> how-it-works · agentic-qa · test-documentation

El Arte de Documentar Test Cases

Scopes, nomenclatura perfecta, técnicas de diseño, parametrización y el arte de priorizar con ROI — el puente entre la QA manual y la automatización, con o sin Xray.

Analiza → Prioriza → Documenta

la disciplina que convierte comportamiento validado en regresión confiable

// la forma sana

80 → 12 → 8

Analizas **80** escenarios, documentas **12**, automatizas **8**. La mayoría debe terminar **Deferred**. Documentar todo es el anti-patrón: un test entra al repositorio porque se va a re-ejecutar, nunca para llegar a un número de cobertura.

// qué cubriremos

Cinco Disciplinas del Test Case

4 quizzes interactivos a lo largo del recorrido — haz clic en las respuestas.

// qué es este skill

El Puente entre QA Manual y Automatización

1 **Analyze**

Identifica escenarios reales (flujos de usuario), separa lo transversal (XSS, perf, a11y se validan DENTRO de otros TCs, no son TCs).

2 **Prioritize (ROI)**

Cada escenario pasa 3 filtros + fórmula ROI → un veredicto: Candidate, Manual o Deferred. Nunca se salta.

3 **Document**

Crea TC/ATP/ATR en el TMS con trazabilidad completa, según la modalidad (Xray o Jira-native).

Prerrequisito duro

Solo se documenta comportamiento **ya validado** (US `QA Approved`, bug cerrado, sesión exploratoria terminada). El TMS es herramienta de documentación y protección de regresión — **no** de exploración.

PARTE 01

Los Scopes
Elige por el Input, no por el Output

Cuatro scopes comparten el mismo pipeline Analyze→Prioritize→Document. Solo cambian la fuente de entrada y los defaults. El input decide el scope.

// los cuatro scopes

Módulo · Historia · Bug · Exploratorio

SCOPE	INPUT	VOLUMEN	CUÁNDO
Module-driven	Un módulo explorado end-to-end	20-100+	Batch bajo el Epic de Regresión. La mayoría será Deferred.
Ticket-driven	Una US `QA Approved` de sprint	3-8	Salida de una sesión sprint-testing. ATP/ATR por US.
Bug-driven	Un bug cerrado con fix verificado	0-2	Corre la decisión bug-driven. ROI sesgado arriba: "falló una vez, puede fallar de nuevo".
Ad-hoc / Exploratorio	Escenarios nuevos de testing exploratorio	1-10	Aplica los 3 filtros con dureza — suelen ser validaciones de una sola vez.

Mismo pipeline, distinto punto de entrada

El scope NO cambia el método (Analyze→Prioritize→Document); cambia de dónde sacas los escenarios y qué labels por defecto aplicas.

// los dos scopes de volumen

Module-driven vs Ticket-driven

Module-driven (Macro)

Exploras un módulo completo. 20-100+ escenarios. Batch bajo el **Epic de Regresión**, agrupado por un **Test Set** (1:1 con el Epic/feature). La mayoría termina Deferred — derivas amplio, persistes poco.

`regression` `e2e` `integration`

Ticket-driven (Medium)

Una US aprobada en sprint. 3-8 escenarios. ATP/ATR creados por historia. Es la salida natural de una sesión de `/sprint-testing` que pasa a documentarse.

`regression` `+ test type`

Source-code validation (obligatorio)

El ATP se escribió ANTES del código. Antes de crear cualquier TC: grep `data-testid=`, rutas, formatos de texto. Corrige divergencias en una sección Refinement Notes. Saltarlo = la causa #1 de tests automatizados inválidos después.

// la regla de oro

Bug-driven — "Donde Hay un Bug Importante, Debe Haber un Test"

No todo bug se vuelve TC de regresión. Pero si ES regression-worthy, **debe** terminar con un Test — en ambas modalidades.

decisión bug-driven

```
# 1. ¿Es candidato de regresión? (filtros Phase-0 + ROI; la regla de bug previo sesga arriba)
   NO  → Sin Test nuevo. Trátalo como test fallido (fix ya verificado) → Deferred.
   YES → paso 2.

# 2. ¿El bug salió de un Test ya ejecutado que falló?
   YES → REUSA ese Test para retest + regresión. Ya existe; enlázalo (tests / is tested by). NO dupliques.
```

Ad-hoc / Exploratorio

Escenarios sueltos de exploración. Aplica los 3 filtros con dureza — son a menudo validaciones de una sola vez → casi siempre Deferred.

// quiz · parte 1

¿Qué Scope Aplica?

