> how-it-works · agentic-qa · defect-management

Defect Management — Gestionar la Calidad por sus Incidencias

Bug, Defect, Improvement: el mismo síntoma, distinta estrategia de gestión. Cómo se reportan bajo un estándar de calidad reusable, cómo viven en su workflow, y cómo se convierten en métricas que responden preguntas de calidad.

Del reporte a la métrica

parte 1: las 3 incidencias · parte 2: workflow + campos del estándar · parte 3: medir (ejemplo ShopFlow)

// el dato que responde

40 / 40

En ShopFlow, las **40 incidencias** tienen `Components` — y cada campo del estándar — relleno. La data no solo existe: **responde cada pregunta de calidad**. Lo que rellenas con disciplina hoy es lo que el dashboard sabe medir mañana.

// qué cubriremos

Tres Partes: Distinguir, Gestionar, Medir

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

PARTE 01

Las Tres Incidencias
Bug · Defect · Improvement

El término no las distingue — léxicamente un bug ES un defecto. Lo que las distingue es **cuándo nacen** y **qué le dicen al equipo** sobre el estado del producto.

// la idea central

Mismo Término, Distinta Gestión

"Bug" y "defecto" son sinónimos en el lenguaje. La diferencia vive en la **línea de tiempo del producto**: ¿la funcionalidad sigue en desarrollo, o ya fue liberada?

El eje que decide todo

Una falla detectada **durante el sprint**, sobre una funcionalidad que aún se está construyendo → **Defect**. La misma clase de falla, descubierta **después del release** sobre algo que ya funcionaba → **Bug**. El Improvement es el tercer camino: ni una ni otra, sino una mejora.

// incidencia #1

Defect — la falla que cazamos en el sprint

Qué indica

Que una **User Story bajo sprint** está saliendo defectuosa y hay que arreglarla **antes** de que suba a ambientes superiores.

Cuándo nace

Durante el desarrollo / la fase de QA del sprint. Es lo **normal y deseable**: el sprint existe para destapar defectos y cerrarlos.

- **Nace de una User Story.** Se enlaza a la US que se está construyendo (edge causation: la US `causes` el Defect).
- **Es contención, no escape.** Encontrarlo aquí es éxito de QA — el defecto nunca llegó al usuario.
- **Si el sprint hace bien su trabajo**, la US entra al release **libre de defectos**.

// incidencia #2

Bug — la regresión que escapó

Qué indica

Que una funcionalidad **ya liberada y desplegada** está fallando. Algo que **se suponía debía funcionar** dejó de hacerlo.

Cuándo nace

Pasado el sprint — semanas o meses después — típicamente como un **edge case** o una **regresión** sobre código ya validado.

- **El Bug es el tipo de incidencia de las regresiones:** cada vez que algo que debería funcionar, falla.
- **Es un escape.** Cruzó la red del sprint y llegó a producción — eso es justo lo que las métricas deben vigilar.
- **Mismo síntoma que un Defect**, distinto momento → distinta lectura de gestión.

// el mismo fallo, dos vidas

Caso: el Carrito de Compra

1

Sprint — nueva US del carrito

Se agrega una historia de usuario sobre el carrito. QA detecta que el cálculo del total falla. `DEFECT` Está en desarrollo, se enlaza a la US, se arregla dentro del sprint.

2

Release — la US sale limpia

El defecto se reparó. La funcionalidad del carrito entra al release **libre de defectos**. El usuario la recibe funcionando.

3

Meses después — un edge case

Aparece un caso límite alrededor del mismo carrito (cupón + moneda extranjera + stock 0). Falla en producción. `BUG` Es una regresión descubierta post-sprint.

// incidencia #3

Improvement — ni falla, sino mejora

Un work type de tipo **mejora**. A veces trivial; a veces el destino de un Bug o Defect que, tras análisis de causa raíz, resulta no ser una falla sino un asunto de requerimientos.

Origen directo

Una mejora pequeña detectada al usar el producto: copy, UX, un atajo. No rompe nada — eleva la calidad.

