> how-it-works · agentic-qa · shift-left-testing

El Arte del Shift-Left Testing

Prevenir el defecto en los requisitos, no detectarlo tras la implementación. Cómo QA refina los Acceptance Criteria **antes** del sprint — y por qué todo proyecto lo necesita.

Seleccionar → Refinar → Entregar

Stage 0 · QA entra en el backlog, no al final

// la economía del defecto

Cuesta **1×** arreglar un defecto en los **requisitos**, **~10×** en **desarrollo**, y **~100×** en **producción**. Shift-left empuja la calidad hacia donde el defecto es más barato de matar: antes de escribir una sola línea.

// qué cubriremos

El Porqué y el Cómo

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

// por qué · la tesis

Prevenir, no Detectar

La QA tradicional **detecta** defectos después de construir. Shift-left los **previene** en el lugar donde nacen: los requisitos. Un AC ambiguo es un defecto esperando a ocurrir.

❌ QA al final (detectar)

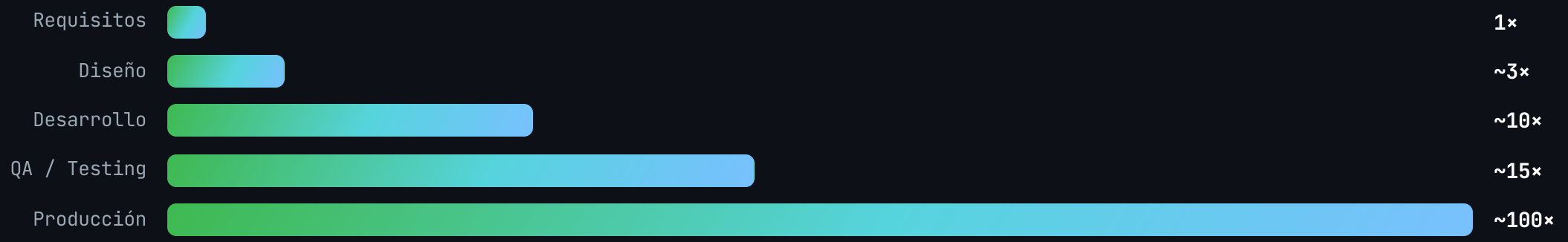
- AC ambiguo → dev construye lo equivocado
- Bug se descubre en QA o en producción
- Retrabajo: re-diseño + re-código + re-test
- El defecto ya costó 10× o 100×

✓ QA al inicio (prevenir)

- QA refina el AC **antes** del sprint
- Ambigüedad resuelta como pregunta a PO
- Dev construye lo correcto a la primera
- El defecto nunca existió — costo 1×

// por qué · la economía

La Curva de Costo del Defecto



~10× por fase (Boehm · IBM SSI)

El costo de corregir un defecto crece de forma **exponencial** con cada fase que cruza sin detectarse. Shift-left invierte la apuesta: gasta horas de refinamiento en la fase 1× para no gastar semanas en la fase 100×.

// por qué · en cada proyecto

Qué Previene Shift-Left — en Todo Proyecto

Ambigüedad → retrabajo

Un AC que admite dos lecturas → dev elige la equivocada → re-diseño. Resuelto como pregunta a PO antes de estimar.

Hueco → defecto en prod

Un camino negativo sin AC → nadie lo construye ni lo prueba → falla en producción. Surgido como gap antes del sprint.

AC no testeable → falso positivo

"Muestra error" sin texto exacto → QA escribe un assert que pasa con el error equivocado. Vuelto medible antes de codear.

Y un beneficio extra: destapar el trabajo mal etiquetado

Al filtrar candidatos, shift-left revela que lo que parecía una Story es en realidad un Bug o Tech-debt mal clasificado. **Surfacing es el valor** — incluso una Story limpia sale con lista de preguntas vacía, y eso también es un resultado válido.

PARTE 01

El Pipeline
Stage 0 — Antes del Sprint

Tres fases, siempre en orden: Selección → Refinamiento → Handoff. Batch por diseño: una sesión refina N Stories para un solo pase de grooming con el equipo.

// tres fases en orden

Selección → Refinamiento → Handoff

1

Selección

Filtra candidatos del backlog (solo Stories), aplica veto + risk score, presenta tabla rankeada. WAIT por OK del usuario.

2

Refinamiento (el corazón)

Por Story: ambigüedades + huecos + edge cases no escritos + testabilidad. Refina ACs (Given/When/Then) y bosqueja outlines. Sin ejecutar.