Q1 Te dan un bug cerrado con fix verificado: un error de cálculo en el carrito que ya ocurrió dos veces antes.

- A Bug-driven — regression-worthy (bug previo), debe terminar con un Test
- B Ad-hoc — es un hallazgo suelto
- C Module-driven — toca el módulo de carrito

Bug-driven. El input es un bug cerrado → scope bug-driven. Con bugs previos en el área, el ROI se sesga arriba ("falló, puede fallar de nuevo") → Candidate, y por la regla de oro DEBE existir un Test (reusa o crea).

Q2 QA terminó de explorar end-to-end el módulo de pagos y tiene ~60 escenarios validados sin ticket asociado.

- A Ticket-driven — hay que crear ATP/ATR por cada uno
- B Module-driven — batch bajo el Epic de Regresión, la mayoría Deferred
- C Bug-driven — son fallos potenciales

Module-driven. El input es un módulo explorado completo (20-100+) → batch agrupado en un Test Set bajo el Epic de Regresión. Deriva amplio, persiste poco: la mayoría de los 60 será Deferred.

PARTE 02

Nomenclatura
El Título Perfecto

Una sola regla de nombrado, una sola regla de identidad. Un nombre mal puesto es un test que nadie encuentra, nadie entiende y nadie mantiene.

// la regla que más importa

El Formato Canónico del Título

GX-101 : **TC1** : **should** **log in successfully**

when credentials are valid

US_ID — SIEMPRE LA STORY · TC# SECUENCIAL · VERBO LITERAL (BDD) · BEHAVIOR — QUÉ HACE · CONDITION — WHEN/IF · OPCIONAL

BEHAVIOR — el comportamiento

qué hace el sistema: `log in successfully`, `show an authentication error`, `create the order`.

CONDITION — la condición que distingue

`when credentials are valid`, `when password is incorrect`, `when exceeding 5 failed attempts`.

Reglas estructurales

Separador literal `:` (dos puntos + espacio). El verbo `should` abre el comportamiento; el conector `when` (o `if`) introduce la condición — opcional, igual que un `given <precondición>` al final. `TC#` = `TC` + número, sin guion ni ceros (`TC1`, no `TC-01`). El prefijo es **SIEMPRE el US_ID** de la Story, en toda modalidad; la membresía a un Test Set es un **link**, nunca el prefijo.

// anti-patrones a rechazar

Títulos: Mal → Bien

❌ MAL	✓ BIEN	POR QUÉ
`Login test`	`GX-101: TC1: should log in successfully when credentials are valid`	Falta ID, TC#, behavior, condition
`Login - error`	`GX-101: TC2: should show an authentication error when password is incorrect`	Demasiado vago
`TC1: Test form`	`GX-101: TC1: should submit the form when all fields are valid`	Falta ID; behavior no específico
`should work`	`GX-101: TC1: should log in successfully when credentials are valid`	should sin behavior ni condición concretos

Ejemplos por tipo de test

Positivo: `should log in successfully when credentials are valid` · Negativo: `should show an authentication error when password is incorrect` · Boundary: `should enforce the limit when entering exactly 50 chars` · Edge: `should merge quantities when the cart has multiple same items`.

// la segunda regla de oro

Un TC = Una (Precondición + Acción)

Todos los resultados esperados del mismo par (precondición, acción) pertenecen al **mismo** TC — no a TCs separados.

✓ Mismo TC, múltiples assertions

- Precond: credenciales válidas + cuenta activa
- Acción: submit login
- Esperado: redirect + token + perfil vía /auth/me + cookie + saludo

vs

TCs distintos — precondición cambia

- TC-A: credenciales válidas → success
- TC-B: cuenta bloqueada → 423
- TC-C: credenciales inválidas → 401

El anti-patrón #1 en reviews

Partir un (precondición, acción) en "TC1: check panel A", "TC2: check panel B"... es el error más diagnosticado. Es **un** TC con múltiples assertions, no tres TCs.

// el sistema completo de nombrado

No Solo el TC — Todo el TMS

ARTEFACTO	FORMATO	EJEMPLO
User Story	`{KEY}-{n}`	`PROJ-123`
ATP — Story Test Plan	`ATP: {STORY-KEY}: {story title}`	`ATP: PROJ-123: Apply discount at checkout`
ATR — Story Test Execution	`ATR: {STORY-KEY}: Story Testing`	`ATR: PROJ-123: Story Testing`
STP/STR — Sprint	`STP/STR: Sprint#{N}: Regression[ Testing]`	`STR: Sprint#30: Regression Testing`
Test Set (agrupa)	`TS: {EPIC-KEY|module}: Validate <feature>`	`TS: GX-101: Validate credit card payment`
Test Case (asevera)	`{US_ID}: TC#: should <BEHAVIOR> when <COND>`	`GX-101: TC1: should log in successfully when…`
ReTesting (bug)	`ReTest: {BUG-KEY}: {summary}`	`ReTest: GX-202: wrong error on bad password`
Precondition	`PRC: {COMPONENT}: {required state}`	`PRC: Payment: Authenticated user with a saved card`