Origen por reclasificación

Un Bug/Defect pasa por **RCA** y se descubre que el "error" era en realidad el comportamiento especificado. Se transforma: a Improvement, o a una nueva User Story.

En el workflow esto es literal

La transición `is not a Bug` lleva un Bug/Defect desde `Open` al estado terminal `Enhancement` — la reclasificación está cableada en el flujo del estándar, no es informal.

// las tres, lado a lado

Defect vs Bug vs Improvement

DIMENSIÓN	DEFECT	BUG	IMPROVEMENT
Momento	Durante el sprint	Post-release	Cualquiera
Estado del producto	En desarrollo	Ya liberado	Funcional
Qué señala	US saliendo defectuosa	Regresión / escape	Mejora o reclasificación
Lectura de gestión	Contención (éxito de QA)	Fuga (riesgo a usuario)	Valor incremental / RCA
Nace de	Una User Story	Un edge case en prod	Uso real o un RCA

Léelo como instrumentación, no como burocracia

Elegir bien el work type no es papeleo: es lo que más adelante hace que la pregunta *"¿estamos deteniendo los defectos antes de que escapen?"* sea **medible**.

// quiz · parte 1

¿Bug, Defect o Improvement?

Q1 Durante el sprint, QA encuentra que el nuevo formulario de registro no valida el email. La US aún está en desarrollo.

- **A** Defect — falla detectada en sprint, sobre una US en desarrollo
- B Bug — es una falla funcional
- C Improvement — el formulario podría mejorar

Defect. El criterio no es la gravedad sino el momento: la funcionalidad sigue en construcción dentro del sprint. Cazarlo aquí es contención, no fuga.

Q2 Tres meses tras el release, un usuario reporta que el login falla solo si la contraseña tiene emojis. Antes funcionaba.

- A Defect — hay que arreglarlo
- **B** Bug — regresión / edge case sobre algo ya liberado
- C Improvement — soportar emojis es una mejora

Bug. Funcionalidad ya desplegada que deja de funcionar = regresión. Es un escape: cruzó la red del sprint y llegó al usuario.

PARTE 02

El Estándar de Calidad
Un Workflow, los Mismos Campos

Las tres incidencias comparten **exactamente** el mismo esquema de workflow y los mismos campos. Reportar disciplinadamente hoy es lo que habilita medir mañana.

// hallazgo del esquema real

Bug, Defect e Improvement → Mismo Esquema

Bug #10021

11 estados · 15 transiciones

Defect #10023

11 estados · 15 transiciones

Improvement #10022

11 estados · 15 transiciones

Idénticos byte por byte

Mismos estados (Open, In Progress, In Review, Ready For QA, Closed…), mismas transiciones, mismos campos para rellenar. Cambias el *work type*, no la mecánica. Esto es deliberado: un solo modelo mental para gestionar las tres.

// el flujo feliz

El Ciclo de Vida de una Incidencia

Open → In Progress → In Review → Ready For QA → Closed ✓

start fixing

Open → In Progress. El dev toma la incidencia.

Pull Request / Fixed & Deployed

In Progress → In Review → Ready For QA. El fix llega a QA para retest.

ReTest Passed

Ready For QA → Closed. QA verifica el fix (terminal de éxito).

Atajos y regresos cableados

`Hard pushed` salta el review (In Progress → Ready For QA). `back` reabre un Closed a Ready For QA cuando una regresión se cuela tras el sign-off.

// la decisión más importante

Triage desde Open — 6 Caminos

Open es el hub de triage. No todo reporte se arregla — la mitad del valor de gestión está en clasificar bien lo que no es un fix.

`start fixing → In Progress`

Es real. El dev lo toma y arranca el fix.

`is duplicated → Duplicated`

Repetido. Ya existe otro ticket. Terminal.

`is WAD → REJECTED`

Working As Designed. No es falla. Terminal.

`is CNR → Cannot Reproduce`

No reproducible. Sin repro, no hay fix. Terminal.