3

Handoff

Escribe a Jira: ACs refinados, ATP DRAFT, preguntas a PO/Dev, labels, transición backlog → estimation. Batch report.

Batch por diseño

No hay modo single-ticket: una sesión refina N Stories (sweet spot **1-12**) para que PO + Dev lead corran un solo pase de grooming con el equipo. >12 → divide en sesiones.

// por qué una disciplina aparte

Shift-Left vs Sprint-Testing

EJE	/SHIFT-LEFT-TESTING	/SPRINT-TESTING
Cadencia	Pre-sprint, batch de N Stories	In-sprint, ticket por ticket
Entry status	Backlog / Shift-Left QA / Estimation	Ready For QA
Exit status	Estimation (refinado, sin estimar)	QA Approved (ejecutado)
Audiencia	PO / BA + tester	Dev + tester
Output	ACs refinados + risk map + ATP DRAFT	ATP + ATR + bugs + evidencia
Ejecución	**Ninguna** — la feature no existe aún	Smoke + UI/API/DB

Reutilización deliberada

~70% de la lógica de refinamiento ya vive en `acceptance-test-planning.md`. Shift-left la cita, no la duplica — y el trabajo hecho aquí se **reutiliza** in-sprint (no se repite).

// qué NO hace

Los Límites del Scope

- **Solo Stories** — Bugs, Spikes, Sub-tasks, Tech-debt: rechazados. Un bug es reactivo, no tiene ACs upstream que refinar.
- **Refina, no ejecuta** — sin smoke, sin DB queries, sin API calls. La feasibility se establece **leyendo** código + APIs + schema.
- **Para en Estimation** — nunca avanza más allá. PO + Dev lead son dueños de `estimate → ready_for_dev`.
- **Sin código, sin commit** — Jira es canónico. El `shift-left-refinement.md` es working artifact gitignored.
- **Sin test data, sin parametrización** — outline names only. Eso se difiere a in-sprint.
- **Sin TCs** — los Test Cases se formalizan en Stage 4 (`/test-documentation`), no aquí.

La línea con sprint-testing

Todo lo que requiere correr el sistema (generación de datos, parametrización, fixtures) pertenece a `/sprint-testing` Stage 1. Shift-left es **doc-only**.

PARTE 02

Selección
Qué Story Refinar

No toda Story merece refinamiento. El veto descarta lo trivial; el risk score gradúa la profundidad. El objetivo: gastar el presupuesto de atención donde hay riesgo real.

// qué califica

Candidatos del Backlog

JQL · candidatos

```
project = {{PROJECT_KEY}} AND issueType = Story
AND status in (Backlog, "Shift-Left QA", Estimation, "Ready For Dev")
ORDER BY priority DESC, created DESC

# filtros opcionales:
AND sprint in openSprints()      # candidatos del próximo sprint
AND labels != "shift-left-reviewed"  # (pero PREFIERE surfacearlos)
```

Filtro de tipo (duro)

Solo **Story**. Bug/Spike/Sub-task/Tech-debt → rechazo, con razón de una línea. Nunca drop silencioso.

Freshness de label

`shift-left-reviewed` <30 días → ya refinada (surfacear: ¿refrescar?). >30 días → re-refinar (pudo derivar).

// el veto vence al score

Veto — Saltar vs Forzar

SKIP (descartar)

- Backend-only sin UI ni cambio de contrato
- CSS / estilo puro
- Copy estático / documentación
- Tech-debt sin cambio de comportamiento
- DB-only sin lógica de negocio

vs

FORCE FULL (forzar)

- Dinero / billing / refund / conversión
- Integridad de datos en entidades core
- Auth / authorization / sesión / RBAC
- Integraciones externas (Stripe, Auth0...)
- Máquinas de estado · cálculos / fórmulas

El veto vence al risk score

Una Story de dinero o auth se refina **completa** sin importar su puntaje. El riesgo de un cálculo mal o un bypass de auth no se negocia con un número.

// si no hay veto: gradúa la profundidad

Risk Score → Profundidad

Factores (suman)

- **+3** feature nueva · **+3** data dinámica (API/DB)
- **+2** ACs explícitos · **+2** user-facing · **+2** alto esfuerzo
- **+2** AC-light (<2 ACs o de 1 línea) → más refinamiento
- **+1** prioridad alta · multi-componente · dep externa
- **+1** >5 comentarios → desacuerdo sin resolver

Veredicto

0-3 LOW

SKIP — PO/Dev escriben ACs sin QA

4-7 MEDIUM

Refinamiento estándar

8+ HIGH

Extendido + scan de edge cases + data feasibility

// quiz · selección

¿Refinar o Saltar?

