DevOpsDays Austin 2026 Observations

Borislav Sabotinov · May 5-6, 2026

After two days of talks, hallway chats, and open spaces (a new concept for me and one that I'm excited to share more about below), the main thread that emerged is that the systems we build around our engineers - process, tooling, AI, governance - matter more than any individual heroics. Here are some of the main observations I want to share.

> "A bad system will beat a good person every time."
>
> — W. Deming

This quote may be interpreted in two ways:

1. Even a bad system is better than no system at all, or
2. Good people can't overcome the limitations of a bad system.

Deming himself meant the second interpretation: even the best individuals can't consistently succeed in a poorly designed system. For me, that means we cannot rely on individuals to compensate for a flawed or dysfunctional process, and need to focus on improving the system itself.

Leading engineering teams

Shift focus away from the individual and toward the efficacy of the team as a whole. Optimizing individuals one at a time or leaving developers alone "so they can focus," both sound reasonable but both produce *worse* outcomes. One thing both approaches have in common is that they treat the team as a collection of separate people rather than an interconnected system that itself develops attributes and behavior independent of the individuals. The team is not just the sum of its members, each operating on their tasks, but the interactions and culture that connect them.

Treat the team as a system with three axes:

- **Product**: value delivery
- **Engineering**: quality gates
- **People**: growth and engagement

Thinking about the whole is what makes leadership proactive rather than reactive. The individual still matters but the system they operate inside is what determines whether their efforts build to something greater or stalls/evaporates.

Git literacy is regressing

New graduates are arriving with shallower Git knowledge than we'd expect, in part because so much of the day-to-day workflow has been abstracted behind GUI buttons. Most can: `clone` → `checkout` → `add` → `commit` → `push`. Few can confidently rebase, cherry-pick a commit, resolve a non-trivial merge, recover a detached HEAD, or reason about what actually lives in `.git/`.

As source control underpins everything we do, including every AI-assisted change, depth here is more valuable than ever. **A periodic team-level Git training** (an hour annually) is cheap insurance.

AI multiplies discipline

The clearest framing I heard all conference:

```
High discipline + AI = excellent results
Low discipline + AI = slop at scale
```

AI doesn't compensate for bad practices or help change them to good ones; it simply amplifies whatever was already there. If your team ships without tests (or low quality ones), rubber-stamps PRs, then AI helps you ship more untested code, faster. If your team has tight feedback loops and good review culture, AI compounds them and helps you move faster.

Practical prompting

- Match the agent to the task: planning, ideation, and building are different jobs and benefit from different models.
- Don't try to one-shot complex problems. Decompose it and solve the pieces, then aggregate. Have a higher-reasoning agent review the work.
- Use markdown files to give agents persistent instructions and context.
- Let the agent ask you clarifying-questions (most support this). The resulting richer and more accurate context is almost always worth the extra time and tokens.

Cognitive control plane & traceability

As more code gets written or co-written by agents, knowing which model produced which change becomes a real engineering concern. Different models have different "dialects" - preferences in structure, naming, error handling, library choice (e.g., one model defaults to broad try/catch wrapping, another prefers narrow exception types and explicit error returns). When a regression shows up, model attribution on the originating PR is the same kind of forensic data that `git blame` provides. Building it in early is much cheaper than retroactively adding it ("I'd rather have it and not need it, than need it and not have it").

AI's impact on hiring

Hiring managers across the industry are consistent that **engineering discipline and fundamentals are still what they're evaluating for**. AI fluency is additive. It doesn't substitute for being able to reason about a system, debug under uncertainty, or write code another human can maintain.

Open spaces are a powerful format

The open space format is unstructured time where attendees propose and self-organize around discussion topics. It was a new experience and a highlight for me. It created space for deep dives (locking down AI for security, the cost of observability, mental health, troubleshooting agents and measuring their success), cross-pollination between talks, and more candid exchanges. Participants are instructed to move on to a different room if they aren't actively learning or participating. I left with greater exposure to how other teams approach AI integration and testing, new tools to try (Docling for document processing and embedding), and strategies for being a true craftsman in an age of vibecoding.

RAG is not solved

The industry moves fast enough that fundamentals get prematurely declared solved. RAG is a recent, clean example. The dominant narrative, "Glean and similar products give you RAG-as-a-service, so nobody needs to build their own pipeline anymore," is only correct for a narrow set of problems (semantic search over documents) and quietly misleading for everything else.

RAG as a pattern to retrieve relevant context, ground the model in it, and generate is not specific to text. It generalizes to any data where you can build a meaningful similarity index:

- **3D geometry and CAD assemblies.** Retrieve geometrically similar parts from a group of past designs to ground a model's reasoning about a new part. This is being applied to part retrieval in CAD assemblies and to 6D pose estimation, where a CAD model serves as a structured knowledge base rather than just supervision.
- **PLM data.** The relationships between parts, BOMs, change orders, simulation results, and supplier data are exactly the kind of structured-plus-unstructured collection where retrieval generation outperforms a generic LLM. A model that can pull the right historical change order before suggesting a design modification is solving a real problem; a model that can't is guessing.
- **Simulation and CAE artifacts.** Mesh metadata, solver inputs, and result summaries are retrievable. The retriever doesn't have to be a vector DB over text; it can be over geometry features, parameter ranges, or solver configurations.

The CoTS RAG-as-a-service products don't index any of this. *The teams that recognize where the pattern still applies, and where the SaaS won't reach, have meaningful headroom left.*

Locking down AI: a non-obvious exploit

A leaked credential in a repo can become an AI agent's route to escalation. An agent with read access to the repo doesn't need access to the protected system; it just needs to read the live, exposed secret and then use it. The agent's permission limit is the union of its explicitly granted permissions PLUS every credential it can read.

A thought-provoking question from the open space discussion: *"If models hallucinate facts, can they hallucinate permissions, convincing themselves they're allowed to call a tool or take an action that policy forbids?"*

Code review should not be a rubber stamp

> "Looks Good to Me" is not a process.

Rubber-stamping PRs is worse than skipping review entirely as it imposes the cost without delivering the benefit, and it gives everyone false confidence that something was actually checked.

The takeaway I want to bring back: **good PRs are not the ones with all friction eliminated**. They're the ones that introduce intentional friction at the points where it reduces risk and improves stability. Drop the ceremonies and formalities - the back-and-forth in comments over style or preference that does not catch actual issues - and keep the substantive evaluation of architecture, bugs, integration, and overall flow. The goal isn't a fast green checkmark; it's a change you'd be willing to defend on a Friday afternoon.

The main takeaway across every session worth remembering: *the system around the engineer is what determines the ceiling we can reach.*