Team Coordination, Project & Issue Management

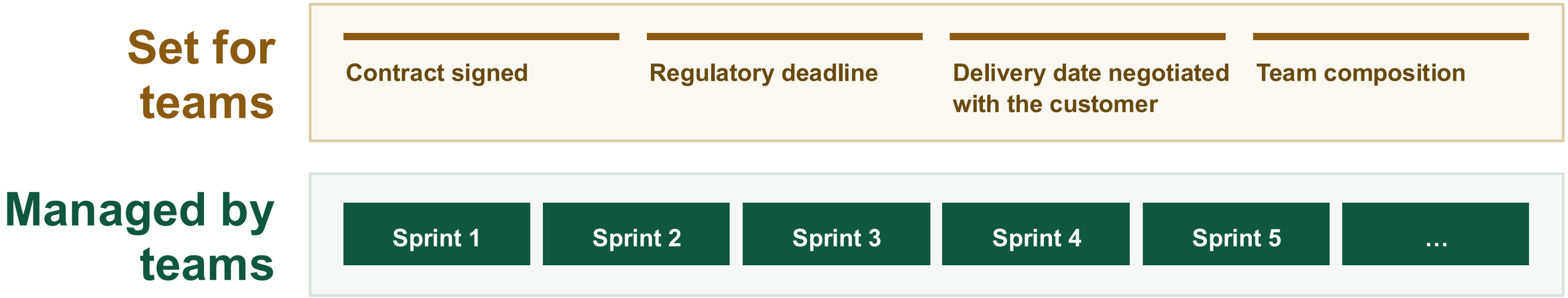
Oscar Chaparro

Agenda

1. Team Coordination — Roles, responsibilities & communication
2. Project Management — Decomposition, estimation & tracking
3. Issue Management — Organizing, triaging & tracking

How work moves

Coordination and constraints



TYPICALLY DICTATED

- Delivery dates — contracts, regulation, customer negotiation
- Who is on the team, and for how long
- Process and tooling, in many organizations

ACTUALLY YOURS

- How work is decomposed and sequenced
- How it is tracked and made visible
- How it is reviewed and handed off

Team Coordination

Roles, Responsibilities & Communication

Relay race or rugby?

Waterfall — a relay race

The baton is passed sequentially. Each runner works alone.

Modern software — rugby

Cross-functional · self-organizing · overlapping phases

Scrum roles at a glance

YOUR PROJECT

TODAY

Roles

Product Owner
ScrumMaster
Development Team

Each role is a distinct accountability — not a job title.

SEE LAST LECTURE

Ceremonies

Sprint planning Daily stand-up Review & retrospective

SEE LAST LECTURE

Artifacts

Product backlog Sprint backlog Increment

The development team is self-organizing — nobody in this picture assigns work.

Product Owner

- Maximizes **return on investment (ROI)** — value delivered against what the work cost
- Owns and orders the product backlog
- Ensures the team understands backlog items
- **One person — not a committee**

ScrumMaster

- **Servant-leader** — not a manager or task assigner
- Ensures Scrum is understood and enacted
- Removes impediments blocking the team
- Coaches the team and the organization

Does not assign tasks and does not decide what gets built.

The Development Team

- **Cross-functional:** all skills to deliver a “Done” increment
- **Self-organizing:** no titles, no sub-teams, no hierarchy
- **Size**: typically ten or fewer · full-time commitment
- **Collectively accountable** for the sprint goal

Design	Code	Test	Deploy

Brooks's law

Adding people to a late project makes it later.

3 PEOPLE

3

communication paths

10 PEOPLE

45

paths — worst case, everyone to everyone

$n(n-1)/2$

Paths grow quadratically. Onboarding cost is paid by the people already working.

- **Team size:** small teams answer superlinear coordination cost
- **Scaling:** ownership and interfaces remove dependencies; frameworks only manage them
- **Planning:** with a fixed date, adding people is not a real lever. Scope is.

Roles / stakeholders

Inside the team*

Developers*
Technical lead*
QA / test engineers*
Product owner
Business/reqs analysts
Architects
UI/UX designers
DevOps · Site Rel. Eng. / on-call
Data engineers DB Admins

Inside the company, outside the team

Project / delivery manager
Security specialists
Release / program managers
Support & customer success
Legal, compliance, procurement

Outside the company

The customer who pays
End users (often not the customer)
Regulatory bodies
Auditors & certification authorities
Vendors and integration partners

Work and roles beyond Scrum

METHODOLOGY	HOW WORK IS SCHEDULED	KEY ROLES	WHERE IT FITS
Scrum	Fixed-length sprints	Prod Owner (sets priority) · Scrum Master · Devs	Stable teams, negotiable scope
Kanban	Continuous flow, WIP limits, pull	No prescribed roles · the queue owner sets priority	Support and ops; work arrives unpredictably
XP (Extreme Prog)	Short iterations plus engineering practices	On-site Customer (sets priority) · Devs in pairs · Coach	Teams that need pairing, TDD, CI, collective
Waterfall / stage-gate	Phases with sign-off gates	Proj Mgr · Analyst · Architect · Devs · QA	Regulated, safety-certified or fixed-price work
SAFe (Scaled Agile Framework)	Quarterly planning across a release train	Release Train Eng · Prod Mgmt · team Prod Owner + Scr Master	Many teams delivering one product

The method is usually chosen by constraints, not by preference.

Team working agreement

YOUR PROJECT

Decide how the team operates on day one. Keep it in the repo. Revisit it at the retro.

Availability

When each person works, expected response time, and what counts as urgent

Where decisions live

If it is not in the issue or the PR, it did not happen

Definition of done

Tests pass, reviewed, merged, issue closed

Review turnaround

A committed maximum — one working day

Meeting cadence

What, when, how long, and what happens if you miss one

Escalation

What the team does when someone goes quiet

Conflict is expected. Set the rules while it is cheap.