Dos niveles: `Validate` agrupa, `should` asevera

El **Set** nombra la feature — `TS: GX-101: Validate login` — y bajo él cada **TC** asevera un comportamiento: `should redirect to dashboard when credentials are valid`, `should show an error when password is incorrect`. El prefijo acrónimo (`ATP/ATR/STP/STR/TS/PRC`) revela altitud y plan-vs-run en el primer token; el verbo te dice el nivel: `Validate` = suite, `should` = caso.

// trazabilidad end-to-end

El Mismo Key — y la Misma Voz

En el TMS (título)

```
GX-101: TC1: should log in successfully when credentials
are valid
```

Voz BDD: **should** <behavior> **when** <condition>.

En el código (KATA)

```
@atc('GX-101')
test('GX-101: should log in when credentials valid', …)
```

Voz BDD: **should** <behavior> **when** <condition>.

Mismo key y misma voz, del requisito al código

El título del TC y el nombre del `test()` hablan ambos `should…when` y llevan el mismo key del TMS. No solo coincide el identificador — coincide la **frase**. Eso es trazabilidad sin costuras: el outline, el TC y el código dicen lo mismo.

// quiz · parte 2

Nomenclatura & Identidad

Q1 ¿Cuál título cumple la convención para un caso negativo de login con contraseña incorrecta?

A `Login - error`

B `GX-101: TC2: should show an authentication error when password is incorrect`

C `Validate login fails`

B. Tiene los segmentos: ID (`GX-101`) + TC# (`TC2`) + `should` + BEHAVIOR (`show an authentication error`) + CONDITION (`when password is incorrect`). A es vago y sin ID; C usa `Validate` (verbo de Set, no de TC) y le falta todo. El TC habla should; Validate se reserva para el Set.

Q2 Login exitoso debe: redirigir al dashboard, devolver token, setear cookie y mostrar saludo. ¿Cuántos TCs?

A Un TC con 4 assertions — misma precondición + acción

B Cuatro TCs, uno por cada validación

C Dos TCs: uno de UI y uno de API

Un TC. Misma (precondición + acción) = mismo TC, sin importar cuántos resultados esperados verifique. Partirlo en 4 "check X" es el anti-patrón más diagnosticado en reviews.

PARTE 03

El Cuerpo
Técnicas, Gherkin y Parametrización

Cómo se escribe un TC: Gherkin para candidatos, tablas para manual; técnicas de diseño que deciden el SET de TCs; parametrización con variables, nunca datos hardcodeados.

// cómo escribir el cuerpo

Gherkin vs Tradicional

CRITERIO	GHERKIN	TRADICIONAL
Candidato a automatización	Sí	No
Pasos deterministas	Sí	Pueden variar
Resultados esperados exactos	Sí	Subjetivos
Requiere juicio humano	No	Sí

La tabla tradicional (manual)

Columnas fijas: `Step | Action | Test Data | Expected Result`. Dato vacío = `-`. Úsala para verificación visual/subjetiva, elementos exploratorios o TCs marcados `manual-only`.

// el patrón de alta calidad

Anatomía de un Gherkin de Candidato

Scenario Outline

```
@critical @regression @automation-candidate @{US_ID}
Scenario Outline: should <behavior> when <condition>
  """ Bugs covered: BUG-1, BUG-2 · Related Story: {US_ID} · ROI: 5.2 """
  # === PRECONDITIONS (tester / script las construye) ===
  Given a <entity> exists with <identifier> where <quantity> <condition>
  # === ACTION ===
  When the user navigates to "<route>" and <main_action>
  # === VALIDATIONS ===
  Then <ui_element> is displayed with format "<expected_format>"
  # === EQUIVALENT PARTITIONS ===
  Examples: Happy path  |  Edge case  |  Singular vs plural
```

Siempre lleva

Tags (priority + `@regression` + `@automation-candidate` + `@{US_ID}`) · docstring con metadata · banners `# === … ===` · variables, nunca datos hardcodeados · un `Examples` por partición de equivalencia.

