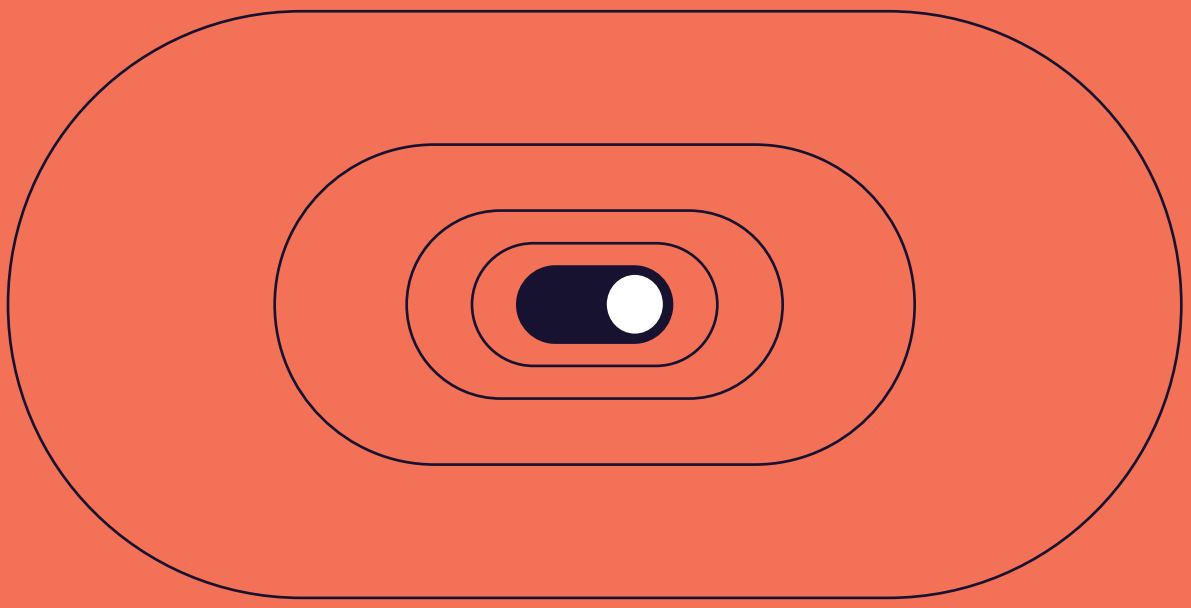
Flagsmith

SCALING FEATURE FLAGS

A Roadmap for Safer Releases & Faster Development



In Collaboration With OpenFeature

Contents

"Our biggest challenge right now is modernisation—what tooling can help and where do we start?"

"How can we reduce risk in our releases without slowing down our development cycles?"

"How do we introduce feature flags across our entire engineering organisation?"

"How are other large organisations using feature flags?"

These are questions we get a lot.

Questions that often kick-start really interesting conversations with software architects, engineering leaders, and developers working in regulated industries. The problem? They're all happening behind closed doors.

This guide throws open those doors, exploring these questions and serving as a roadmap for introducing feature flags into large, regulated, or complex organisations—and then scaling them.

Foreword

MOVING FAST, WITHOUT BREAKING THINGS

Modern software delivery with feature flags

Pete Hodgson
Fractional CTO & OpenFeature
Governance Board Member

OpenFeature

No matter what industry you're in, the pace of change keeps ratcheting up. Organisations see increased competition from both established incumbents and nimble upstarts, while at the same time the external landscape these businesses operate in seems increasingly turbulent—regulations change, supply chains are shocked, revolutionary technology appears seemingly overnight...

> **"It turns out that, when you use the right practices, you can move fast, with safety."**

To respond to all this change, enterprises need to become more nimble—they need to be able to make quick decisions, experiment, and adapt. And given the increasingly digital nature of every enterprise, this naturally leads to higher and higher expectations around the pace of software delivery.