Sprint 0 covered some of this. Real development starts now: agree to it explicitly.

Leader vs. Manager

Leader

Sets vision and direction
Shapes team culture
Leads by example, earns trust
Inspires rather than controls

Manager

Makes work assignments and schedules
Finds and allocates resources
Gives team members autonomy
Facilitates communication

Technical lead

Architecture and technical decisions. Leads through credibility and judgement.

Project / delivery manager

Schedule, scope, reporting, external commitments. Coordinates through process.

Product owner

What gets built and in what order.

Leadership without a title

In your team, nobody has authority. Leadership is behavior.

- Who calls the meeting when the team has drifted
- Who unblocks the teammate who has been stuck for three days
- Who notices the integration risk before the last week
- Who writes down the decision everyone thinks they remember
- Who says the uncomfortable thing in the retro

On your team: rotate the role weekly. The person on duty runs stand-up, grooms the board, and runs the retro.

Resolving team conflicts

Conflict is normal — it usually peaks early, before the team has agreed how it works.

- Goal: find a solution and move forward
- Ensure everyone works from the same set of facts
- Establish discussion ground rules before starting
- Express how you feel — never presume others' intent
- Stay calm; if triggered, step away briefly and return
- If impasse: escalate to a team leader or facilitator

“Fight forward, not back.”

Unresolved conflict stalls the team.

The daily stand-up

Time-boxed: **15 minutes** · same time and place every day · the team runs it, not the ScrumMaster

1

What did I complete since yesterday?

2

What will I do today?

3

What obstacles are blocking me?

✗ Status report for a manager

✓ **Peer commitment to the team**

Sprint review & retrospective

Sprint Review

EXTERNAL FOCUS

Team demos completed work to stakeholders
Stakeholders provide feedback
PO updates the backlog from that feedback

Time-boxed: 1–2 hours

Sprint Retrospective

INTERNAL FOCUS

START — what should we start doing?

STOP — what should we stop doing?

CONTINUE — what is working?

Team-owned; ScrumMaster facilitates. Output: concrete improvement actions.

How to run a meeting

1

Set the agenda

Share it before the meeting — topics, owners, time slots

2

Start on time, end on time

Respect everyone's schedule

3

End with action items

What, who, by when — captured in GitHub Issues or notes

Coordinator

Runs the meeting, keeps it on track

Scribe

Captures notes and action items

Checker

Signals when discussion drifts off-topic

Do you need a meeting?

Why do you want to call a meeting?

Complex discussion · decision needs real-time alignment	Information sharing · simple status update	Decision or feedback that must be tracked
↓	↓	↓
Schedule a meeting	**Send an async message or email**	**File a GitHub issue or comment**

Meetings are expensive — default to async when you can.

Communication channels

Paper / documentation	Email	Chat / IM	Phone / audio	Video call	Face-to-face	Face-to-face at a whiteboard

Least rich / effective

Most rich / effective →

Rule: match channel richness / effectiveness to task complexity. Consider info persistence.

- Casual status update → chat is fine
- Architectural decision → face-to-face or video

An argued heuristic, not a measurement. Cockburn and Ambler drew this from practitioner observation. Ambler adds a second axis: the richest channels leave no record.

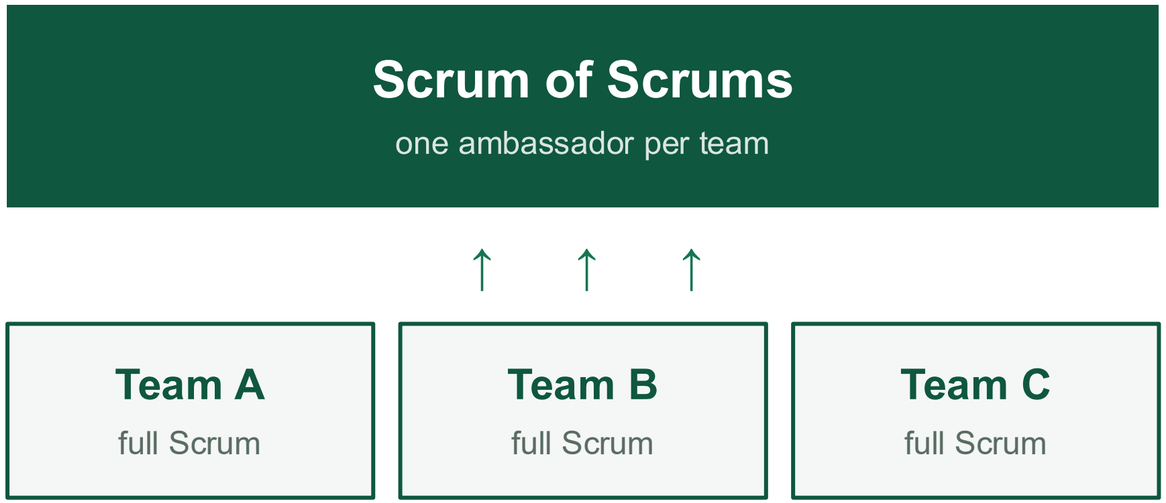
It predates distributed work. Written async communication is durable, searchable, timezone-independent and forces clarity — the model scores all of that as loss.

Pick the channel(s) for each.

Docs	Email	Chat	Phone	Video	Face-to-face	Whiteboard

- A design decision the team will need to look up in three weeks
- Telling a teammate their share of the work isn't landing
- A bug you can't reproduce and need a second pair of eyes on
- Deciding which features or scope to drop two days before the demo
- Onboarding a teammate to code you wrote

Scaling: Scrum of Scrums



The ambassador attends the sync; the rest of each team stays focused.