// 1 AC → N TCs (deriva por técnica)

Técnicas de Diseño — por Disparador

DISPARADOR EN EL AC	TÉCNICA	TCS QUE PRODUCE
Cualquier input (siempre)	**Equivalence Partitioning**	mismo output → 1 TC parametrizado; distinto output → TCs separados
Rango / límite / longitud / fecha	**Boundary Value Analysis**	`min-1·min·min+1 … max-1·max·max+1` + zero/empty/null/overflow
Un estado / ciclo de vida	**State-Transition**	1 TC por transición válida + 1 por inválida
2+ condiciones que interactúan	**Decision Table**	enumera combos, colapsa equivalentes, 1 TC por regla
3+ factores combinables	**Pairwise**	set all-pairs (registra la reducción)

Derivar es gratis, persistir es ROI-gated

Estas son candidatas derivadas por técnica — **aún no** work items del TMS. La técnica gobierna la **forma** de lo que documentas, no el **cuánto**. EP siempre; BVA donde haya rango; el resto por disparador.

// técnica 1/5 · teoría

Equivalence Partitioning

Particiona el dominio de entrada en **clases donde todos los miembros se comportan igual**. Prueba UN representante por clase — reduce infinitas entradas a pocos casos.

Disparador

Cualquier input (**siempre** aplica). Es el piso de todo diseño de TC.

Regla de colapso

Mismo output → **1 TC parametrizado**. Distinto output → **TCs separados**. Cubre clases **válidas E inválidas**.

El error común

Probar 5 emails válidos distintos es desperdiciar 4 TCs: todos viven en la **misma clase**. Un representante basta. El valor está en cubrir **todas** las clases, no en repetir dentro de una.

// técnica 1/5 · en la práctica

EP — Login por Clases de Salida

Clase válida → 200

credenciales correctas

`1 TC representante`

Clase inválida → 401

password mala · user inexistente · formato roto

`1 TC parametrizado (mismo 401)`

Clase bloqueada → 423

cuenta con 5 intentos fallidos

`TC separado (distinto output)`

El veredicto de diseño

Las 3 razones de fallo "credencial inválida" comparten salida (401) → **un** `Scenario Outline` con 3 filas de `Examples`. Pero 423 es otra clase de salida → **su propio TC**. EP decidió el SET: 3 TCs, no 5.

// técnica 2/5 · teoría

Boundary Value Analysis

Los defectos se **concentran en los bordes** de las clases de equivalencia. EP elige el representante; BVA ataca justo donde el código suele equivocarse: el límite.

Disparador

Rango · límite · longitud · ventana de fecha · cuota · paginación.
Donde haya un número que marque una frontera.

Los valores a probar

```
min-1 · min · min+1
max-1 · max · max+1
+ zero / empty / null / overflow
```

Por qué EP sola no basta

EP probaría "un valor válido y uno inválido" y se perdería el clásico **off-by-one** (< vs <=). BVA es **obligatoria** siempre que un AC nombre un rango o límite.

// técnica 2/5 · en la práctica

BVA — Password de 8 a 20 Caracteres

LONGITUD	POSICIÓN	ESPERADO
0 / vacío	zero / empty	`reject` "campo requerido"
7	min − 1	`reject` "mínimo 8"
8	min	`accept`
9	min + 1	`accept`
19	max − 1	`accept`
20	max	`accept`
21	max + 1	`reject` "máximo 20"

El foco está en 7-8 y 20-21

Esos cuatro valores prueban que la comparación usa `>=8` y `<=20` exactos. Como comparten salida por lado, van como filas de un `Examples` dentro de uno o dos TCs — no siete TCs sueltos.

// técnica 3/5 · teoría

State-Transition

Para entidades con **estado**, modela los estados y las transiciones entre ellos. Cada movimiento permitido y cada movimiento prohibido es un caso de prueba.

Disparador

Un campo de **estado** o ciclo de vida: orden, suscripción, ticket, cuenta, documento.

La regla

1 TC por **transición válida** + 1 por **transición inválida**. Intentar lo prohibido debe ser rechazado limpiamente.

Dónde viven los defectos

En las **transiciones inválidas**. El happy path (avanzar de estado) casi siempre funciona; lo que rompe es cuando alguien intenta saltar o retroceder un estado que el negocio no permite.

// técnica 3/5 · en la práctica

State — Ciclo de Vida de una Orden

Pending → Paid → Shipped → Delivered ✓

Transiciones válidas → 1 TC c/u

- Pending → Paid (pagar)
- Paid → Shipped (despachar)
- Pending → Cancelled (cancelar permitido)

Transiciones inválidas → 1 TC c/u

- Shipped → Cancelled (rechazar 409)
- Delivered → Paid (rechazar)
- Pending → Shipped (saltar Paid → rechazar)

// técnica 4/5 · teoría

Decision Table

Cuando 2+ condiciones interactúan para decidir una acción, enumera todas las combinaciones de condiciones → su resultado. Colapsa las imposibles o equivalentes. 1 TC por regla sobreviviente.

