·network·shared state·random·selector·async wait를 진단합니다. Quarantine은 임시 통제이며, owner·기한·복귀 gate 없이 영구 격리하지 않습니다.

19. Release에는 테스트 목록이 아니라 증거 묶음이 필요합니다

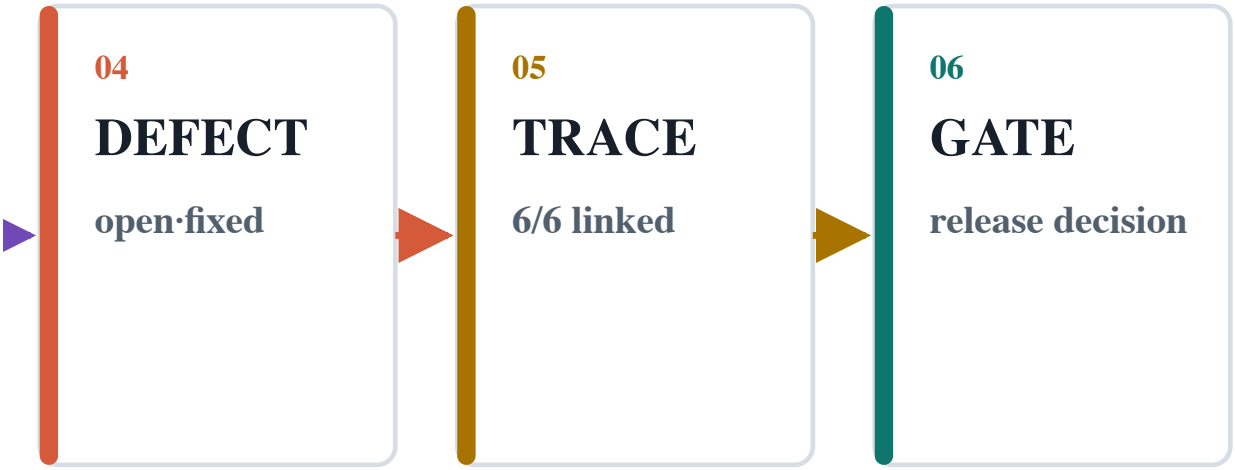
그림 14. REQ·TC·RUN·DEFECT·TRACE·GATE를 같은 version chain으로 묶어야 release 결정이 재현됩니다.

Release 기준 테스트 추적 구조

요구사항부터 판정까지 같은 version chain으로 보관합니다.



변경마다 같은 package를



새 version으로 다시 실행

19.1 Evidence package의 최소 구성

묶음	포함 내용
REQ	baseline·owner·rationale·acceptance
TC	conditions·techniques·coverage items·case version
RUN	build·environment·data·actual·verdict
DEFECT	open·fixed·severity·confirmation·regression
TRACE	REQ↔TC↔result↔defect·orphans·change impact
GATE	thresholds·actual·residual risk·approver·decision

19.2 실습의 release gate

gate	threshold
requirement coverage	6/6 · 100%
case pass	16/16 · 100%
high-risk case pass	100%
orphan requirement	0
orphan test case	0
duplicate case ID	0
release regression	4/4 PASS

이 threshold는 합성 학습 시스템의 값입니다. 실제 서비스는 위험·규정·운영 복구·표본·환경에 맞춰 정합니다.

19.3 Gate 통과도 범위를 넘는 주장을 하지 않습니다

`candidate-v3`가 16/16을 통과해도 합성 예약 규칙 6개와 선택한 case 범위에서 준비됐다는 뜻입니다. 실제 부하·권한·보안·동시성·외부 결제·접근성을 모두 증명하지 않습니다.

19.4 Conditional go에는 만료가 필요합니다

미충족 기준을 예외 승인한다면 영향·임시 통제·owner·재검증일·만료일·rollback 조건을 기록합니다. “나중에 고침”은 승인 근거가 아닙니다.