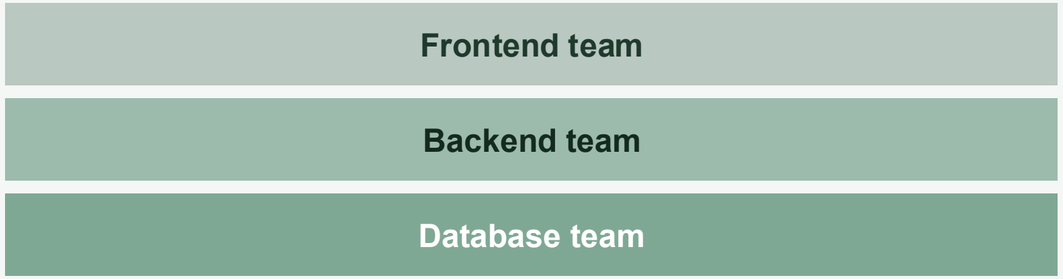
Three questions at the sync

- What did my team do that affects yours?
- What will we do next?
- What cross-team blockers exist?

Conway's law

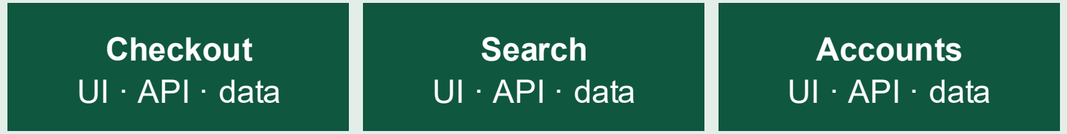
A system's structure mirrors the communication structure of the organization that built it.

Functional teams → one monolith



One deployable, three owners. Every feature crosses all three teams.

Vertical teams → modules or services



One owner each. Most features live inside one team; coordination only at the seams.

It has been measured. In Windows Vista, the components most likely to fail were those touched by most teams. Code ownership predicted bugs better than code complexity.

On your team: whoever keeps working in a module owns it. Where two people's work meets, agree on what gets passed before either builds.

Amazon two-pizza teams

INDUSTRY

- **Small teams:** small enough to be fed with two pizzas
- **Service ownership:** one team owns one service end to end — build, deploy, operate
- **The 2002 API mandate:** all inter-team communication goes through service interfaces
- **“You build it, you run it”:** the team that ships the code carries the pager

Catalog	Cart	Checkout	Payments	Search	Recommendations	Reviews	Shipping

WHY IT WORKS

Teams agree on an interface once, then build without meeting together. Most communication paths stop existing instead of having to be managed.

WHAT IT COSTS

Every team runs its own module: deploys, monitoring, on-call. Duplication across teams (their own API, DB, etc.) is accepted as the price of independence.

Project Management

Decomposition, Estimation & Tracking

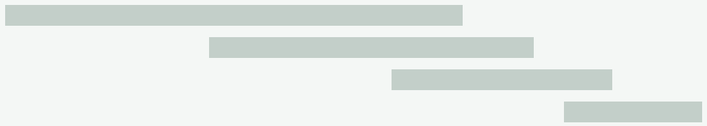
Iterative planning

⚠ NOT OUR FOCUS TODAY

Plan-driven

Gantt · PERT · CPM

Full scope defined upfront
Detailed schedule down to individual tasks
Changes are expensive and require replanning
Works when requirements are stable



WHAT WE USE

Iterative / Agile

Sprints · backlog · velocity

Scope emerges sprint by sprint
Only the near horizon is planned in detail
Changes are expected — reordering the backlog is cheap
Works when requirements evolve

How planning actually works

INDUSTRY

Year	Budget, headcount, a few committed objectives	**Decided by:** Executives, finance
Quarter	Roadmap themes and epics, sized in weeks, cross-team dependencies	**Decided by:** Project / program management with team input
Sprint	Detailed decomposition and estimation of two weeks of work	**Decided by:** The team

WHAT DRIVES THE PLAN

- Fixed external dates are inputs, not outputs
- Capacity is deliberately underplanned — unplanned work is guaranteed
- Cross-team dependencies; where most quarterly plans fail

Most projects overrun. A review of estimation surveys found schedule or budget overruns in 60–80% of projects. Re-planning each quarter is the mechanism, not a failure of the last plan.

One plan, three horizons

EXAMPLE

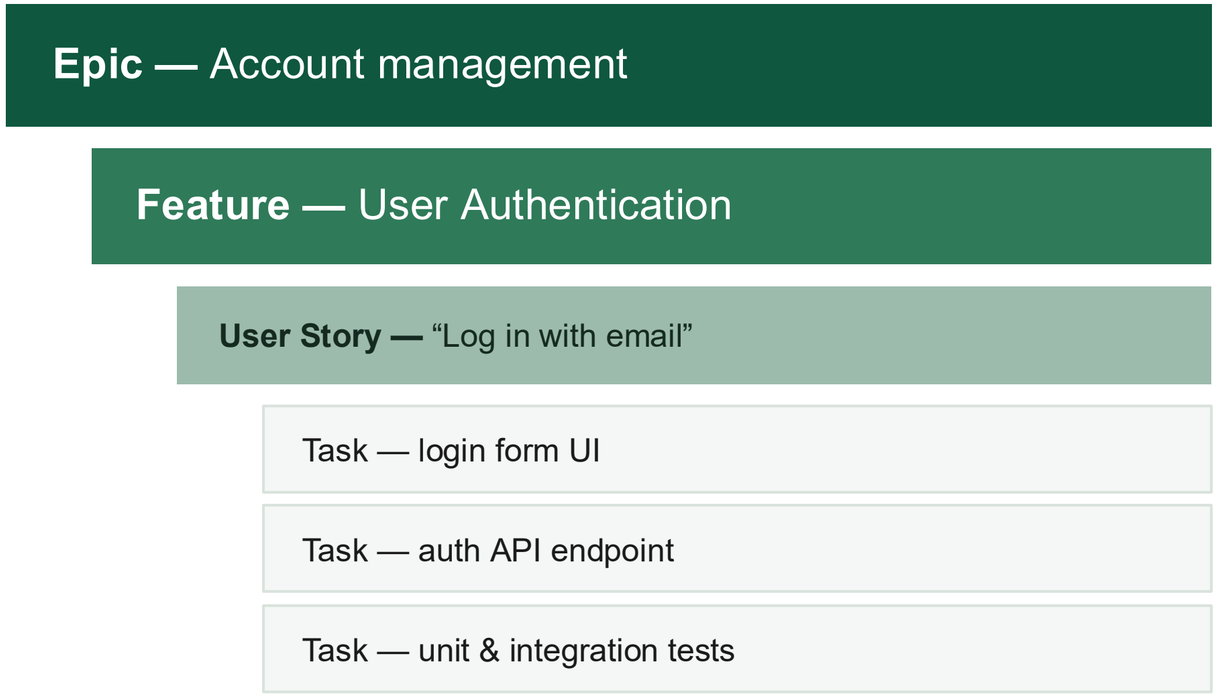
A payments team of eight engineers, in January.