But while expectations around the pace of software delivery are increasing, expectations around quality, security, and compliance remain the same, particularly for larger enterprises operating in higher regulatory environments.

On the face of it, this appears to be an unresolvable conflict. Surely accelerating the pace of change means accepting a trade-off in terms of risk—higher risk for bugs and errors slipping past QA, higher risk for deployment incidents, higher risk for security vulnerabilities. "Move fast and break things", as the saying goes.

Happily, that's not necessarily the case. Over the last couple of decades, leading software organisations have demonstrated that accelerating the rate of change can actually reduce risk. It turns out that, when you use the right practices, you can "move fast, with safety".

This claim—that a higher pace of software throughput can lead to greater stability—may provoke some healthy scepticism from yourself or your peers. It certainly sounds a little counterintuitive. However, this idea is backed up by solid empirical research, as detailed by Dr. Nicole Forsgren and her DORA team in their book *Accelerate*. This research group applied statistical modelling techniques to data from over 23,000 survey responses, collected over the course of multiple years, to try and understand the practices that organisations around the world use to build software, and how well those practices work.

One of the most interesting findings from this wide-ranging group was the correlation between a higher pace of software throughput and higher levels of stability and quality. It's also worth noting that their findings didn't seem to be strongly affected by the size or type of organisation—big or small, regulated or not, the same correlations were evident.

The most exciting part is that the DORA team didn't stop at just looking at these performance metrics; they also looked at the practices these different

organisations were using. They identified a specific group of "high performer" organisations who were differentiated on a key set of software delivery metrics, and most importantly they were able to lay out a set of software delivery practices that these high-performing organisations used to achieve speed with safety.

The obvious question is, what are these organisations doing that allows them to accelerate their pace of software delivery while still maintaining quality and stability? In large part, we can say that their success lies in a set of practices that accelerate the feedback loop between code being written and code being deployed— a set of practices often referred to as Continuous Delivery.

This cluster of techniques works together to achieve an interconnected set of principles:

- Reduce manual work in testing and deployment activities
- Work in small batches
- Keep codebase in a deployable state
- Architect systems for decoupled deployments
- Make frequent deployments to production

What the DORA research showed us is that organisations that focus on these Continuous Delivery practices are able to move faster than their competitors, while still maintaining safety and quality.

One practice that is near-ubiquitous within these high-performing organisations is feature flagging, something you'll no doubt be happy to hear since you're holding this eBook in your virtual hands.

Feature flagging facilitates a central tenet of Continuous Delivery: separating deployment from release. If a team wants to work in small batches and make frequent deployments to production, they are confronted with a challenge: what to do with work that doesn't fit into one of those small batches. Let's say the team doesn't want to accumulate more than a week's worth of code changes without a production deployment. What do they do if they have a feature which is going to take two weeks to implement? The answer is that they incrementally merge and deploy their changes all the way to production, despite it still being a work-in-progress. That's ok though, because the changes are deployed as latent code, inactivated behind a feature flag. Then, once the feature is fully implemented (and tested) the feature flag is flipped on and the feature is released. This is what we mean when we say deployment is separated from release—the implementation of the feature is deployed incrementally, but the feature is only released when ready.

We shouldn't stop here. While Continuous Delivery can transform a team's way of working, feature flagging can do a lot more. It's also a central part of practices like A/B testing and experimentation, helping to level up the entire product delivery process. On the operational side of things, feature flagging unlocks advanced techniques like canary releasing and dark launching; another way for organisations to move faster, with safety.

I first started talking to engineering leaders about feature flagging in 2016, and have served on the OpenFeature governance board for the last three years. Over the past 10 years, I've had the opportunity to see first-hand how, as companies navigate an increasingly challenging and dynamic landscape, feature flags keep on showing up to enable a practical balance of speed and safety. It's been gratifying to watch the adoption of these practices grow from a quirky idea evangelised by some upstarts in Silicon Valley to a very mainstream technique suitable for even the most mature engineering organisations.