Disparador

2+ condiciones que se combinan: reglas de negocio con AND/OR, permisos, pricing, elegibilidad.

El método

2 condiciones → 2^2 = 4 combos. 3 → 8. Enumera, **colapsa** reglas equivalentes, queda un set mínimo que cubre toda la lógica.

Por qué gana a probar "casos sueltos"

Garantiza que **ninguna combinación** de condiciones queda sin probar — el hueco típico cuando se diseñan TCs a ojo.

// técnica 4/5 · en la práctica

Decision Table — Envío Gratis

Regla: envío gratis si **es miembro** O **el carrito > $50**.

REGLA	¿MIEMBRO?	¿CARRITO > $50?	ACCIÓN	TC
R1	Sí	Sí	Envío gratis	colapsan → 1 TC
R2	Sí	No	Envío gratis	
R3	No	Sí	Envío gratis	TC propio
R4	No	No	Cobra envío	TC propio

El colapso

R1 y R2 dan el mismo resultado porque **ser miembro** ya basta — el carrito es irrelevante ahí. Colapsan en 1 TC ("miembro → gratis sin importar carrito"). 4 combos → **3 TCs** que cubren toda la lógica.

// técnica 5/5 · teoría

Pairwise (All-Pairs)

Con 3+ factores combinables, la combinación completa explota. Pairwise cubre cada par de valores con muchísimos menos casos — porque la mayoría de defectos surgen de la interacción de dos factores, no de cinco a la vez.

Disparador

3+ factores que se combinan: browser × OS × plan, rol × permiso × feature flag, idioma × moneda × región.

La regla

Genera el set **all-pairs** y **registra la reducción** — que se vea "apliqué pairwise: 72 → 9", nunca un recorte silencioso.

El supuesto que la habilita

Pairwise asume que los bugs por interacción de **3+ factores simultáneos** son raros. Si tu dominio sí los tiene (ej. seguridad), complementa con casos dirigidos.

// técnica 5/5 · en la práctica

Pairwise — Compatibilidad de Plataforma

Browser

Chrome · Firefox · Safari

OS

Windows · macOS · Linux

Plan

Free · Pro

18

combinación completa

3 × 3 × 2 = todas las combinaciones

9

set pairwise

cada par (browser-OS, browser-plan, OS-plan) cubierto

–50%

casos ahorrados

misma cobertura de pares, mitad del esfuerzo

Lo que se documenta

Un solo TC parametrizado: `Scenario Outline` con 9 filas de `Examples` (browser, os, plan) + la nota "**pairwise: 18→9**" para que la reducción sea auditable.

// las 5 técnicas en una mirada

Técnica → Disparador → Ejemplo

TÉCNICA	DISPARADOR EN EL AC	PRODUCE	EJEMPLO DEL DECK
Equivalence Partitioning	Cualquier input *(siempre)*	1 TC por clase de salida	login: inválidas→401 (1 TC) vs bloqueada→423 (1 TC)
Boundary Value Analysis	Rango · límite · longitud · fecha	`min±1`, `max±1` + zero/null	password 8-20: prueba 7·8·9 y 19·20·21
State-Transition	Un campo de estado / ciclo de vida	1 TC por transición válida + inválida	orden: cancelar un `Shipped` → rechazo
Decision Table	2+ condiciones que interactúan	1 TC por regla (colapsa equivalentes)	envío gratis: 4 combos → 3 TCs
Pairwise	3+ factores combinables	set all-pairs (log de la reducción)	browser×OS×plan: 18 → 9

El mapa mental

EP es el piso (siempre aplica); las demás se **gatillan por la forma del AC.** Derivas amplio por técnica (1:N), persistes por ROI. La técnica gobierna la **forma** de lo que documentas, no el **cuánto**.

¿Qué Técnica Aplica?

Q1 El AC dice "el cupón acepta entre 5 y 100 caracteres". ¿Técnica obligatoria?

A Equivalence Partitioning — un válido, un inválido

B Boundary Value Analysis — 4·5·6 … 99·100·101 + vacío

C Pairwise — combinar longitudes

BVA. Hay un rango (5-100) → obligatoria. Prueba los bordes (min-1/min/min+1 y max-1/max/max+1) + vacío/null. EP elige el representante; BVA caza el off-by-one en la frontera.