Year	Three themes: international expansion, fraud reduction, build and test speed. Scoped to what eight engineers can plausibly do in a year. No feature list.
Quarter	Two epics pulled from those themes: multi-currency checkout, fraud scoring. Sized in weeks, not stories. One probably won't fit.
Sprint	Stories from the active epic: “accept EUR at checkout”, “show converted totals in the cart”. Written the week before, not in January.

What could happen. Fraud scoring hit an unplanned data-retention review and slipped a quarter. Multi-currency shipped early. The roadmap never changed — it only ever committed to the themes.

On your team: the milestones and demo date are your year. The sprint is your quarter. The issues you pick each week are the rest.

Work decomposition



Why decompose?

- Makes work estimable and assignable
- Enables parallel work across the team
- Gives visibility into progress at multiple levels

In your project, everything is a **GitHub issue**. Most issues are stories; split one into task issues when it is too big for one person.

User stories

A user story captures **what** a user wants and **why** — not how it is built.

GOOD — ACCOUNTING SOFTWARE

"As an **accountant**, I want to **see all outstanding invoices** so that **I know what payments are owed**."

FORMAT

As a **[role]** , I want **[function]** , so that **[value]**.

✗ "As a developer, I want to refactor the database layer." — no user value expressed; this is a task, not a story.

User story format & the 3 C's

As a [role]

Who benefits? The user — not the developer, unless the developer *is* the user

I want [function]

What capability? Behavior-level, not implementation

So that [value]

Why does this matter? It justifies building it

The 3 C's — Ron Jeffries

Card

The written story — a reminder to have a conversation

Conversation

The discussion that fleshes out the details

Confirmation

Acceptance criteria that define “done”

The 3 C's in practice

EXAMPLE

Card
When the idea appears

“As a student, I want to sign in with my Google account, so that I don't manage another password.”

Conversation
Before and while building

What if the Google account matches an existing email — link the accounts or reject it? Does the old password still work afterwards? **Decision: link on matching email, keep the password.**

Confirmation
Written before code, checked after

New Google account creates a user · matching email links to the existing account · password login still works · denied consent returns to sign-in with a message

INVEST criteria

I **Independent** — developable without depending on other stories (minimize, not eliminate)

N **Negotiable** — not a contract; details stay open until the sprint starts

V **Valuable** — delivers value to an end user or customer

E **Estimable** — the team can size the effort; enough detail exists

S **Small** — fits in a sprint; completable in a few days

T **Testable** — acceptance criteria exist; the team knows when it's done

If a story fails any criterion — refine it or split it.

Evaluate these stories.

1. *“As a user, I want a dashboard, so that I can see my data.”*
2. *“As a user, I want search results to load in under 200ms, so that the app feels responsive.”*
3. *“As an admin, I want to manage users, so that I can keep the system secure.”*

Independent | **Negotiable** | **Valuable** | **Estimable** | **Small** | **Testable**

Each one is well-formed. Check all six — which does it fail, and how would you rewrite it?

What you estimate

Sign in with Google #142

As a student, **I want** to sign in with my Google account, **so that** I don't manage another password.

ACCEPTANCE CRITERIA

- ☐ New Google account creates a user
- ☐ Matching email links to the existing account
- ☐ Password login still works afterwards
- ☐ Denied consent returns to sign-in with a message

OUT OF SCOPE

Other providers. Unlinking an account.

enhancement | auth | Sprint 3 | 5 points

Story points

- Capture complexity + uncertainty + volume in one number
- Relative sizing: story A is twice as complex as story B
- Avoids the false precision of hour estimates
- A point is meaningful **within** a team — never comparable across teams

COMMON SCALES

1	2	3	5	8	13	21

Fibonacci, or T-shirt sizes (S / M / L / XL)

EXAMPLE — HOTEL RESERVATION BACKLOG

Story	Points
Search availability	3
Book a room	5
Cancel booking	3
Payment integration	8

What do you estimate in?

UNIT	WHERE IT IS USED
T-shirt sizes	S / M / L / XL — epics and roadmap items, where precision is pointless
Story points	1, 2, 3, 5, 8, 13 — non-linear, because uncertainty grows with size
Ideal days	“How long uninterrupted” — intuitive, but read as calendar days
Story counting	Count completed stories instead. Needs consistent sizing
Hours	Tasks only — 2h, 4h, 1d. Never stories

A story at 13+ is a signal to split, not an estimate. Velocity turns points back into a forecast — within one team only.