I've also seen that these benefits don't come for free—feature flags do have a carrying cost, and can contribute their own technical burden if not managed intentionally. This is especially true at scale and for large organisations. If, however, your feature flag adoption is done strategically, with practitioners learning the correct technical approaches along the way, then the agility and speed that can be gained is undeniable. The following pages provide a roadmap for how to safely and effectively do this at scale.

WHY FEATURE FLAGS?

Feature flags aren't a new phenomenon. Notable early converts include companies like Flickr, Etsy, and Facebook, with Martin Fowler and Pete Hodgson writing this foundational 2017 article that still influences thinking today.

As their popularity has grown, companies operating in complex environments have started to see the benefits. At a high level, they can:

- Remove development bottlenecks
- Minimise risk around releases
- Speed up time-to-market for new features
- Improve end-user experience (by significantly shortening feedback loops)

Of course, this doesn't happen from simply introducing a few "if" statements into your codebase. Feature flags are more of a social innovation than a technical one. Their real magic is revealed when flags are introduced to an organisation at scale—impacting the very way software gets built. The technology is only half the battle. Determining how to use it, who can use it, and how to drive adoption successfully is just as important.

Because feature flags live deep in your codebase, it's important to introduce them in the right way, avoiding locking you into specific vendors—or bad habits.

As large organisations modernise their development practices, introducing metrics like DORA or SPACE, Continuous Delivery has taken centre stage. Feature flags play a natural supporting role here, serving as prerequisites for adopting CD practices that unlock faster release cycles, better code quality, greater developer productivity, and increased team collaboration.

Feature flags are more of a social innovation than a technical one. Their real magic is revealed when flags are introduced to an organisation at scale—impacting the very way software gets built.

Banks, insurance, healthcare, and government agencies face a unique set of challenges when it comes to modernising their development practices, operating in highly-regulated environments with release practices that can feel like they're set in stone.

The following pages provide strategies and examples used by some of the largest enterprises in North America and Europe. The goal is to share practices that can help introduce feature flags at scale, allowing companies to introduce modern development practices that give developers greater flexibility and speed, while increasing the safety of releases.

BUILDING A STRONG FOUNDATION FOR FEATURE FLAGS

How to fight vendor lock-in: Insights from OpenFeature

Michael Beemer
OpenFeature Governance Board Member

Feature flags have become essential to modern software delivery, but implementation across your enterprise presents a critical choice: how should feature flags be managed? Many options exist, ranging from homegrown solutions to third party vendors. Until now, each choice required developers to use solution-specific feature flagging patterns, locking organisations into their choice and presenting a high switching cost to explore other options.

Following in the footsteps of OpenTelemetry, OpenFeature, a CNCF incubating project with broad industry support, solves this problem by providing a vendor-neutral, open standard for feature flags. Your development teams write code against one consistent API, while your organisation maintains the freedom to use any provider. Many organisations report that introducing OpenFeature's abstraction layer is a key step in migrating from disparate internal solutions to a consolidated feature flagging provider, or even abstracting multiple different providers behind a shared interface.

Why vendor-neutral standards matter

Many enterprises face a common challenge that compounds as they scale: feature flags exist in multiple forms across the organisation, from dedicated vendors to multiple homegrown feature flagging and configuration management solutions. This fragmentation creates inconsistent developer experiences and complicates governance, especially in regulated industries where oversight is paramount.

By providing an abstraction layer between your application code and feature flag providers, OpenFeature addresses those challenges by offering:

- A consistent developer experience regardless of the underlying implementation
- Common terminology through a standardised glossary that aligns communication regardless of the tool or vendor
- Flexibility for different teams to use their preferred providers while maintaining standardisation
- Seamless migration between providers as your needs evolve without disrupting operations
- Streamlined governance through consistent policies and monitoring across all feature flags
- Reduced training costs as developers learn one pattern that works across projects and teams

OpenFeature in practice

Consider a large financial institution with multiple business units, each with its own feature flag solutions. By implementing OpenFeature, they can:

1. Standardise how developers interact with feature flags in code
2. Gradually consolidate to enterprise-wide feature flag platforms where beneficial
3. Maintain flexibility where different teams have unique requirements
4. Enable consistent governance and reporting across all implementations

For organisations undertaking modernisation initiatives mentioned throughout this guide, OpenFeature provides a critical path for successfully scaling feature flags across your organisation while avoiding the pitfalls of vendor lock-in.