Q2 El precio depende de DOS condiciones: ¿es miembro? y ¿pago anual? Quieres cubrir todas las combinaciones sin huecos.

A Decision Table — enumera las 4 combinaciones, colapsa equivalentes, 1 TC por regla

B Boundary Value Analysis — están en los bordes

C Equivalence Partitioning — una clase por condición

Decision Table. 2+ condiciones que interactúan → tabla de decisión. Garantiza que ninguna combinación queda sin probar (el hueco típico al diseñar a ojo), y colapsa las reglas que dan el mismo resultado.

// quiz · las 5 técnicas (2/2)

¿Qué Técnica Aplica?

Q3 Validar la app en Browser (3) × OS (3) × Plan (2) sin correr las 18 combinaciones completas.

A State-Transition — hay configuraciones

B Equivalence Partitioning — agrupa por clase

C Pairwise — cubre cada par con ~9 casos, log de la reducción 18→9

Pairwise. 3+ factores combinables → all-pairs. Cubre cada par (browser-OS, browser-plan, OS-plan) en ~9 casos en vez de 18. Asume que las interacciones de 2 factores cazan la mayoría de defectos.

Q4 Una orden no debe poder cancelarse una vez está "Shipped". ¿Qué técnica modela esto?

A State-Transition — 1 TC por transición inválida (Shipped → Cancelled rechazada)

B Boundary Value Analysis — está en el borde del flujo

C Decision Table — combina estado y acción

State-Transition. Es una entidad con estado; "cancelar un Shipped" es una transición **inválida** → su propio TC que verifica el rechazo. Ahí viven los bugs de máquina de estados.

// data-driven sin hardcodear

Dos Tablas que NO se Confunden

`Examples:` — qué cambia

Los valores que **varían** por corrida, una tabla por partición de equivalencia. Un valor que *cambia el caso* → columna de Examples.

`Variables` — cómo obtener

En la Description: explica **cómo obtener** cada variable en runtime (la query SQL). No varía por corrida. Un valor que *buscas para correr* → fila de Variables.

variables, nunca hardcode

```
# MAL — hardcoded
Given a mentor exists with user_id "550e8400-e29b-41d4-a716-446655440000"
# BIEN — patrón variable + tabla Variables que dice cómo obtenerlo
```

Único caso donde el literal SÍ va

Cuando el AC mismo define el valor: `Then the field must accept maximum 500 characters` o `Then the rating is displayed in "X.X/5.0" format`. Beneficios: durabilidad, portabilidad, claridad, automatización.

// quiz · parametrización

Parametrización

Q1 En un `Scenario Outline`, ¿qué va en la tabla `Examples` y qué en la tabla `Variables`?

- **A** Examples = los valores que cambian el caso (por partición); Variables = cómo obtener cada dato en runtime (la query)
- **B** Examples = las queries SQL; Variables = los valores de prueba
- **C** Son lo mismo, se usa cualquiera de las dos

A. Regla: un valor que *cambia el caso* → columna de `Examples` (uno por partición de equivalencia). Un valor que *buscas para poder correr* → fila de `Variables` con su query. Un TC parametrizado típicamente tiene ambas.

Q2 En el Gherkin necesitas el UUID de un mentor verificado. ¿Dónde va y cómo?

- **A** Hardcodeado en el Given para que sea reproducible
- **B** `{mentor_id}` + fila en la tabla Variables con la query SQL para obtenerlo
- **C** Como columna en Examples

B. Nunca hardcodees UUIDs/emails. Variable `{mentor_id}` + tabla Variables que dice cómo obtenerlo en runtime (no varía el caso → no es Examples). Durabilidad y portabilidad.

PARTE 04

Priorización
El Arte del ROI

La mayoría de los escenarios deben terminar Deferred. El ROI decide cuáles valen el costo de mantener: Valor sobre Costo, sesgado por Riesgo.

// deriva amplio, persiste poco

Tres Capas, Tres Conteos

1

Derivar (Diseño)

Considera muchos casos por técnica (1:N). Vive en el análisis de priorización, **no aún** en el TMS. Gratis.

2

Documentar (este skill)

Crea un TC persistente **solo** para lo que vale re-ejecutar: Candidate + Manual. Deferred va al reporte, **no** al TMS.

3

Automatizar

Los Candidates fluyen a `/test-automation`. Los pocos.

La forma sana: "80 → 12 → 8"

Analizaste 80, documentaste 12, automatizaste 8 — nunca "documenta los 80". Un test entra al repositorio porque se va a re-ejecutar, jamás para llegar a un conteo. Si >50% queda Candidate/Manual, re-aplica el filtro más duro.