Planning Poker

YOUR PROJECT

1
PO reads a story; team asks clarifying questions

→

2
Each member privately selects an estimate card

→

3
All cards revealed simultaneously

→

4 · 5
Consensus → record.
Divergence → **the high and the low explain their reasoning** , then re-vote

↻

BEFORE REVEAL

AFTER REVEAL — NO ANCHORING

The estimate is the byproduct. The product is shared understanding.

Size these three against each other.

A Add a logout button

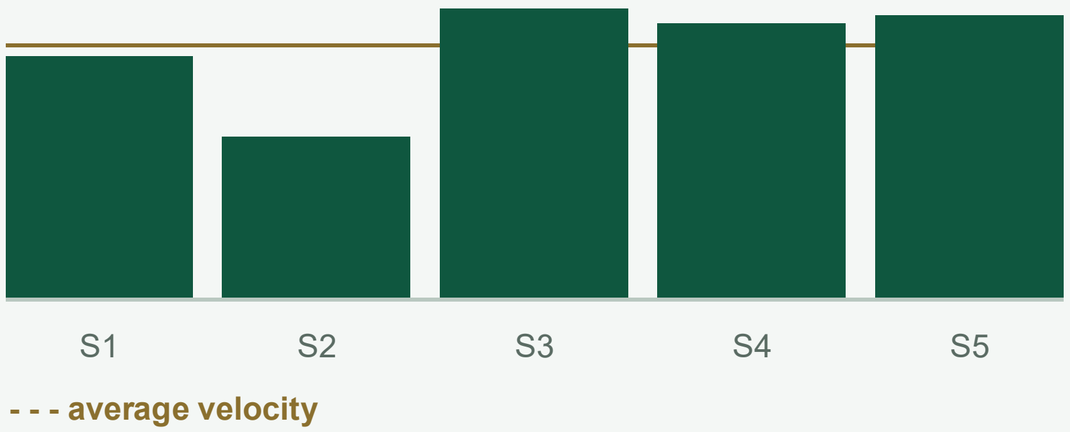
B Add search with filters

C Export the user list as CSV

Order them from smallest to largest, then say how much bigger each step is

Velocity

POINTS COMPLETED PER SPRINT



Velocity = story points completed per sprint.

- Emerges after 2–3 sprints. You have none until sprint 1 closes — calibrate against a reference item until then
- Once stable, forecasts releases: 80 ÷ 20 = 4 sprints — treat it as a range, not a date

⚠ Not a performance metric

Do not compare across teams or use it to pressure a team. Velocity is self-reported in a self-defined unit: the moment it is used to judge a team, the points inflate and the measure stops tracking anything. Rising velocity is recalibration, not improvement.

Measure the process, not the people

INDUSTRY

> **“When a measure becomes a target, it ceases to be a good measure.”**
>
> Goodhart's law, as phrased by Strathern

Measure the process — quantitatively

- **Cycle time** — how long one item takes from starting it to delivering it
- **Work in progress** — how many items are in progress at once. Start less to finish sooner
- **Escaped defects** — bugs found after release

Assess people — qualitatively

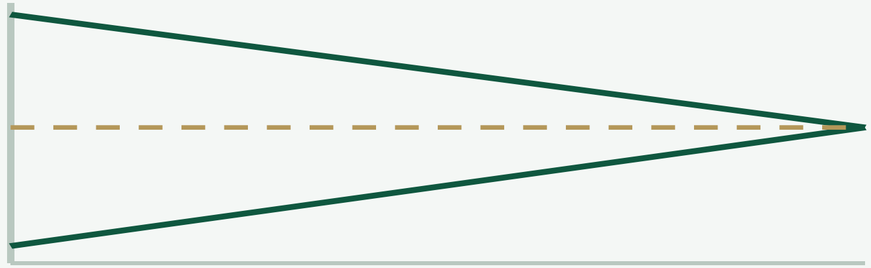
- Did what they built work and get used — not how much they produced
- Peer feedback from the people who work with them
- Growth against a written standard for their role, not against teammates
- Judgement where there is no right answer: reviews, design, incidents

Lines of code, commits and tickets closed are trivially gamed, and they punish reviewing, mentoring and unblocking.

Estimation is hard

- Early estimates are expected to be wrong — calibrate as you learn
- Estimates are forecasts, not commitments — give a range, not a date
- Accuracy improves as the team matures and requirements stabilize
- When in doubt: over-estimate rather than under-estimate — protect team capacity
- Use the retrospective to recalibrate systematically

CONE OF UNCERTAINTY



Estimate accuracy narrows as the project progresses and knowledge accumulates.

The product backlog

An **ordered** list of everything that might be built in the product.

- Owned and ordered by the Product Owner
- Never complete — continuously refined (“groomed”)
- Top items: detailed, estimated, ready to sprint
- Lower items: coarse, may lack estimates
- Backlog refinement: a regular session to break down and estimate upcoming items

HOTEL RESERVATION SYSTEM — PRODUCT BACKLOG

Story	Priority	Points	Sprint
✓ **Search availability**	P1	3	1
✓ **Book a room**	P1	5	1
✓ **Cancel booking**	P1	3	1
Payment integration	P2	8	2
Loyalty points	P3	—	—
Multi-language UI	P4	—	—

✓ = sprint-ready: detailed, estimated, ordered at the top

P1 now · P2 this sprint · P3 a coming sprint · P4 someday

Sprint planning

INPUTS

Product backlog (ordered)
Team capacity Previous velocity

→

SPRINT PLANNING MEETING

PO presents top items
Team discusses
Team identifies tasks
Team estimates and verifies