People & processes

Doing a good job of introducing any new technology into a large organisation starts with people—the people behind the technology and the internal teams who will adopt it.

Driving change in large organisations is an iterative approach—it won't happen overnight and it doesn't need to be overwhelming, especially if you take the time to plan out your implementation. Keep these steps in mind when approaching your planning:

1. Ensure that company leadership is on board—transformations driven by executive sponsorship tend to have a higher rate of success
2. Identify a champion or working group who can own the implementation, help with internal onboarding, and work directly with your feature flag provider to get up and running
3. Decide if a pilot programme is needed
4. Come up with a plan for a phased rollout (more on this on page 17)
5. Map out how you are currently releasing new features and who is part of the process
6. Identify any existing feature flagging solutions currently in use—they might be disguised as configuration management

Map your current release strategy

Introducing feature flags at scale often happens as part of a wider modernisation project, which means you may have already done things like identifying the different players and processes in your development cycle. If not, teams often find this to be a valuable exercise when preparing to introduce feature flags. Think about:

People

- Which teams are involved?
- How do product and engineering teams currently work together?
- How do the engineers and product managers work together on specific branches?
- Who has the ability to deploy vs. release?
- Who else in your organisation outside of engineering and product is involved in releases?

Processes

- How do your engineers currently do branch management?
- How does a new feature move from idea > iteration > deployed > released?
- Which parts of this process add value?
- What are the bottlenecks? Do certain steps require approvals?

Introduce change thoughtfully

By introducing feature flags into your software development lifecycle, you're fundamentally changing the way teams build software. Your engineering, QA, product, and DevOps/infrastructure teams will experience a massive change in the way they work together (often moving from long, sometimes painful branches to shorter branches that allow for more flexibility). Introducing this change in a thoughtful manner will mean bringing together stakeholders from each of these functions for rounds of feedback.

Bringing on feature flags will look slightly different for every organisation, but there are some best practices you can keep in mind (we'll go over a few on page 10). Many teams also start small, whether with shovel-ready use cases or a pilot programme, and then build from there.

An important part of a successful company-wide adoption will be securing time: the time of your champion or working group and the time of any teams piloting this new way of working. Budget for this up front and create a plan with specific milestones and owners—as well as a strict timeline, so that the project keeps moving forward. Ensure this plan is shared at a company level, so that those involved are motivated to stay on time and on track.

TAILORING FEATURE FLAGS TO YOUR ORGANISATION'S NEEDS

As you begin to determine how to introduce feature flags into your release processes, ensure your team is familiar with the different kinds of flags as well as best practices to keep your flags healthy and reduce tech debt.

Create governance for how you will use feature flags

Flags can become an anti-pattern if implemented incorrectly, so keep in mind their different types and lifecycles as you begin to determine how your organisation will use them. We like to think of feature flag governance as a series of layers—every type of flag will look different depending on your infrastructure and development practices and the level of security you need.

Set roles & permissions

Data security is top of mind for every company, but organisations operating in regulated industries are held to a higher standard.

Determining feature management access levels will be part of the conversation as you begin to introduce feature flags at scale. It's important to find the right balance between rigorous security standards and productivity. You want to avoid creating unnecessary bottlenecks.

One way to manage this is by creating roles and permissions for users and taking advantage of features your provider has, like change requests and feature approval workflows.

This means that individuals will be given different levels of control and have their usage restricted based on need. Just like any other software permissions in your business, the best practice here is to only give users the level of access necessary for them to do their jobs.

