Worksheet

# Is your homegrown flag system still worth the upkeep?

A 5-minute self-check for teams running their own feature-flag system.

CloudBees

# Homegrown feature flags are the right call when it's one team and a few dozen flags.

## You build a simple system, wire it into your releases, and shipping gets easier.

But as you grow and flags, environments, and compliance requirements accumulate, that once-simple internal tool becomes a platform your engineers have to maintain. As flags pile up faster than you can retire them, you lose track of who approved each change. Often, just one engineer understands how the system actually works (and there's no plan for when they quit).

It's easy to ignore because nothing breaks all at once. The system just drags on every release and ties up engineers who could be shipping instead.

## So how do you know when it's time to move off it?

This assessment tells you whether your homegrown system still fits your team. Answer six quick questions about your flag system, tally the points, and get a clear list of your system's gaps.

Trade that gut feeling (this is getting harder to manage) for a score that tells your team what decision to make.

## How to score your feature-flag system

Each question has three answers. Pick the answer that reflects how your feature flags run today.  Give yourself 3 points for the first answer, 2 for the second, and 1 for the third.

Add up your total and read your result at the end.

Note: This is a risk score, not a performance score. The higher your total, the more urgently you need to move off, or start considering moving off, your homegrown system.

CloudBees

## Question 1: Engineering maintenance

Your score:

How much engineering time goes into keeping your flag system running?

- [ ] Engineers regularly troubleshoot it, fix scripts, maintain integrations, or build missing capabilities. (3 points)
- [ ] One or more engineers spend time supporting or maintaining it most weeks. (2 points)
- [ ] Maintenance is minimal, predictable, and handled through standard processes. (1 point)

**The hidden cost:** No one budgets for flag-system upkeep, but it adds up: Every hour someone spends patching the SDK, fixing the config service, repairing the broken dashboard, and answering “Why is this flag behaving like that?” tickets is an hour not spent shipping.

## Question 2: Flag ownership and cleanup

Your score:

Can you identify the owner, purpose, status, and removal plan for every live flag?

- [ ] Many flags have no clear owner, purpose, or removal plan. (3 points)
- [ ] Ownership is tracked informally through tickets, documents, or individual teams. (2 points)
- [ ] Every flag has a clear owner, lifecycle status, and cleanup process. (1 point)

**The risk:** An unowned flag is one no one dares to remove. As the system ages, more flags lose their relevance, but each one remains live in production, driving behavior nobody is monitoring.

## Question 3: Audit readiness

Your score:

If an auditor asked about a flag change today, how quickly could you provide its full history?

- [ ] We could not reliably provide it. (3 points)
- [ ] We could reconstruct it by checking several systems and speaking with individual engineers. (2 points)
- [ ] Approvals, configuration changes, and audit records are centralized and immediately available. (1 point)

**What it costs you:** When your flag history lives in Slack scrollback and one engineer’s memory, a simple audit question turns into days of digging through logs and pinging engineers. You need the who, what, and when of every change captured as it happens, not reconstructed under deadline.

Question 4: Key-person dependency

Your score:

What would happen if the person who built or maintains the system left?

- [ ] We would lose critical knowledge and might struggle to operate or update it. (3 points)
- [ ] A small number of people could take over, but there would be a significant learning curve. (2 points)
- [ ] Nothing. Ownership is distributed, and processes are standardized, so no single person is a point of failure. (1 point)

**The exposure:** A flag system only one engineer understands is one resignation away from becoming a liability nobody can safely change. It works fine right up until that person leaves.

Question 5: Multi-team scalability

Your score:

How easily can a new engineering team start using the system?

- [ ] Adding a new team requires custom development, manual setup, or significant support. (3 points)
- [ ] Teams can adopt it, but they usually need help from the original developer or platform team. (2 points)
- [ ] Teams connect the SDK, configure access, and start using governed flags through a standard process. (1 point)

**What breaks at scale:** A flag system built for one team rarely scales to a second one easily. The next team either has to wait on the one engineer who understands it, fork its own copy, or start flipping flags blindly.

Question 6: AI-velocity strain

Your score:

As AI-assisted development speeds up how fast changes reach production, can your flag system keep up and keep control?