20. 테스트 설계 스튜디오에서 세 variant를 비교합니다

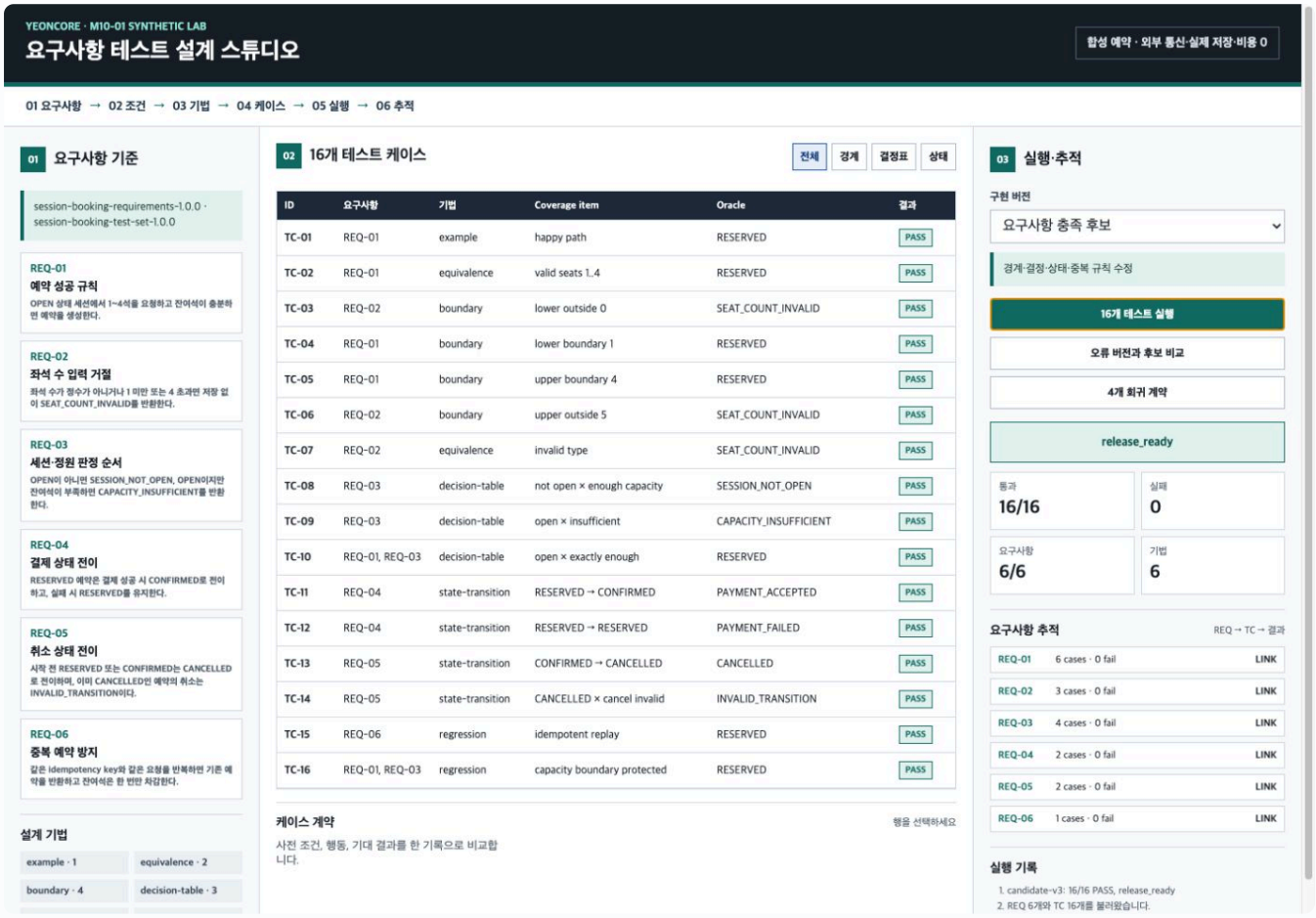


그림 15. 데스크톱 화면. 왼쪽의 version·요구사항, 가운데의 16개 case, 오른쪽의 결과·trace를 한 번에 비교합니다.