You'll also want to think about, and define, how different types of flags will require different levels of security within your organisation. High-stakes experiments, for example, should always require two users, a "4-eyes" sign off, with the appropriate permissions to approve changes. As another example, "break glass" feature flags can be used in special circumstances to allow immediate, emergency-level access to a critical function or behaviour, typically bypassing existing permissions.

Enable audit trails

Once you've decided who can make changes and where, it's still important to track changes that do get made. Recording any features that are created, changed, or deleted, as well as who took those actions and when, is especially important if you have a large number of developers working in different teams. Having this information can save a lot of time and confusion in knowing what has been done, when, and by whom. For companies operating in regulated industries, audit logs are also an important part of compliance.

5 FEATURE FLAG BEST PRACTICES

1. Clean up your feature flags!

The best way to get the most out of feature flags (and enjoy using them while you're at it) is to clean them up after they've served their purpose. Create a built-in process for flag archiving, making this a regular part of your engineers' workflows, just as important as reviewing pull requests or clearing issues in your issue tracker. This will help manage tech debt and save your team time and messiness down the road. Part and parcel of this is understanding the difference between long-lived and short-lived flags and when to use them (more on flag lifecycles soon).

2. Design features around flags

Feature flag deployment should be planned early on in the software development process. If flags are simply layered in as an afterthought, they won't be as effective. Whenever you start work on a new feature, think about how you can put it behind a feature flag. The possibilities of feature flags are endless, which makes it even more important to define a flag's scope.

Start by defining exactly what each flag will do when:

- Disabled/enabled
- There's a remote configuration value

Additionally, think about the rollout plan. Will you roll the feature out all at once, to segments, or to individuals?

3. Make flags as small as possible

Multitasking is great, but not when it comes to feature flags. Keep your feature flag scope specific. Start small—on/off flags are the simplest way to get started. When you're comfortable with this, you can introduce more advanced capabilities.

4. Flags shouldn't replace business logic

Feature flag values shouldn't be used as a replacement for business logic. If you create a situation where flags are interpreting code with branches that interpret rules, it becomes very easy to make a mistake. Feature flag software is not an integrated development environment. If you treat it like one, you'll lose a lot of the benefits of just writing code and being able to test it properly. For instance, you shouldn't use feature flags for access control. If you feel that you're using your feature flags to guard your data and segregate between different types of users, you should probably use business logic instead.

5. Create a naming convention

Having a well-defined naming convention is important for two reasons: One, better team collaboration: users will understand what happens when a flag is turned on/off from its name alone. Two, it will remove the burden of decision every time you introduce a new flag and have to name it. As Phil Karlton said, "There are only two hard things in Computer Science: cache invalidation and naming things."

USING FEATURE FLAGS FOR SAFER RELEASES & FASTER DEVELOPMENT

It may seem counterintuitive to introduce greater flexibility and speed into your release practices when stability and predictability are paramount. Yet, predictability has a downside: if your company's deployment practices involve long development cycles that are predictably slow and painful, predictability isn't doing you any favours, it's likely blocking innovation and increasing the risk of something breaking along the way. And, as Pete Hodgson points out in his foreword, with the right development practices in place, it's now possible to move fast, with safety.

Feature flags can help you reduce risk, improve development velocity, and shorten feedback loops, strengthening your product offering.

To explore what this looks like in practice, in late 2024 we conducted a survey of our Enterprise customers in order to understand the ways feature flags are impacting their development processes. The results revealed three main areas of impact: risk reduction, development velocity, and experimentation.