// el gate antes del ROI

Tres Preguntas Filtro — Falla una → Deferred

1

¿Protege contra regresiones FUTURAS?

Si fue un typo de una sola vez en área estable, la respuesta es no → Defer.

2

¿Hay bugs PREVIOS en esta área?

Sí → prioriza aun con ROI moderado ("falló una vez, puede fallar de nuevo"). La regla de bug previo sobreescribe los umbrales.

3

¿Es concern de APP o de FEATURE?

XSS / a11y / performance / responsive son suites a nivel app, no TCs por feature → Defer de este scope (handoff explícito, no drop silencioso).

Lo transversal no es un TC

"Mobile responsive", "XSS", "Performance" se validan **dentro** de otros TCs o en una suite a nivel app — nunca como TC propio.

// valor sobre costo

ROI = (F × I × S) / (E × D)

Frequency × Impact × Stability — el **valor/riesgo** (multiplican) — dividido por Effort × Dependencies — el **costo** (dividen). Un "flujo crítico" con Effort=5 y Dependencies=5 tiene ROI bajo por diseño. Eso es correcto, no un bug de la fórmula.

// cada factor, escala 1-5

Los Cinco Factores

FACTOR	1	2	3	4	5
Frequency (cuán seguido corre)	Anual	Por release	Por sprint	Diario	Por PR/commit
Impact (si falla)	Cosmético	Molestia menor	Degrada UX	Bloquea feature	Revenue / core
Stability (del flujo)	Muy volátil	Inestable	Moderado	Estable	Sin cambios meses
Effort ÷ (automatizar)	Trivial	Bajo (horas)	Medio (1-2 días)	Alto (varios días)	Muy alto (semana+)
Dependencies ÷	Ninguna	1-2 simples	3-4	5+	Externos complejos

Bonus de valor por componente

Si un TC se reutiliza en N flujos E2E: `Component Value = Base ROI × (1 + 0.2 × N)`. Un átomo de ROI bajo como `authenticateSuccessfully` se vuelve automatizable puro por reutilización.

// cada escenario termina en uno

Candidate · Manual · Deferred

VEREDICTO	LO DISPARA	A DÓNDE VA
Candidate	ROI > 3.0, O (ROI 1.5-3.0 Y bug previo), O happy path crítico	Alimenta `/test-automation`. Draft→In Design→Ready→In Review→Candidate
Manual	ROI 0.5-1.5 Y no automatizable (juicio humano, inspección visual)	Terminal: suite de regresión manual. No es callejón sin salida — puede volver a In Review si el ROI cambia
Deferred	ROI < 0.5, O falla un filtro Phase-0, O validación de una sola vez	Terminal: NO entra a regresión. **jira-native**: no se crea TC, va al reporte. **jira-xray**: el Test de sprint queda sin promover

La modalidad cambia el verbo, no el veredicto

El veredicto ROI es idéntico en ambas modalidades. **jira-native**: Phase 3 *crea* Tests para Candidate+Manual. **jira-xray**: los Tests ya existen del sprint; Phase 3 los *promueve* al Test Plan de regresión y los enriquece.

// scoreando un escenario real

ROI en Acción

Checkout — pago con tarjeta (happy path)

Frequency = 5 (por PR)

Impact = 5 (revenue/core)

Stability = 4 (estable)

Effort = 3 (1-2 días) ÷

Dependencies = 4 (gateway pago) ÷

ROI = (5×5×4)/(3×4) = 100/12 ≈ 8.3

> 3.0 + happy path crítico → **Candidate**

Typo de tooltip en pantalla de ajustes

Frequency = 1 (rara vez)

Impact = 1 (cosmético)

Stability = 5 (no cambia)

Effort = 2 ÷

Dependencies = 1 ÷

ROI = (1×1×5)/(2×1) = 5/2 = 2.5

Sin bug previo + validación de una vez → falla Phase-0 → **Deferred**

// quiz · parte 4

Priorización & ROI

Q1 Un flujo crítico tiene Effort=5 y Dependencies=5, dando ROI bajo. ¿Qué significa?

- **A** Es correcto por diseño — Effort y Dependencies son divisores; automatizarlo cuesta mucho
- **B** Es un bug de la fórmula — un flujo crítico siempre debe ser Candidate
- **C** Hay que ignorar el ROI y automatizarlo igual

A. Effort y Dependencies van en el denominador: más alto = peor ROI. Un flujo crítico carísimo de automatizar con muchas dependencias tiene ROI bajo a propósito — quizás Manual hasta que baje el costo.