`is not a Bug → Enhancement`

Reclasificación. Era requerimiento → mejora. Terminal.

`defer → Deferred`

Pospuesto. Aceptado pero no ahora. Reanudable.

// cómo termina una incidencia

Estados Terminales y sus Categorías

ESTADO	CATEGORÍA	SIGNIFICA	LECTURA DE MÉTRICA
Closed	done	Fix verificado por QA	Trabajo resuelto — el numerador del progreso
Duplicated	done	Repetido de otro ticket	Ruido de reporte
REJECTED	done	Working As Designed	Falso positivo
Cannot Reproduce	done	Sin repro confirmable	Reporte incompleto
Enhancement	done	Reclasificado a mejora	Asunto de requerimiento
Deferred	indeterminate	Pospuesto, reanudable	Deuda viva (aging)
ABORTED	done	Abandonado sin fix	Descartado

// qué se rellena al reportar

Los Campos del Estándar de Calidad

Siempre (alta señal)

- **Summary** — `Épica: Componente: Síntoma`
- **Steps to Reproduce** — pasos numerados
- **Actual** vs **Expected Result**
- **Severity** · **Priority**
- **Error Type** · **Test Environment**
- **Evidence** — captura / trace / log

Después / contexto

- **Root Cause** — se fija tras el RCA
- **Fix** — bugfix vs hotfix
- **Workaround** — mitigación temporal
- **Components** — agrupación (¡clave! →)
- **Labels** — `bug`, dominio
- **Links** — US `causes` Bug

Regla de oro

Cada campo que dejas vacío es una pregunta que el dashboard **no podrá responder**. La disciplina al reportar ES la materia prima de la métrica.

// clasificar el impacto

Severity → Priority

SEVERITY	CRITERIO	PRIORITY	EJEMPLO
Crítica	Core bloqueado, sin workaround, pérdida de datos	Highest	Login caído, checkout falla
Mayor	Feature principal rota, workaround difícil	High	Búsqueda devuelve resultados erróneos
Moderada	Issue con workaround fácil	Medium	Orden roto pero el filtro sirve
Menor	Bajo impacto, baja prioridad	Low	Validación de edge case faltante
Trivial	Solo cosmético	Lowest	Typo, leve desalineación

Severity ≠ Priority

Severity es **impacto técnico** (lo fija QA); Priority es **urgencia de negocio** (la fija el equipo). Un typo en el logo (Trivial) puede ser High priority antes de una demo. Sepáralos siempre.

// dos taxonomías que alimentan métricas

Error Type (qué) vs Root Cause (por qué)

Error Type — el síntoma

Se infiere al reportar, desde el comportamiento observado.

`functional` `visual` `content` `performance` `crash` `data` `integration` `security`

Root Cause — la causa

Se fija **después** del fix, en el post-mortem.

`code_error` `config_env_error` `data_error` `environment_error` `integration_error` `requirement_error` `third_party_error` `working_as_designed`

Por qué importan las dos

Error Type responde *"¿qué clase de fallas dominan?"* (¿somos débiles en seguridad? ¿en performance?). Root Cause responde *"¿de dónde vienen?"* (¿código? ¿requerimientos? ¿integraciones?). Dos preguntas, dos campos, dos gráficos.

// el campo subestimado

Components — Agrupar para Medir

Un campo nativo de Jira, configurable para que aparezca en cada incidencia. Su superpoder: **agrupar** Bugs, Defects e Improvements por la parte del producto que afectan.

Flexible según la estrategia del proyecto

Los componentes se definen como convenga: por **features**, por **épicas**, por módulos. ¿20 épicas? → 20 components. Cada incidencia marca el/los componente(s) afectado(s).

Para qué sirve realmente

"¿Qué parte de la app falla más?" se vuelve una sola query. Habilita **ownership** (qué equipo dueño de qué componente) y **routing** automático de incidencias.

Y ShopFlow lo rellena siempre

Las **40 incidencias** de ShopFlow marcan su Component → el dashboard de ownership existe y dice qué módulo concentra los defectos. Regla en una línea: *lo que no rellenas no lo puedes medir* — por eso es la recomendación #1 de higiene, y ShopFlow no deja ninguno vacío. Lo vemos en la parte 3.