Q1 Una Story de cálculo de impuestos tiene risk score 2 (LOW). ¿Qué se hace?

- **A** FORCE FULL — el veto de "cálculos/fórmulas" vence al score bajo
- **B** SKIP — score LOW, PO escribe los ACs
- **C** Refinamiento estándar — score intermedio

FORCE FULL. El veto vence al risk score: dinero, datos, auth, integraciones, máquinas de estado y **cálculos/fórmulas** se refinan completos sin importar el puntaje. La precisión de un impuesto no se negocia.

Q2 El usuario pega 5 IDs para refinar; uno es un Bug de "el botón no responde". ¿Qué pasa con el Bug?

- **A** Se refina igual — es trabajo de QA
- **B** Se rechaza con razón de una línea — shift-left es solo Stories
- **C** Se convierte en Story automáticamente

Se rechaza (visible). Filtro de tipo duro: solo Stories. Un bug es reactivo — no tiene ACs upstream que refinar. Se surfacea con razón, nunca drop silencioso. Surfacear el mal etiquetado ES parte del valor.

PARTE 03

Refinamiento
El Corazón del Shift-Left

Aquí está el valor: destapar las ambigüedades, los huecos y los edge cases que la Story calla. Los ACs son el piso — el refinamiento empuja más allá del happy path.

// cómo refinar bien

Las Cinco Sub-Análisis del Corazón

1 · Ambigüedades

Texto que admite varias lecturas → ubicación + pregunta a PO + impacto en testing + clarificación sugerida.

2 · Gaps (info faltante)

Tipo (AC / detalle técnico / regla de negocio) + por qué crítico + qué añadir + riesgo si se omite.

3 · Edge cases no escritos

Escenario + comportamiento esperado (mejor estimación) + criticidad + acción. Marcado `NEEDS PO/DEV CONFIRMATION`.

4 · Testabilidad

Yes / Partial / No + lista de issues (AC vago, sin texto de error, sin datos de ejemplo, sin criterio de perf, no aislable).

5 · Contradicciones

Secciones de la Story que se contradicen. Surfacear explícito: *"La descripción dice X pero el AC3 dice Y"*.

// qué hace bueno a un AC

Testabilidad — los Modos de Falla

AC NO testeable

- **Vago** — "funciona rápido" (sin métrica)
- **Sin texto de error** — "muestra error"
- **Sin datos de ejemplo**
- **Sin criterio de performance**
- **No aislable** — depende de otra feature

AC testeable

- **Medible** — "responde en < 2 segundos"
- **Error exacto** — texto + status code
- **Con datos concretos**
- **Con budget de perf (p95)**
- **Aislable** — verificable solo

El costo de un AC vago

"Muestra error" sin texto exacto → QA escribe un assert de "muestra *algún* error" que **pasa con el error equivocado**. Falso positivo en QA + defecto en producción. Por eso la testabilidad se valida ANTES de codear.

// los ACs son el piso, no el techo

Explota 1:N — Empuja Más Allá del Happy Path

Un AC no-trivial implica **varios** escenarios. El refinamiento deriva por la forma del AC, y etiqueta cada hueco a su técnica.

SI EL AC TIENE…	TÉCNICA	DERIVA
un rango / límite	**Boundary Value Analysis**	min-1·min·min+1 … max-1·max·max+1
un campo de estado	**State-Transition**	1 por transición válida + 1 por inválida
2+ condiciones que interactúan	**Decision Table**	1 por regla sobreviviente
3+ factores combinables	**Pairwise**	set all-pairs (log de la reducción)

Colapsar a 1 escenario exige justificación

"Trivialmente atómico" (un booleano, sin rangos/estados/interacciones) — y se dice por escrito. Un AC de 1 línea que **esconde** un rango o un estado NO es un AC de 1 escenario.

// la herramienta del refinador

El Banco de Preguntas — Rúbrica, no Guion

1

Identifica el arquetipo

Auth · Money · Search · State Machine · Integration · CRUD · Notification · RBAC · Performance · a11y/i18n. Cada Story toca 2-3.

2

Recorre la lista del arquetipo

Cada pregunta sin respuesta explícita en la Story = candidato a gap.

3

Clasifica el gap

Critical → bloquea sprint planning (pregunta a PO) · **Important** → bloquea implementación (a Dev) · **Edge** → testeable, no bloquea.