20.1 화면을 왼쪽에서 오른쪽으로 읽습니다

왼쪽	가운데	오른쪽
어떤 REQ·variant·gate인가	어떤 TC·technique·expected·actual인가	몇 개가 통과했고 무엇이 연결됐는가

20.2 Variant별 학습 포인트

variant	실패	탐지하는 기법	decision
boundary-bug-v1	TC-03·TC-06	boundary value	defects_detected
rule-bug-v2	TC-08·12·14·15	decision·state·regression	defects_detected
candidate-v3	없음	6 techniques·trace gate	release_ready

20.3 Candidate 통과를 case 단위로 읽습니다

1. 16/16 pass인지 봅니다.
2. High-risk cases가 모두 pass인지 봅니다.
3. 6/6 requirements가 적어도 하나의 TC와 연결됐는지 봅니다.
4. Example·equivalence·boundary·decision·state·regression 여섯 기법을 덮는지 봅니다.
5. Orphan·duplicate가 0인지 봅니다.
6. 4/4 regression을 별도로 실행합니다.

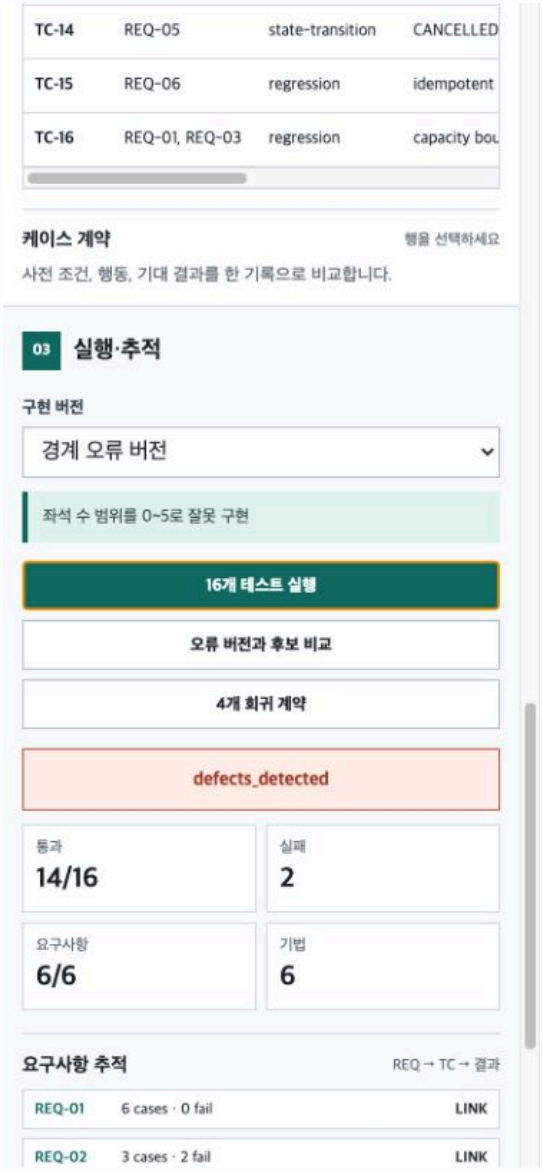


그림 16. 모바일 화면에서도 핵심 실행 결과와 실패 두 건을 세로 흐름으로 확인할 수 있습니다.

20.4 작은 화면에서도 정보 우선순위를 유지합니다

모바일에서는 panel이 세로로 쌓입니다. 먼저 decision·pass·fail을 보고, 실패 row에서 TC·coverage item·expected·actual을 확인한 뒤, 마지막에 요구사항 trace로 거슬러 올라갑니다.

20.5 자동 evidence

생성기는 다음을 먼저 검증합니다.

```
219 automated tests → OK
45/45 design contract audits → PASS
6 requirements · 16 test cases
boundary bug → exactly 2 failures
rule bug → exactly 4 failures
candidate → 16/16 PASS
release regression → 4/4 PASS
```

테스트 수가 많아서 좋은 것이 아닙니다. 계약의 구조·기법·추적·안전 경계·예상 실패를 자동으로 다시 확인할 수 있다는 점이 중요합니다.

21. 요구사항·테스트 케이스·추적성 기록서를 완성합니다

템플릿은 16개 section으로 구성됩니다. 처음부터 모든 칸을 채우지 않습니다.

21.1 먼저 채울 여섯 칸

1. Test basis와 requirement baseline.
2. 요구사항 원자화와 모호성 질문.
3. Acceptance criteria와 positive·negative conditions.
4. 선택 기법과 coverage items.
5. 최소 case record 하나.
6. REQ↔TC trace link.