Workflow & Campos

Q1 Triage de un Bug recién abierto: el equipo confirma que el comportamiento es exactamente el especificado. ¿Transición y estado?

- A `start fixing` → In Progress
- B `is WAD` → REJECTED
- C `defer` → Deferred

is WAD → REJECTED. Working As Designed: no es una falla, es comportamiento especificado. Estado terminal (categoría done). Si además el cliente quisiera cambiarlo, sería `is not a Bug` → Enhancement.

Q2 El equipo de producto quiere un dashboard de "¿qué parte de la app concentra más defectos?". ¿Qué campo lo habilita?

- A Severity
- B Priority
- C Components

Components. Es el campo de agrupación por parte del producto (features/épicas). En ShopFlow está relleno en las 40 incidencias, así que ese dashboard se construye entero y muestra que Checkout y Auth lideran.

PARTE 03

Medir la Calidad
De Campos a Preguntas Respondidas

Un dashboard no es una lista de datos — es un set de **respuestas**. La meta: que producto vea, con sus propios ojos, cómo va la app. Lo ilustramos con el proyecto de ejemplo ShopFlow.

// el principio rector

Los Datos Solo Importan si Responden Preguntas

❌ Dashboard como vertedero

- Lista de 200 tickets sin agrupar
- Conteos sin contexto ("hay 40 incidencias")
- Métricas que nadie sabe leer
- El equipo de producto no entiende nada

→

✓ Dashboard como respuesta

- "¿Estamos deteniendo defectos antes del escape?"
- "¿Qué componente falla más?"
- "¿La calidad del fix mejora o empeora?"
- Producto abre Jira y **entiende** la salud

// snapshot · proyecto de ejemplo ShopFlow (casa en orden)

Un Snapshot Típico: ShopFlow

40

familia de incidencias

26 Defect · 10 Bug · 4 Improvement — todas tipadas

26

Closed (verificadas)

6 In Progress · 5 Ready For QA · 3 Open

4

Críticas

11 Mayor · 17 Moderada · 8 Menor — 40/40 con Severity

// lo que rellenas, lo puedes medir

La Casa en Orden: Campos al 100%

El diagnóstico

ShopFlow rellena **cada campo del estándar** en las 40 incidencias → las 8 preguntas de la parte 3 tienen respuesta, sin ningún slice "sin asignar". El principio se cumple, pero al revés: *lo que no rellenas no lo puedes medir* — y aquí no quedó nada vacío. La disciplina al reportar HOY es la métrica fiable de MAÑANA.

// el mapa que importa

Pregunta de Calidad → Métrica → Campo

LA PREGUNTA QUE EL EQUIPO SE HACE	MÉTRICA QUE LA RESPONDE	CAMPO(S)
¿Detenemos los defectos antes de que escapen a prod?	**Defect Escape Rate** = Bugs / (Bugs+Defects)	issuetype
¿Qué tan grave es lo que encontramos?	Distribución por **Severity** + Severity Index	Severity
¿Qué parte de la app falla más?	Defectos por **Component**	Components
¿Por qué fallamos? (mejora de proceso)	Distribución por **Root Cause**	Root Cause
¿Qué clase de fallas dominan?	Distribución por **Error Type**	Error Type
¿Qué tan bueno es el fix? (¿reabrimos?)	**Reopen Rate**	historial de status (`back/re-open`)
¿Cuánto tardamos en cerrar?	**Defect Age** / MTTR	created → resolved
¿Cuánto ruido genera el reporte?	**Rejection Ratio** = (Dup+REJ+CNR)/total	resolution / status