Un gap no es solo una pregunta — es también un outline faltante

Y cada arquetipo mapea a su técnica: State Machine → State-Transition; Money/ranges → BVA; RBAC → Decision Table; configs multi-factor → Pairwise.

// las que casi siempre faltan

Preguntas Universales — el Núcleo

#	PREGUNTA	SI NO SE RESPONDE...
U1	¿Texto EXACTO del error + status code por cada camino de fallo?	QA escribe assert que pasa con el error equivocado
U3	¿Qué pasa en éxito — redirect, toast, reload?	El test pasa por "sin error" pero la UI nunca actualizó
U5	¿La operación es idempotente? ¿Y si se dispara dos veces?	Cargos / registros duplicados
U7	¿Rollback en fallo parcial (3 de 5 registros fallan)?	Mutaciones a medias dejan el sistema inconsistente
U10	¿Hay estado "primera vez" / vacío distinto del estado normal?	El onboarding muestra una tabla vacía aterradora

Pero solo si la Story no las responde ya

Las universales NO se incluyen por defecto — se recorren cuando la Story las deja sin responder. El archivo de refinamiento es de alta señal, no exhaustivo.

// las preguntas asesinas por dominio

Por Arquetipo — Donde Viven los Defectos

Auth / Sesión

- A1 · ¿Rol/permiso por cada AC?
- A8 · ¿No-autenticado → 401, 403 o 404? (403 vs 404 revela existencia)
- A10 · ¿Rate-limit + lockout por intentos?

Money / Billing

- M1 · ¿Moneda, precisión, redondeo?
- M3 · ¿El provider de pago hace timeout — retry, error, cola?
- M9 · ¿Proración al cambiar/cancelar plan?

State Machine

- SM1 · ¿TODOS los estados + transiciones válidas?
- SM3 · ¿Transición inválida (cancelar un delivered)?
- SM10 · ¿Transiciones concurrentes — lock/cola?

El catálogo cubre 10 arquetipos

Auth · Money · Search/List · State Machine · Integration · CRUD · Notification · RBAC · Performance · a11y/i18n — cada uno con ~10 preguntas asesinas y su modo de falla.

// de gap a AC accionable

ACs Refinados — Given / When / Then

Cada escenario refinado lleva Tipo (Positive/Negative/Boundary/Edge) + Priority + Given/When/Then con valores exactos: UI + API (status+body) + DB + estado del sistema.

outlines derivados — "Should <behavior> <condition>"

```
Should redirect to dashboard after successful OTP entry with valid code
Should display "Code expired" error after OTP entry with code older than 5 min
Should reject negative refund amount on POST /refunds
# inferido, no literal en la Story:
Should show "Session expired" when OTP submitted after session expiry    NEEDS PO/DEV CONFIRMATION
```

El marcador que sobrevive a Jira

Todo escenario que el refinamiento **infirió** (no estaba literal en la Story) lleva `NEEDS PO/DEV CONFIRMATION` — verbatim, en inglés. El marcador viaja al comentario de Jira para que PO lo vea en sprint planning.

// calidad > cantidad

El Arte de NO Preguntar

El recurso más escaso en un grooming es la atención del equipo. Una pregunta redundante entrena al equipo a ignorar el output futuro de QA.

- **Story clara → dilo**. "Calidad de la Story: buena, sin gaps significativos" supera a inventar preguntas. Lista vacía es un resultado válido.
- **Rúbrica, no guion**. Si la Story ya es explícita en un tema, no lo re-preguntes.
- **Cita el contexto en vez de preguntar**. Si `business-data-map.md` ya define la moneda o la máquina de estados, cítalo. PO tiene paciencia finita.
- **Clasifica con juicio**. Meter una pregunta de texto-de-éxito junto a una de bypass de auth señala falta de criterio.

> **Anti-patrón L1 (el más importante)**
>
> NUNCA fuerces preguntas para llenar un checklist. Shift-left aporta valor **destapando riesgo real**, no inflando conteos de preguntas.

// quiz · refinamiento

El Corazón del Refinamiento

Q1 El refinamiento infiere un edge case que no está en la Story original. ¿Cómo se marca?

A Se añade al AC directamente como hecho

B Se marca `NEEDS PO/DEV CONFIRMATION` — verbatim, viaja a Jira

C Se descarta si no está en la Story

NEEDS PO/DEV CONFIRMATION. Todo lo inferido (no literal en la Story) lleva ese marcador exacto. Viaja al comentario de Jira para que PO lo confirme en sprint planning. El tooling después lo grepea — por eso es verbatim.