21.2 실행할 때 채울 칸

section	실행 중 기록
data·environment	build·flags·fixture·seed
run header	run ID·operator·time
execution log	actual·verdict·evidence
defect	minimum reproduction·impact·fix
RTM	latest result·defect·coverage
gate	threshold·actual·residual risk

21.3 템플릿 review 질문

- 각 REQ에 owner와 version이 있는가.
- 각 condition에 positive·negative와 risk가 있는가.
- 각 TC에 technique과 coverage item이 있는가.
- Expected가 관찰 가능한 field와 side effect를 포함하는가.
- REQ에서 result로, result에서 REQ로 돌아갈 수 있는가.
- 변경된 REQ가 영향 TC를 표시하는가.
- Release decision에 approver와 잔여 위험이 있는가.

22. 현재 표준과 변하지 않는 원칙을 구분합니다

22.1 바뀌기 어려운 공통 원칙

- 요구사항은 검증 가능하고 추적 가능해야 합니다.
- Test analysis와 design을 구분합니다.
- 대표값 선택에는 설명 가능한 기법이 필요합니다.
- Expected와 actual을 관찰 가능한 기준으로 비교합니다.
- 실패 원인을 진단한 뒤 defect를 확정합니다.
- 변경할 때 영향 케이스와 회귀를 다시 실행합니다.
- Release 결정은 versioned evidence를 필요로 합니다.

22.2 도구·조직에 따라 바뀌는 구현

바뀔 수 있는 것	유지할 계약
Test management tool	stable REQ·TC·run·defect IDs
Gherkin·table·code 형식	precondition·action·observable expected
CI provider	동일한 build·environment·data·verdict 기록
Browser automation library	technique·coverage·oracle·trace
조직 role 이름	requirement·test·data·release owner 책임

23. 공식 근거와 추가 읽기

23.1 ISTQB CTFL v4.0.1

ISTQB Certified Tester Foundation Level Syllabus v4.0.1은 test analysis·test design·traceability와 black-box techniques인 equivalence partitioning, boundary value analysis, decision table testing, state transition testing의 공통 기준을 제공합니다.

이 매뉴얼에 적용한 핵심:

- Test basis에서 test conditions을 찾고, conditions에서 cases를 설계합니다.
- Traceability는 coverage·변경 영향·감사 가능성을 돕습니다.
- EP는 non-overlapping·non-empty partitions와 대표값을 사용합니다.
- BVA는 partition 경계와 인접값을 봅니다.
- Decision table은 실행 가능한 조건 조합을 coverage items로 다룹니다.
- State testing은 valid·invalid transitions와 상태 행동을 봅니다.

23.2 ISO/IEC/IEEE 29119 series

ISO/IEC/IEEE 29119 series 공식 안내는 software testing의 개념·process·documentation·techniques를 시리즈로 정리합니다. Part 2는 process, Part 3는 test documentation, Part 4는 test techniques의 공통 틀을 제공합니다.

이 매뉴얼은 특정 조직이 표준 인증을 받았다고 주장하지 않습니다. 용어·과정·문서·기법을 일관된 작업 흐름으로 정리하는 참고 근거로 사용합니다.

23.3 NASA Systems Engineering Handbook

NASA Systems Engineering Handbook은 요구사항의 unique ID·rationale·source·owner·verification 관련 metadata와 양방향 traceability의 중요성을 설명합니다.

이 매뉴얼의 `REQ↔COND↔TC↔RESULT↔DEFECT` chain은 해당 원칙을 학습 규모의 software test record에 적용한 것입니다.

23.4 Cucumber Gherkin Reference

Cucumber Gherkin Reference는 Given·When·Then, Scenario Outline, Examples, Rule의 의미를 설명합니다. 이 매뉴얼은 Gherkin을 UI macro가 아니라 business rule의 실행 가능한 예를 표현하는 언어로 사용합니다.

23.5 근거 사용 원칙

1. 표준과 공식 문서는 공통 용어와 원칙을 정렬하는 데 사용합니다.

2. 실제 threshold·suite·approval은 제품 위험과 조직 책임에 맞게 정합니다.
3. Edition·도구·policy가 바뀌면 review 날짜와 version을 갱신합니다.
4. 이 학습 매뉴얼은 전문 보안·법률·안전 인증을 대체하지 않습니다.