- [ ] AI is pushing more changes than we can safely flag, leading to more production incidents. (3 points)
- [ ] We're keeping up for now, but faster releases mean more manual coordination and more chances to miss something. (2 points)
- [ ] Progressive rollout, targeted exposure, and instant rollback are automated and keep pace no matter how fast we ship. (1 point)

**Why speed makes it worse:** AI ships more code faster, and every change is a potential flag. In a manual feature flag system, every minute a bad change stays live exposes it to more users.

| Question | Your Score |
| --- | --- |
| Q1: Engineering maintenance | |
| Q2: Flag ownership and cleanup | |
| Q3: Audit readiness | |
| Q4: Key-person dependency | |
| Q5: Multi-team scalability | |
| Q6: AI-velocity strain | |
| **Total:** | |

## What your score says about your flag system

Add up your points, then read the result that matches your total.

### 🔴 14-18 points: You need a dedicated tool soon

What was once a simple flagging utility has become a full feature management system your team has to maintain. It leans hard on engineering intervention and one or two people's knowledge. Expect recurring interruptions, unclear ownership, slow audit preparation, and rising key-person and release risk.

### 🟡 10-13 points: You're outgrowing it faster than you think

The core functionality works, but gaps in governance, visibility, or scalability keep pulling engineers away from customer-facing work. More teams, faster releases, greater flag volume, and AI-generated code will only raise the pressure, surfacing as rising maintenance costs, stale flags, manual audits, and inconsistent practices between teams.

### 🟢 6-9 points: Your homegrown system still works

Your feature management practices are standardized, traceable, and scalable. Teams control exposure without constant engineering interruptions, leaders see production changes as they happen, and releases ship faster and safer without leaning on any one engineer.

## What to do with your score?

- **If you scored above the bottom band** → add up the time your team spends maintaining the system each month. That number is your business case for a dedicated tool, or the start of one.
- **If you're in the bottom band** → plan to revisit this scorecard as your team grows.

CloudBees

# What a focused feature management tool solves

Building your own flag system made sense at first. But the more teams and flags you add, the more time your engineers spend keeping it alive. And it only takes one failed audit, or the one person who understands it quitting, to turn it into a real problem.

## That’s what CloudBees Feature Management handles for you:

- Installs into your stack in a few steps. Works with the CI/CD you already run.
- Teams manage their own flags safely, with no one on-call to flip a switch by hand and no cleanup scripts to maintain.
- Progressive rollout, an instant off-switch, and a full record of who changed what.
- Focused feature management, not a big platform to buy into.

CloudBees gives your team back the hours lost to internal tooling, makes audits routine, and keeps maintenance costs from climbing as you scale.

### Ready for a closer look?

A Feature Flag Governance & Risk Assessment prices what your homegrown system really costs to run, and shows what a focused tool would save, without touching your CI/CD stack.

Explore Feature Management  Book a demo

CloudBees

# AI prompt add-on

Paste this prompt into your AI tool of choice for a personalized cost-and-gap estimate.

```
You are a DevOps advisor helping me evaluate the true cost and
risk of my team's homegrown (in-house) feature-flag system. Base
your analysis only on the details I provide and clearly label any
assumptions.


Here is my setup:

- Engineers involved in maintaining the flag system: [e.g. 2 part-
time]

- Approximate hours per month spent on flag maintenance, upkeep,
and support: [e.g. 20]

- Number of active feature flags in production, and roughly how
many have a clear owner: [e.g. ~150 flags, ~60 with a known owner]

- Our audit / compliance situation and release cadence: [e.g. SOC
2 in scope, we deploy to production several times a day]


Using this, please give me:

1. A rough estimated ANNUAL cost of maintaining this system,
showing your math (use a fully-loaded engineer cost of ~$150K/year
unless I gave you a different figure) and covering both direct
maintenance hours and the risk costs of ungoverned flags, key-
person dependency, and audit exposure.

2. A prioritized GAP LIST across these six areas: engineering
maintenance, flag ownership/cleanup, audit readiness, key-person
dependency, multi-team scalability, and keeping pace as AI speeds
up releases. Rank by which exposes me most, with one concrete first
action for each.

3. A one-line verdict: Is my homegrown system likely still fine at
my scale, close to outgrowing it, or something I need to move off
soon?

Be direct and specific. Don't recommend any particular vendor.
```