// de la pregunta a la query

JQL — el Lenguaje de los Dashboards

JQL · proyecto SHOP (ejemplo)

```
# Todos los defectos críticos aún abiertos
project = SHOP AND issuetype in (Bug, Defect) AND "Severity[Select List]" = Crítica AND statusCategory != Done

# Escape: bugs post-release vs defects in-sprint
project = SHOP AND issuetype = Bug      → 10 (escaparon)
project = SHOP AND issuetype = Defect   → 26 (contenidos)

# Agrupar por componente (Components relleno en las 40)
project = SHOP AND issuetype in (Bug, Defect) ORDER BY component

# Ruido de reporte: descartados
project = SHOP AND status in (Duplicated, REJECTED, "Cannot Reproduce")  → 3
```

// vocabulario de métricas de defectos

Las Métricas que Debes Nombrar

MÉTRICA	QUÉ MIDE	FÓRMULA (EN TÉRMINOS DEL ESTÁNDAR)
Defect Escape Rate	Fugas a producción	Bugs / (Bugs + Defects)
DDP / DRE	Eficiencia de detección	Defects / (Defects + Bugs) — el inverso del escape
Defect Density	Concentración	Defectos por Story / por Component / por módulo
Reopen Rate	Calidad del fix	reabiertos (back) / cerrados
Defect Age / MTTR	Velocidad de cierre	tiempo medio created → Closed
Rejection Ratio	Calidad del reporte	(Duplicated + REJECTED + CNR) / total
Severity Index	Gravedad ponderada	Σ(peso × conteo) / total

// la métrica estrella del shift-left

Defect vs Bug = tu Medidor de Escape

Aquí la taxonomía paga. Como **Defect = en sprint** y **Bug = post-release**, su proporción **es**, literalmente, la salud del shift-left: ¿cuánto contenemos vs cuánto se fuga?

26

Defects (contenidos)

cazados dentro del sprint

10

Bugs (escapados)

descubiertos post-release

28%

Escape Rate

10 / (10+26)

Un 28% es shift-left sano

Como ShopFlow usa el work type Defect con disciplina, la lectura es fiable: **72% de los defectos se contienen en sprint** y solo el 28% escapa a producción. La taxonomía Defect/Bug bien aplicada es justo lo que hace creíble esta métrica.

// el home de producto en jira

El Dashboard que Producto Quiere Ver

Salud general

Gadgets: por **type**, por **status** (lifecycle slice), Escape Rate. `listo en ShopFlow`

Gravedad & causa

Pie por **Severity**, por **Root Cause**, por **Error Type**. `listo en ShopFlow`

Ownership

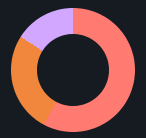
Por **Component** + Reopen trend + Aging. `listo: Components al 100%`

El objetivo final

Una página donde el PO/PM abre Jira y, sin preguntar a nadie, **ve**: cuántas incidencias vivas, qué tan graves, qué parte de la app sufre, si el fix mejora. La gestión de incidencias se vuelve transparente.

// leyenda · antes del dashboard

Cómo Leer Estos Gráficos



Donut / Pie

Proporción de un total; las partes suman 100%. **"¿Cómo se reparte el total?"**

Barras

Comparar la magnitud entre categorías. **"¿Cuál es mayor?"**

Barra apilada

La composición de UN total dentro de una sola barra. **"¿De qué se compone esto?"**

Gauge (medidor)

Un valor único contra su máximo/meta (un %). **"¿Qué tan cerca del tope?"**

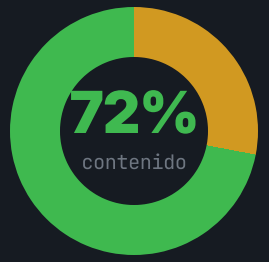
TÉRMINO	QUÉ ES	EJEMPLO EN ESTE DASHBOARD
Rate / Tasa	Proporción expresada de 0 a 100%	Escape Rate = fugados / total
Ratio	Relación entre dos cantidades	Rejection Ratio = ruido / total
Distribución	Cómo se reparte el conteo entre categorías	por Severity, por Root Cause...
Índice ponderado	Σ(peso × conteo) / total — no todo pesa igual	Severity Index: una crítica pesa más que una menor
Slice "Sin asignar"	La porción de datos faltantes; mostrarla = honestidad	si un campo quedara vacío iría aquí (muted) — en ShopFlow no aplica: todo relleno

// dashboard en vivo · bloque 1/3 · ejemplo ShopFlow

Dashboard · Salud General

¿Detenemos los defectos antes de que escapen a prod?