24. 셀프 테스트 30문항

1. Requirement를 바로 test steps로 바꾸면 왜 위험한가?

정답 보기
무엇을 검증할 test condition과 어떤 값을 고를 design technique가 섞여 모호함·경계·조건 조합·불변식이 누락될 수 있기 때문입니다.

2. Test analysis와 test design의 차이는 무엇인가?

정답 보기
Analysis는 test basis에서 testable feature와 condition을 찾는 일이고, design은 condition에서 coverage item·data·case를 만드는 일입니다.

3. 판정 가능한 요구사항의 여섯 원자는 무엇인가?

정답 보기
대상, 시작 상태, 입력 값·범위, 조건·우선순위, 관찰 결과, 유지돼야 할 불변식입니다.

4. “적절히 예약한다”를 그대로 expected로 쓸 수 없는 이유는?

정답 보기
범위·형식·상태·우선순위·관찰 결과가 정해지지 않아 실행자마다 다른 판정을 내릴 수 있기 때문입니다.

5. 모호함을 테스터가 조용히 가정하면 어떤 문제가 생기는가?

정답 보기
테스트가 승인되지 않은 숨은 요구사항이 되고 실패 뒤 요구 의미를 둘러싼 논쟁과 잘못된 결함이 생깁니다.

6. Test case의 최소 필드는 무엇인가?

정답 보기
고유 ID·REQ/condition 링크·기법/coverage item·precondition·data·한 action·observable expected·oracle·owner·실행 evidence가 필요합니다.

7. Steps가 길수록 좋은 case가 아닌 이유는?

정답 보기
긴 UI 절차는 구현 변화에 취약하고 실패 원인을 흐립니다. 재현 가능한 시작 상태·한 사건·관찰 가능한 결과가 더 중요합니다.

8. Equivalence partitioning의 목적은 무엇인가?

정답 보기
모든 값을 실행하지 않고 같은 처리 결과가 예상되는 집합마다 대표값을 선택하는 것입니다.

9. 좋은 partition의 세 조건은 무엇인가?

정답 보기
서로 겹치지 않고, 비어 있지 않으며, 같은 집합의 값이 같은 방식으로 처리될 근거가 있어야 합니다.

10. Valid partition만 검사하면 왜 부족한가?

정답 보기
형식 오류·범위 밖·누락 값의 거절과 오류 뒤 부작용 0을 확인하지 못해 invalid 처리 결함을 놓칩니다.

11. 1~4석의 핵심 경계값 네 개는 무엇인가?

정답 보기
하한 바로 밖 0, 유효 하한 1, 유효 상한 4, 상한 바로 밖 5입니다.

12. 대표값 2 하나로 1~4 범위를 검증할 수 없는 이유는?

정답 보기

0~5를 허용하는 잘못된 구현도 2에서는 성공하므로 포함·제외와 off-by-one 결함을 드러내지 못합니다.

13. Boundary가 숫자 외 어디에 있는가?

정답 보기

문자 길이, 목록 개수, 날짜·자정·timezone, 금액·반올림, 권한 만료, 저장 한도 등에 있습니다.

14. Decision table의 한 feasible column은 무엇을 뜻하는가?

정답 보기

실제로 발생 가능한 한 조건 조합과 그 기대 행동이며, 적어도 하나의 test case로 덮어야 할 coverage item입니다.

15. 결정표의 빈 행동 열은 무엇을 알려 줄 수 있는가?

정답 보기

가능한 조건 조합의 expected behavior가 요구사항에서 누락됐음을 알려 줄 수 있습니다.

16. 세션이 닫혔고 정원도 부족할 때 오류가 왜 하나로 고정돼야 하는가?

정답 보기

조건 우선순위가 없으면 구현과 테스트마다 다른 결과를 선택해 재현 가능한 oracle을 만들 수 없기 때문입니다.

17. State transition case의 다섯 요소는 무엇인가?

정답 보기

시작 state, event, guard condition, transition action, target state입니다.

18. Invalid transition도 검사해야 하는 이유는?

정답 보기

금지된 행동이 성공처럼 처리되거나 거절 뒤 state·quantity가 변하는 결함을 찾기 위해서입니다.