Q2 Un escenario tiene ROI 2.2, pero hay un bug cerrado previo justo en esa área. ¿Veredicto?

- **A** Deferred — ROI debajo de 3.0
- **B** Candidate — la regla de bug previo sobreescribe (ROI 1.5-3.0 + bug previo)
- **C** Manual — ROI intermedio siempre es manual

Candidate. "Falló una vez, puede fallar de nuevo": un bug previo en el área promueve a Candidate aun con ROI 1.5-3.0. La regla de bug previo sobreescribe los umbrales.

PARTE 05

El TMS
Xray o Jira-native

Los mismos conceptos (ATP, ATR, TC) viven en contenedores distintos según la modalidad. Resuélvela en Phase 0, antes de documentar.

// mismos conceptos, distinto contenedor

Xray vs Jira-native — el Mapeo

ARTEFACTO	MODALITY JIRA-XRAY	MODALITY JIRA-NATIVE
ATP	Issue `Test Plan`, enlazado al Story	Campo `{{acceptance_test_plan}}` del Story (o comment fallback). Sin issue.
ATR	Issue `Test Execution` con Test Runs por TC, Environment, fechas	Campo `{{acceptance_test_results}}` del Story. Sin issue.
TC	Issue `Test` (Manual / Cucumber / Generic)	Issue type `Test` nativo; Description lleva el template completo
Test Set / Precondition	Issue types de primera clase de Xray	No existen — labels + agrupación por Epic
Sync de resultados	CI importa JUnit/Cucumber → Test Runs auto-actualizan	Script custom actualiza el campo Test Status por TC

Resuélvela primero (Phase 0)

¿El proyecto tiene Xray instalado y licenciado? Auto-resuelve por `{{TMS_CLI}}`, luego `master-test-plan.md`, luego listando issue types. Solo pregunta si los 3 fallan. Sticky — no la re-resuelvas a mitad de sesión.

// la distinción que arregla los dashboards

Test Status (Workflow) ≠ Execution Status (Run)

Test Status — el ciclo de vida

"¿Dónde está el TC en su vida de documentación/automatización?" Persiste entre corridas.

`Draft` `In Design` `Ready` `In Review` `Candidate` `In Automation` `Pull Request` `Automated` `Manual` `Deprecated`

Execution Status — el resultado de la corrida

"¿Pasó la última vez que se corrió?" Resetea cada ejecución.

`TODO` `EXECUTING` `PASS` `FAIL` `BLOCKED` `ABORTED`

Son dos campos independientes

`Automated` (workflow) + `FAIL` (última corrida) es una combinación válida y común: el TC está vivo en CI, pero hoy falló. El rollup PASSED/FAILED del ATR viene del **Execution Status**, no del Test Status. Confundirlos es la causa #1 de dashboards y JQL malos.

// el state machine del TC

Del Draft al Automated — Sin Saltos

Draft → In Design → Ready → In Review → Candidate → In Automation → Pull Request → Automated ✓

ramas terminales: Ready → Manual | cualquiera → Deprecated

No saltar

Draft no salta a Automated.

Back limitado

Solo "back to In Design" y "→ Deprecated".

Manual no es final

Manual puede volver a In Review si el ROI cambia.

Automated es la meta

Mueve cuantos razonables a Automated.

// cómo cuelgan los artefactos

La Cadena US → ATP → ATR → TC

modelo de trazabilidad (jira-xray)

```
User Story (PROJ-123)
   ├── is tested by ──→ ATP (ATP: PROJ-123: Apply discount at checkout)
   └── is tested by ──→ ATR (ATR: PROJ-123: Story Testing)
            ATP designs          ATR executes
                  \                /
                   ▼              ▼
                  [ TC-1, TC-2, … TC-N ]   # cada uno cubre un AC
# El Story enlaza SOLO a ATP y ATR. Los TCs agregan vía ATP/ATR — nunca directo.
```

jira-xray — indirecto

El TC **nunca** se enlaza al Story directo. ATP `designs` TC, ATR `executes` TC. Evita ruido de links. Crea ATP y ATR **antes** del primer TC.

jira-native — directo

Sin issues ATP/ATR, el TC **sí** se enlaza al Story vía `is tested by` — es la única arista de trazabilidad disponible.

Deriva Amplio. Documenta lo Repetible. Automatiza los Pocos.

Scope por el input → título de 4 segmentos + un (precond, acción) por TC → Gherkin con técnicas y variables → ROI = Valor/Costo sesgado por Riesgo → el TMS correcto, con o sin Xray.

← → navegar · S notas del orador · O resumen · T tema