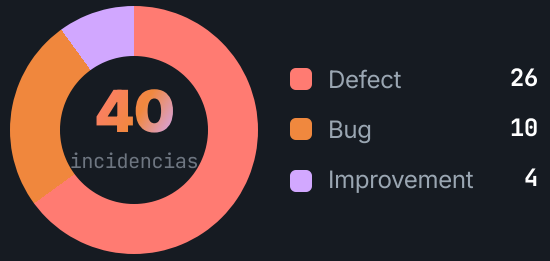
métrica · Defect Escape Rate



Shift-left funciona: 72% se contiene en sprint — Escape Rate 28%, sano.

¿De qué se compone la familia de incidencias?

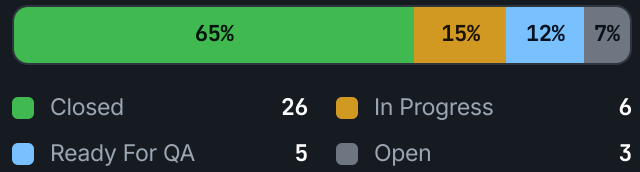
métrica · Type split (40 total)



Defects (26) > Bugs (10): se caza antes de release.

¿En qué punto del ciclo están?

métrica · Status lifecycle (40 total)



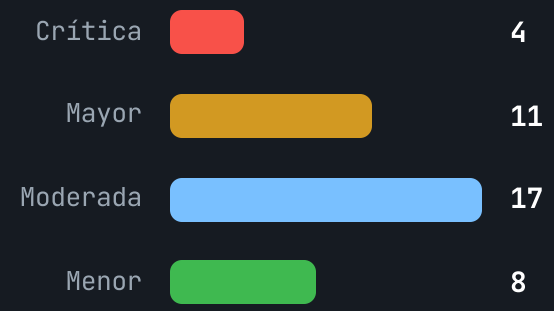
26 cerradas; 14 vivas avanzando por un flujo sano.

// dashboard en vivo · bloque 2/3 · ejemplo ShopFlow

Dashboard · Gravedad & Causa

¿Qué tan grave es lo que encontramos?

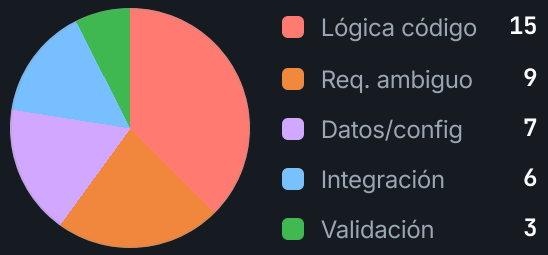
métrica · Severity dist. (40 total)



Mayoría Moderada/Mayor; 4 críticas bajo control.

¿Por qué fallamos?

métrica · Root Cause (40 total)



Lógica de código domina → foco de mejora de proceso.

¿Qué clase de fallas dominan?

métrica · Error Type (40 total)

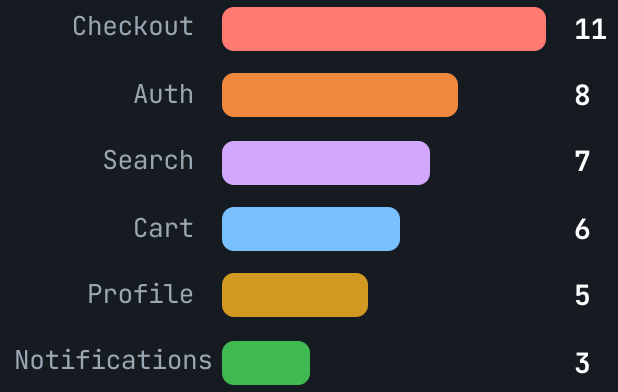
Funcional domina; UI y validación siguen.