19. Given·When·Then의 역할은 각각 무엇인가?

정답 보기

Given은 알려진 초기 상태, When은 결과를 일으키는 한 사건, Then은 외부에서 관찰 가능한 결과와 불변식입니다.

20. Scenario Outline이 설계 기법을 대신하지 못하는 이유는?

정답 보기

Outline은 선택된 값을 반복 표현하는 형식일 뿐 어떤 partition·boundary·rule을 고를지는 EP·BVA·결정표 같은 기법으로 결정해야 합니다.

21. Test oracle은 무엇인가?

정답 보기

실제 결과가 올바른지 비교하는 요구사항·규칙·reference·invariant·독립 계산 등의 판정 기준입니다.

22. Result code가 같아도 FAIL일 수 있는 이유는?

정답 보기

상태·수량·저장 record·외부 event 같은 부작용이 expected와 다를 수 있기 때문입니다.

23. Expected를 actual을 본 뒤 수정하면 왜 위험한가?

정답 보기

요구사항을 검증하는 대신 현재 구현을 정답으로 승인하게 되며 결함을 oracle에 흡수할 수 있기 때문입니다.

24. Bidirectional traceability의 두 방향 질문은 무엇인가?

정답 보기

요구사항에서 어떤 cases·results가 검증하는지 내려가고, 실패·defect에서 어떤 requirement·rule 때문에 존재하는지 올라가는 질문입니다.

25. Orphan requirement와 orphan test case는 각각 무엇인가?

정답 보기
전자는 연결된 case가 없는 요구사항이고, 후자는 검증할 requirement·risk·rule 링크가 없는 case입니다.

26. FAIL이 곧 product defect가 아닌 이유는?

정답 보기
요구사항·case·data·environment·oracle·evidence 자체가 잘못돼도 같은 FAIL 증상이 나기 때문입니다.

27. Confirmation testing과 regression testing의 차이는?

정답 보기
Confirmation은 보고된 결함이 고쳐졌는지 같은 조건으로 확인하고, regression은 그 변경이 기존 행동을 깨지 않았는지 넓게 재검사합니다.

28. 자동화하기 좋은 case의 특징은?

정답 보기
반복 빈도와 위험이 높고, 결과가 결정적이며, setup·data를 고정할 수 있고, 유지비보다 반복 가치가 큰 case입니다.

29. Evidence package에 최소 무엇이 필요한가?

정답 보기
Requirement baseline, versioned test cases, run·actual·verdict, defect·fix·regression, traceability·coverage, release gate·잔여 위험·승인이 필요합니다.

30. Candidate 16/16 통과가 모든 품질을 증명하지 않는 이유는?

정답 보기
합성 요구사항 6개와 선택한 case·환경의 범위만 증명하며 실제 부하·권한·보안·동시성·외부 연동 전체는 별도 검증이 필요하기 때문입니다.

25. 한 장 요약

단계	핵심 질문	남길 evidence
Basis	무엇이 기준인가	REQ·owner·version·rationale
분석	무엇을 검증하나	test conditions·risk
EP	같은 처리 집합은	valid·invalid partitions·대표값
BVA	규칙이 바뀌는 안팎은	lower·upper probes
결정표	어떤 조건 조합인가	feasible rules·priority·TC link
상태전이	어떤 이동·거절인가	state·event·guard·invariant
Case	누가 실행해도 같은가	Given·When·Then·oracle
Run	실제로 무엇이 나왔나	build·environment·actual·verdict
진단	진짜 product defect인가	REQ·case·env·oracle·evidence review
추적	무엇이 덮이고 영향받나	RTM·coverage·orphans·change impact
회귀	고친 실패가 다시 막히나	confirmation·protected cases
Release	어떤 근거로 결정하나	evidence package·gate·residual risk

마지막 판별

좋은 테스트 케이스는 “어떻게 눌렀는가”를 자세히 기록한 절차가 아닙니다. 어떤 version의 요구사항에서 어떤 조건을 뽑아 왜 그 대표값을 골랐고, 어떤 시작 상태와 oracle로 실제 결과를 판정했으며, 그 증거가 변경과 release 결정에 어떻게 연결되는지를 다시 설명할 수 있는 기록입니다.