🧭 Product Context

- SaaS product live for 1 year
- Customers request visibility into **team usage**
- Need a **dashboard for org admins**

Requested metrics:

- Active users over time
- Most used features
- Last login per user
- Org-wide usage summary

🧱 Current Codebase Reality

- Backend logs activity in `jsonb`, semi-structured
- Internal analytics exist but not reusable
- Frontend has an admin panel, but it’s underdeveloped
- Mixed code quality:
  - User/auth logic is clean
  - Analytics code is messy and scattered
  - Minimal test coverage in this area
  - No design system for frontend

🧠 How Do We Manage the Timeline to Roll Out the Feature?

1. 🪜 Breakdown the task for easier estimation
2. 🔍 Spike tasks to deal with unknowns
3. 🎯 Use correct pointing system
4. 🔧 Correct coding strategy
5. 🌀 How do we avoid mid-sprint randomness?

🪜 How Should We Break Down a Task?

Goal: Make tasks small, focused, and estimate-friendly.

🔹 Good Practices

- ❌ Avoid vague tasks — be specific and outcome-focused
- ✅ Define “done” for every task
- 🔄 Split work by delivery steps, not tech layers
- 🔍 Use spike tasks for investigation or unclear work
- 🧼 Include supporting tasks (tests, docs, cleanup)

🔍 Why Spike Tasks Matter

Spikes = Time-boxed investigation tasks

Used to explore or reduce uncertainty before estimation.

💥 When to Use Them

- Task has too many unknowns to estimate confidently
- Need to assess feasibility or options
- Risk of under/over-estimation is high

✅ Benefits

- De-risk upcoming work
- Build shared understanding
- Allow better estimation next sprint
- Prevent random deep-dives mid-sprint

🎯 Why Points Can Fail in Small Teams

Story points are often **too abstract** for small teams, especially without consistent velocity data.

⚠️ Common Issues

- Not enough historical data to calibrate
- Points become guesses, not comparisons
- People confuse points with hours
- Too few people → team velocity is volatile

✅ What to Do Instead

- Use **hours or day-size chunks** if you prefer concreteness
- Focus on **task size + clarity**, not exact number
- If using points, **build shared examples** (e.g. “This is a 2-pointer”)

🔧 How Should We Code It?

Goal: Deliver high-quality features with minimal friction.

🔹 Good Practices

- 🧼 Start with refactoring — helps understand the code and add safety (tests)
- 🧠 Align early with a 1-pager design for complex features
- 🧾 Avoid over-documenting — write self-explanatory code instead
- ✅ Invest in automated tests — they save hours later

🌀 How Do We Avoid Mid-Sprint Randomness?

Goal: Stay focused on what we committed, while being realistic.

🔹 Strategies That Help

- 📦 Keep the sprint backlog tight — only what's well-defined and ready
- 📢 Communicate — speak up early when something goes off track
- 🚫 Avoid bundling tech debt into feature work unless planned
- 🧯 Escalate scope creep early — replan if needed