// dashboard en vivo · bloque 3/3 · ejemplo ShopFlow

Dashboard · Ownership & Proceso

¿Qué parte de la app falla más?

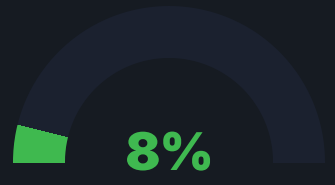
métrica · Defectos por Component (40 total)



Checkout y Auth concentran los defectos → dónde reforzar.

¿Qué tan bueno es el fix? (¿reabrimos?)

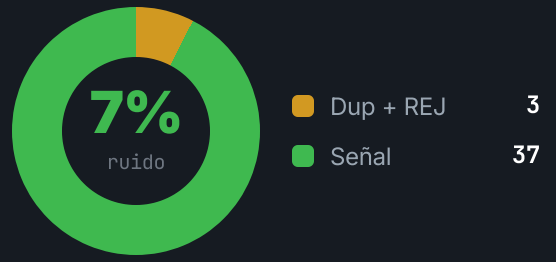
métrica · Reopen Rate



Solo 8% se reabre: fixes sólidos.

¿Cuánto ruido genera el reporte?

métrica · Rejection Ratio (40 total)



7% de ruido: reportes limpios (2 duplicados + 1 rechazado).

// quiz · parte 3

Métricas & Dashboards

Q1 El PM pregunta: "¿estamos atrapando los problemas antes de que lleguen al usuario?". ¿Qué métrica responde?

- **A** Defect Escape Rate = Bugs / (Bugs+Defects)
- **B** Severity Index
- **C** Defect Age

Defect Escape Rate. La proporción Bug/Defect mide exactamente la fuga, porque Defect=en sprint y Bug=post-release. Es la métrica núcleo del shift-left.

Q2 El equipo de producto abre el dashboard de ShopFlow y pregunta "¿qué parte de la app concentra más defectos?". ¿Qué módulo lidera?

- **A** Checkout (11 defectos)
- **B** Notifications (3 defectos)
- **C** No se puede saber: falta rellenar Components

Checkout (11). Como ShopFlow rellena Components en las 40 incidencias, el gráfico de ownership se dibuja entero: Checkout y Auth (8) lideran → ahí reforzar. Si el campo estuviera vacío, la respuesta sería C — pero esa es exactamente la disciplina que este equipo no descuidó.

// checklist de higiene

Disciplina de Defect Management

- **Usa el work type correcto** — Defect en sprint, Bug post-release. Es lo que hace medible el escape.
- **Rellena Components siempre** — sin él, no hay dashboard de ownership.
- **Severity ≠ Priority** — dos ejes, dos campos, dos preguntas.
- **Evidencia siempre** — captura, trace o log embebidos.
- **Fija Root Cause tras el RCA** — alimenta la mejora de proceso.
- **Triagea con honestidad** — Duplicated/WAD/CNR mantienen limpia la señal.
- **Enlaza US ↔ incidencia** — `causes` / `is blocked by` para trazabilidad.
- **Primero la pregunta, luego la métrica** — nunca el dashboard por el dashboard.

// para llevar

Tres Ideas que se Quedan

1

El momento define el tipo

Bug = defecto = síntoma idéntico. Lo que cambia es cuándo nace y qué le dice al equipo. Defect contiene; Bug se fuga; Improvement mejora.

2

Un workflow, los mismos campos

El estándar de calidad: las tres incidencias comparten esquema. La disciplina al rellenar — sobre todo Components — es la materia prima de toda métrica.

3

Mide preguntas, no datos

Un dashboard es un set de respuestas. Pregunta primero, métrica después, campo al final. Escape Rate es la estrella del shift-left.

Reporta con Disciplina. Mide con Preguntas.

Distinguir (Bug/Defect/Improvement) → Gestionar (un workflow, los mismos campos) → Medir (de campos rellenos a dashboards que el equipo de producto entiende de un vistazo).

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