→

OUTPUTS

Sprint goal Sprint backlog

Getting to sprint 1

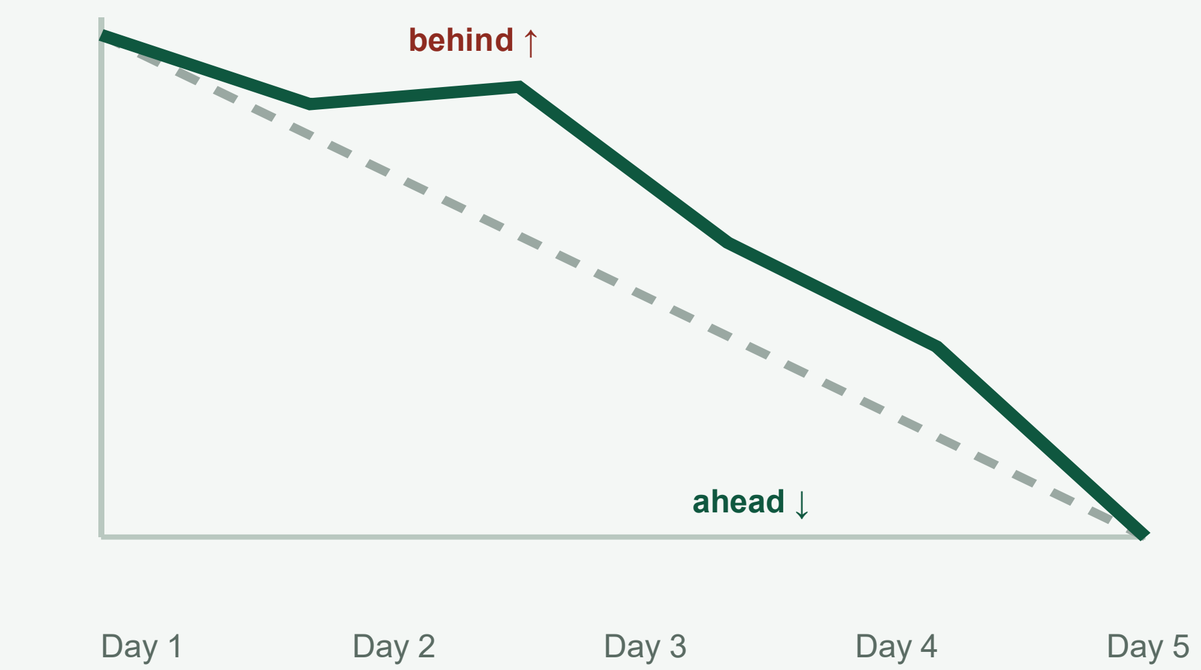
Sprint 1 runs **Sep 22 – Oct 8**. One meeting — before the sprint if you can, in its first days if not. Everything else async.

	STEP	WHAT HAPPENS	IF IT DOESN'T
1	**Self-nominate** Async · before the meeting	Read the issue list. Comment on the three or four you would take.	Someone else picks your issues for you.
2	**Priority input** Async · before you plan	We comment on the issues to say which features matter most — treat us as the customer.	—
3	**Plan together** One 45-minute meeting, as early as you can	Three decisions: the order, what is in sprint 1, who owns what.	It runs at its scheduled time with whoever is there. Decisions stand.
4	**Detail your issues** Async · within 48 hours of planning	The owner writes acceptance criteria, a task checklist, and open questions.	Undetailed issues drop to the bottom of the list.
5	**Read your neighbours** Async · once yours are detailed	Read the issues that touch yours. Comment on what gets passed between them.	You find out at integration instead.

Load: two to three medium issues each, or the equivalent. Split anything over about three days.

Decisions go in issue comments, so whoever missed them catches up by reading.

Sprint burndown chart



- X-axis: sprint days · Y-axis: points or hours remaining
- **Ideal line:** straight from the day-1 total to zero on the final day
- **Actual line:** real remaining work each day
- Above ideal → behind schedule; below ideal → ahead