Q2 Tras analizarla, la Story está genuinamente clara y completa. ¿Qué entregas?

A Inventas algunas preguntas para que el refinamiento "se vea completo"

B "Calidad de la Story: buena, sin gaps" — lista de preguntas vacía

C La rechazas porque no hay nada que refinar

Lista vacía. Una Story limpia sale con cero preguntas y eso es válido (anti-patrón L1: nunca fuerces preguntas). Calidad > cantidad: preguntas redundantes entrenan al equipo a ignorar a QA.

PARTE 04

Artefactos + Handoff
Entregar a PO y Dev

El ATP DRAFT hace estimable la Story. El handoff la mueve a Estimation con labels que dejan que el sprint reutilice el trabajo — no lo repita.

// el artefacto central

El ATP DRAFT — `shift-left-refinement.md`

Secciones

- Phase 1 — Análisis Crítico (contexto + complejidad)
- Phase 2 — Calidad (ambigüedades, gaps, edge, testabilidad)
- Phase 3 — ACs Refinados (Given/When/Then)
- Phase 4 — Outlines (NOMBRES) + estimación de cobertura
- Critical Questions PO · Tech Questions Dev
- Suggested Improvements · Data feasibility

DRAFT, no ATP completo

- NO tablas de parametrización
- NO test-data JSON por outline
- NO pasos numerados ni Faker
- NO creación de TCs
- SÍ estimación de cobertura (para PO)
- SÍ outline names + precondición de 1 línea

Status: "Refined — Awaiting PO Estimation"

El DRAFT es lo que hace la Story **estimable**. Lo diferido (datos, parametrización, pasos) lo añade `/sprint-testing` Stage 1 como un **superset** de este archivo.

// el dato que estima PO

Estimación de Cobertura

TIPO	COUNT	NOTAS
Positive	4	variantes del happy path
Negative	5	inputs inválidos, no autorizado, campos faltantes
Boundary	2	min/max/empty/null/unicode
Integration	1	por punto de integración
Total	**12**	(maneja la estimación de PO)

Muestra el 0, no lo escondas

La tabla debe mostrar **0** en los tipos vacíos. Los ceros ocultos sesgan la estimación de PO. El conteo por tipo es lo que convierte el refinamiento en story points defendibles.

// la máquina de estados

El Handoff — Backlog → Estimation

Backlog — analyze → Shift-Left QA — estimate → Estimation ✓ ⊘ Ready For Dev

Escribe a Jira

ACs refinados al campo · ATP DRAFT al custom field · sección "QA Refinements" en la descripción (append, nunca overwrite).

Notifica

1 comentario de handoff con @PO / @Dev (solo si el handle está en project.yaml).

Transiciona

analyze → estimate. Si ya pasó estimation: log warning + SKIP (el refinamiento igual aterriza).

Guardrail: NUNCA más allá de Estimation

PO + Dev lead son dueños de `estimate → ready_for_dev`. Shift-left para en estimation — la Story queda lista para que el equipo le ponga puntos.

// trabajo hecho una vez, reutilizado

Los Labels y el Short-Circuit

`shift-left-reviewed`

El **marcador suave**. `/sprint-testing` Stage 1 lo lee para saber que el refinamiento pre-sprint ya ocurrió.

`shift-left-{YYYY-MM-DD}`

El **marcador de frescura**. Si <30 días, el refinamiento se considera vigente y se puede short-circuitar.

→

Cuando la Story llega a Ready For QA...

`/sprint-testing` detecta el label, **salta las Phases 1-3** (ya hechas), valida que los ACs sigan vigentes, y añade lo diferido (datos, parametrización, pasos) como un superset.

Esta es la economía del shift-left

El refinamiento se hace **una vez**, pre-sprint, y se **reutiliza** in-sprint — no se repite. El costo de calidad se paga en la fase 1×, no en la 10×.

// qué sale de la sesión

Entregables — por Story y por Batch

Por Story

- `shift-left-refinement.md` (working file)
- ACs refinados al campo de Jira
- ATP DRAFT al custom field
- Preguntas a PO (bloquean) + a Dev
- 2 labels + transición a estimation

Por Batch (sesión)

- `batch-report.md`
- Preguntas críticas a PO **dedupeadas** (rankeadas por cuántas Stories bloquean)
- Distribución de riesgo (LOW/MED/HIGH)
- Blockers de feasibility de datos
- Orden recomendado de sprint planning

El batch report va al épico padre

Si todas las Stories comparten un épico, el reporte se postea como comentario ahí. Nunca en cada Story — eso es ruido.

// quiz · handoff

Artefactos & Handoff

Q1 Una Story refinada está en estado Backlog. ¿Hasta dónde la transiciona el handoff?

A Hasta Ready For Dev — lista para el sprint

B Hasta Estimation (analyze → estimate) y para ahí — guardrail

C Hasta QA Approved

Estimation. backlog → (analyze) → shift_left_qa → (estimate) → estimation. Guardrail duro: nunca más allá. PO + Dev lead son dueños de estimate → ready_for_dev; ellos le ponen los puntos.

Q2 ¿Para qué sirve el label `shift-left-reviewed` cuando la Story llega luego a Ready For QA?

A Es decorativo — solo marca que QA la vio

B Deja que /sprint-testing salte las Phases 1-3 — el trabajo se reutiliza, no se repite

C Bloquea que la Story entre al sprint

Short-circuit. /sprint-testing Stage 1 lo detecta y salta las Phases 1-3 de planning (ya hechas pre-sprint). Esa reutilización es la economía del shift-left: refinas una vez en la fase 1×, no repites en la 10×.

// nunca hagas esto

Anti-Patrones del Shift-Left

- **L1 · No fuerces preguntas** para llenar checklist. Story limpia = lista vacía válida.
- **L2 · No saltes el label** `shift-left-reviewed` al transicionar — rompe el short-circuit.
- **L3 · No mezcles bugs** con Stories en el batch. Solo Stories.
- **L5 · No transiciones a Estimation** sin un ATP DRAFT poblado — sin él, PO estima a ciegas.
- **L4 · No escribas ADF a mano** — Markdown local, deja que la conversión md-to-ADF actúe.
- **L6 · No refines >12 Stories** por sesión — la calidad y la atención colapsan.
- **L7 · No preguntes lo que el AC ya responde** — la banda del lector es el recurso más escaso.
- **Guardrail · No avances más allá de Estimation** — PO/Dev son dueños de estimar.

// el checklist del refinador

Cómo Hacerlo Bien

- **Veto antes que score** — dinero/auth/datos/integraciones se fuerzan a full.
- **Recorre arquetipos**, no preguntes al azar — cada Story toca 2-3.
- **Cita el contexto** en vez de preguntar lo ya documentado.
- **Etiqueta cada gap a su técnica** (BVA/State/Decision/Pairwise).
- **Marca lo inferido** con `NEEDS PO/DEV CONFIRMATION`.
- **Clasifica con juicio** — Critical (bloquea PO) vs Edge (no bloquea).
- **Incluye la estimación de cobertura** — alimenta los story points.
- **Calidad > cantidad** — una pregunta real vale más que diez de relleno.

La meta

Que PO + Dev lleguen al sprint planning con ACs sin ambigüedad, riesgos visibles y una Story estimable — para que Dev construya lo correcto a la primera.

// por qué en TODO proyecto

Por Qué Cada Proyecto lo Necesita

1×

costo en requisitos

la fase más barata para matar un defecto — donde shift-left actúa

~70%

trabajo reutilizado

la lógica de refinamiento se reusa in-sprint, no se repite

0

preguntas forzadas

solo riesgo real — una Story limpia sale limpia

No es solo para proyectos grandes

Cualquier proyecto con requisitos tiene ambigüedades, huecos y ACs no testeables. Shift-left los convierte en preguntas baratas **antes** de que sean defectos caros. El ROI es estructural: horas de refinamiento vs semanas de retrabajo.

// para llevar

Tres Ideas que se Quedan

1

Prevenir, no detectar

QA entra en los requisitos, no al final. Un AC ambiguo es un defecto esperando. Costo 1× vs 100×.

2

El corazón es el refinamiento

Ambigüedades, huecos, edge cases, testabilidad. El banco de preguntas como rúbrica — calidad > cantidad.

3

Entrega que se reutiliza

ATP DRAFT estimable + labels que dejan al sprint saltar el trabajo ya hecho. Una vez, no dos.

Mueve la Calidad a la Izquierda.

Selecciona por riesgo → refina el corazón (ambigüedades, huecos, el banco de preguntas) → entrega un ATP DRAFT estimable que el sprint reutiliza. Previene el defecto donde cuesta 1×, no donde cuesta 100×.

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