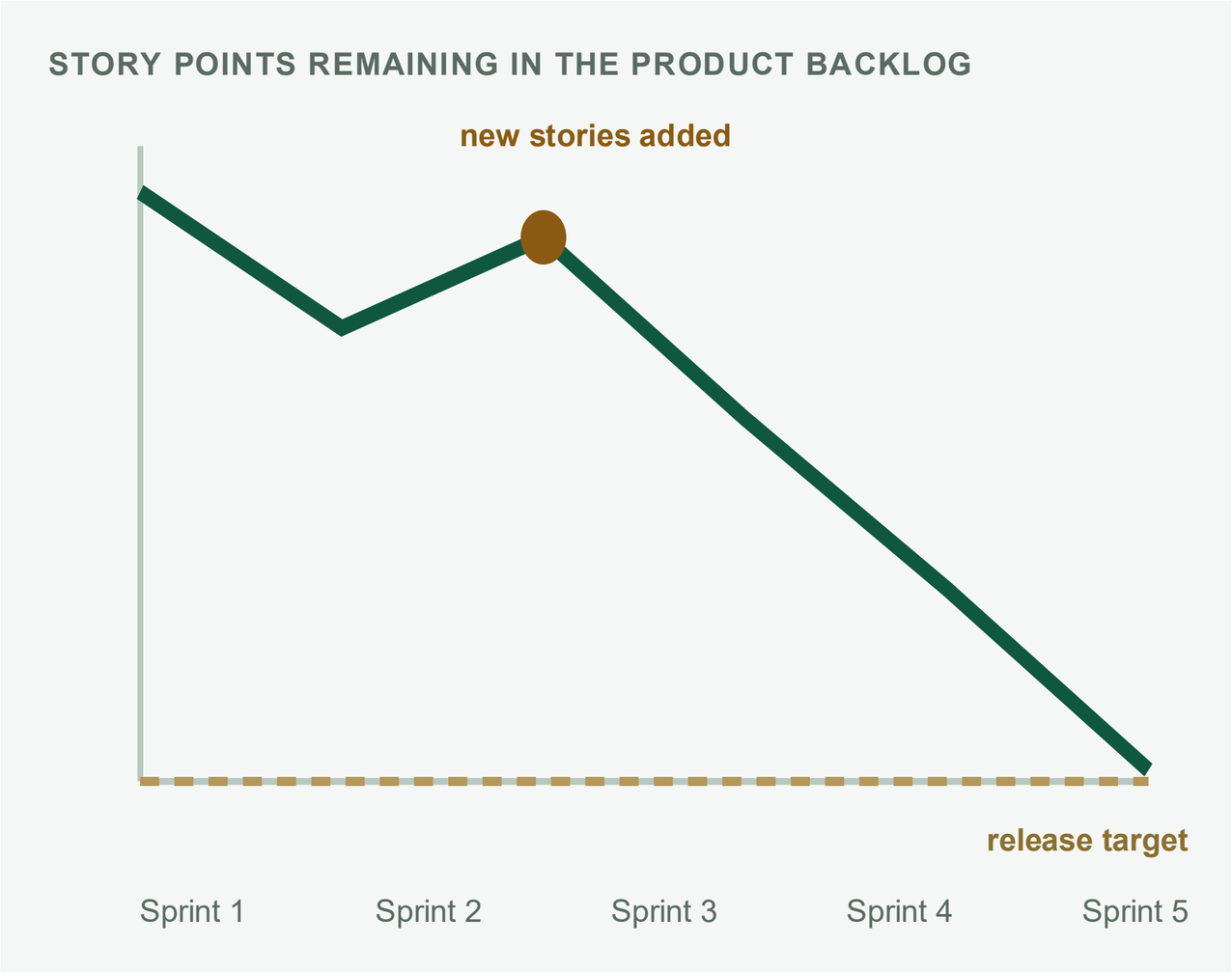
- - - ideal
—— actual

Used for daily inspection and early warning.

Which units? Hours historically. Points commonly, today. For a small team, item count — the milestone bar gives it free.

A flat burndown usually means scope was added — and the chart cannot show that. A burn *up* can.

Release burndown



Progress across the whole release — not one sprint.

- Sprints across · points remaining in the product backlog up
- Ideal: a steady decline toward the release target
- Reality: bumpy — stories get added mid-release

Pair it with a burn-up. A burndown cannot tell “we are slow” from “scope grew” — both flatten the line. A burn-up draws scope separately, so growth shows.

Forecast a range, not a date. Extend trend lines from your best and worst recent sprints.

This is the stakeholder-facing chart. The sprint burndown is team-internal.

Issue Management

Organizing, Triaging & Tracking

What is an issue?

Any unit of **trackable work** — not just a bug.

Bug

Something is broken or incorrect

sometimes → story

Feature request

New capability requested by users or stakeholders

written as a story

Enhancement

Improvement to an existing feature

written as a story

Refactoring / tech debt

Internal improvement, no visible behavior change

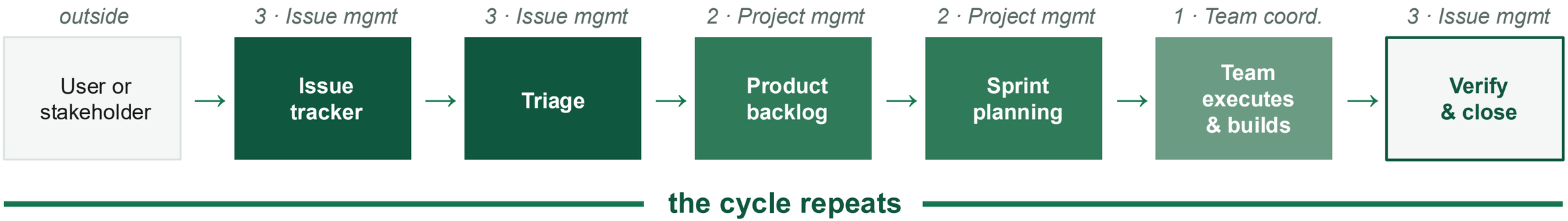
Task

Operational work — update docs, set up CI

A story is a *format*, not a type — it applies whenever the work has user-visible value.

If you can't name a user in it, it isn't a story — label it a task or tech debt and move on.

The issue management process



Anatomy of a good issue report

ISSUE REPORT FIELDS

Title — specific and searchable

Type — bug / feature request / enhancement / task

Description — full context, problem or idea

Steps to reproduce (bugs) or **scenario** (stories) — a concrete walkthrough

Expected vs. actual (bugs) or **acceptance criteria** (stories)

Severity and **priority**

Environment — OS, browser, version, hardware

Assignee, label(s), milestone

Attachments — screenshots, logs, stack traces

THE FOUR-PART ANATOMY

Context
Why does this matter? Who is affected?

Problem / idea
What exactly is wrong or desired?

Previous attempts
What was already tried?

Proposed solution / next step
What should happen now?

A bad issue report

The image slider is broken #2

Open alex opened this 6 days ago · 3 comments

When I open the slides, it just doesn’t work: I can’t see my images.

maria 5 days ago Which page? I can't reproduce it.

alex 2 days ago The one with the photos. It just doesn't work.

maria yesterday Can you send a screenshot?

No labels · no assignee · no milestone

What's missing?

- ✗ No context — when does it break? on which page?
- ✗ No steps to reproduce
- ✗ No expected vs. actual behavior
- ✗ No environment information

Result: impossible to triage, assign, or fix without extensive follow-up.

A good issue report

Slider images are cropped on the right at narrow widths #3

Open · bug · S3 · P2 · maria · Sprint 3

Context

The gallery slider shows one photo at a time and should scale each to fit its frame at any window width.

Steps to reproduce

1. Open the gallery page 2. Narrow the window to about 800px 3. Click the right arrow twice

Expected

The whole image fits inside the frame.

Actual

Roughly 80px is cut off the right edge. Screenshot attached.

Environment

macOS 12.1 · Chrome 97 · MacBook Pro M1 · commit a4f19c2

Why it works

- ✓ **Context** — what the slider is supposed to do
- ✓ **Numbered steps** to reproduce
- ✓ **Expected vs. actual** — “should fit without cropping” / “cropped on the right”
- ✓ **Environment** — macOS 12.1, Chrome 97, M1 MacBook Pro

Any team member can pick it up and start immediately — no follow-up questions.

A good enhancement issue

Remember slider position between visits #47

enhancement

STORY

As a returning visitor, **I want** the slider to reopen on the image I last viewed, **so that** I don't lose my place.

SCENARIO

Given I viewed image 3 and left the page
When I return within 24 hours
Then the slider opens on image 3

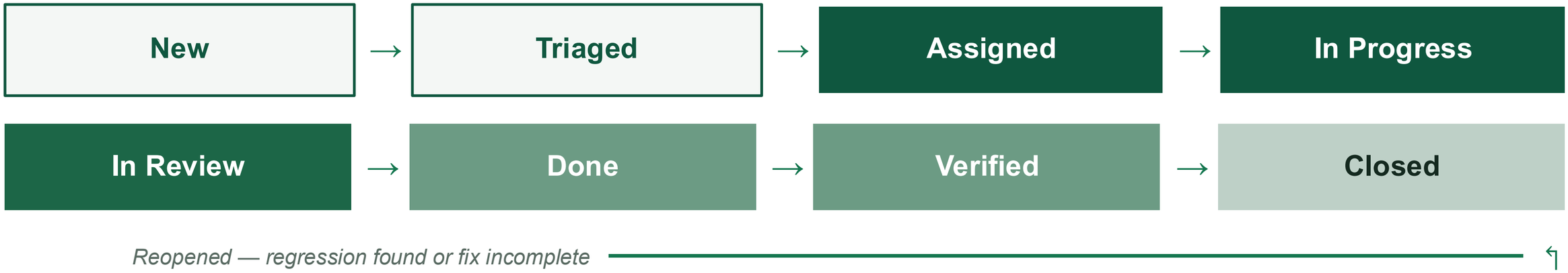
ACCEPTANCE CRITERIA

· Position persists for 24 hours
· Falls back to image 1 if expired or storage cleared
· No change for first-time visitors

Why it works

- ✓ **Story sentence** — a user, a want, and a why
- ✓ **Scenario** — one concrete path, given / when / then
- ✓ **Acceptance criteria** — every condition for done, edge cases included
- ✓ **Typed and labelled** — an enhancement, not a bug

Issue lifecycle



Rejected

From Triaged — not valid or out of scope

Won't Fix

From any active state — a disposition without resolution

Duplicate

From Triaged — flagged, linked to the existing issue, then closed

The same lifecycle applies to every issue type.

Issue Triage

Reviewing new issues and deciding what to do with them.

WHAT

- **Validate:** is this real and reproducible?
- **Classify:** type, severity (S1–S4), priority (P1–P4)
- Label and assign to a milestone or sprint
- Request more info if needed
- Mark duplicates; reject invalid reports

WHEN

A regular cadence — daily or weekly. Plus on-demand for P1 issues.

WHO

Typically the ScrumMaster or Product Owner. Or a designated triager in rotation.

Severity vs. priority — not the same thing

YOUR PROJECT

Severity — impact on the system

S1	Critical — crash, data loss, security
S2	Major — core feature broken, no workaround
S3	Minor — feature affected, workaround exists
S4	Trivial — cosmetic, low impact

Priority — urgency of fixing

P1	Now — drop what you're doing
P2	This sprint
P3	A coming sprint
P4	Someday — bottom of the backlog

Set by the Product Owner, weighing customer impact, release timing, and business context.

S1 + P4 — HIGH SEVERITY, LOW PRIORITY

A crash in a rarely-used feature

S4 + P1 — LOW SEVERITY, HIGH PRIORITY

A typo on the homepage the day before launch

These two can diverge.

Give each a severity and a priority.

SEVERITY — IMPACT

S1 Critical — crash, data loss

S2 Major — broken, no workaround

S3 Minor — workaround exists

S4 Trivial — cosmetic

PRIORITY — URGENCY

P1 Now

P2 This sprint

P3 A coming sprint

P4 Someday

- A crash on one internal test account
- A typo in the product name on the landing page
- Data loss in a feature shipping next month
- A slow page the biggest customer complains about weekly

Roles in issue management

Role	Responsibility	Maps to Scrum
Reporter	Files the issue	Anyone — user, tester, developer, stakeholder
Triager	Reviews and classifies	ScrumMaster or Product Owner
Assignee	Implements the fix or feature	Development team member
Verifier	Confirms the fix works and closes the issue	A **different** development team member

Verifier ≠ Assignee

“The developer who wrote the fix should not be the one who closes it.” Separating implementation from verification is essential for quality.

At scale: dedicated support tiers

Most organizations don't absorb production load inside the product team. They tier it.

L1 · Support
Intake, triage, known fixes

L2 · Sustaining
Escalations and non-urgent fixes

SRE
Availability, paging, incident command

Product team
Sees only what escalates

Error budget
A reliability target that decides when feature work stops and reliability work starts.

Incident ≠ fix
Response restores service. The postmortem produces issues that enter the backlog normally.

A dedicated tier protects product velocity, but it adds a handoff and distances the team from how its software fails.

Which is the argument for the opposite model: *you build it, you run it.* Both are widely practised — it's a team-boundary decision.