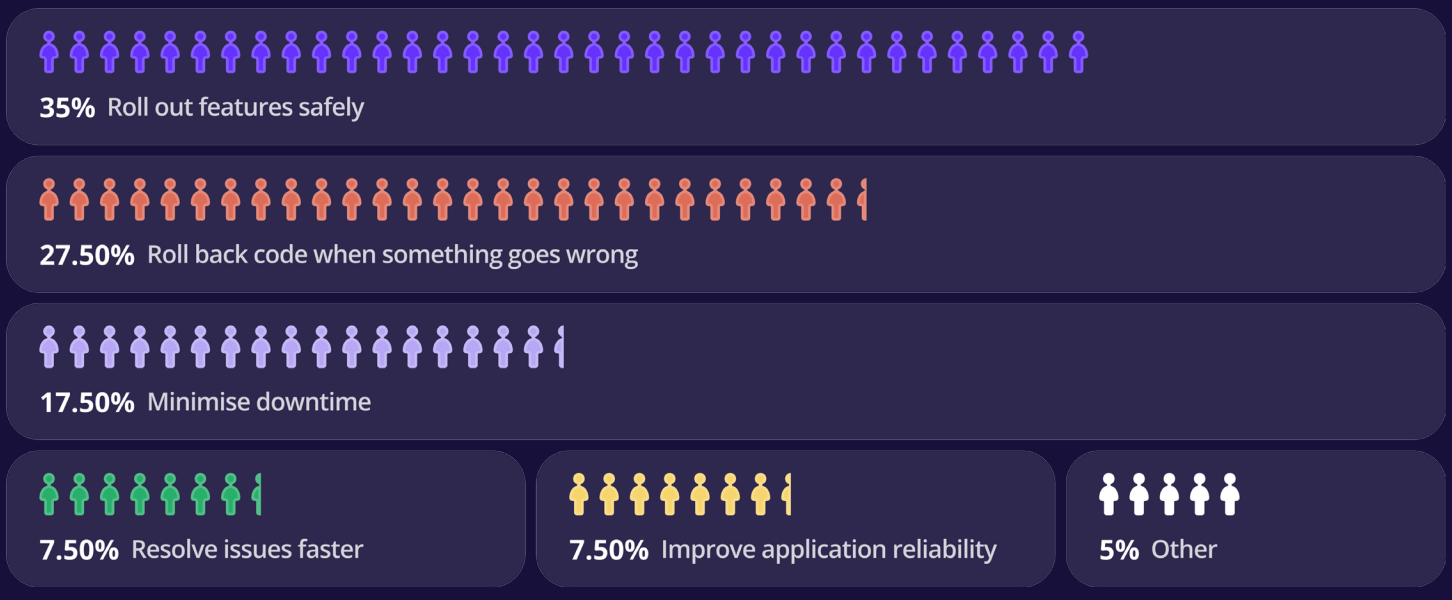
Risk reduction

84%

Releases feel **less risky**

What does this actually mean?

Feature flags reduce risk in a really simple way. Wrap code in a flag, roll it out to users in a controlled way (e.g. with canary deployments) and see how it performs. If anything goes wrong, catch it in real time and roll back without needing to redeploy.



In software development, every new feature is a leap into the unknown and brings up questions like: will this work as intended, is it going to be well-received by the users, will it introduce new bugs? Feature flags form a shield against these uncertainties by reducing exposure to risk via:

Canary deployment

A canary deployment allows you to release code to a small subset of users before gradually scaling up after checking its viability.

Kill switch

If something is broken in your code and it's wrapped in a feature flag, you can immediately disable it by turning the flag off with the click of a button. This has an added benefit of allowing you to continuously progress in your code commits, rather than having to remove the code at issue or redeploy. This way one issue doesn't stall the whole development path in its tracks. Plus, you can roll it back instantly rather than expose more and more users to buggy code.

Faster recovery

When things don't go as planned, instead of rolling back an entire deployment or trying to cherry pick what to roll back, you can just toggle the offending feature off. This keeps your services stable and your users happy. And if you need to isolate an unstable or underperforming feature during incidents (for example, downtime, cyberattacks, etc.), you can do so without bringing down the entire application.

Development velocity

50%

of customers that released **every month** before Flagsmith, now release **every day** or **every week** using Flagsmith

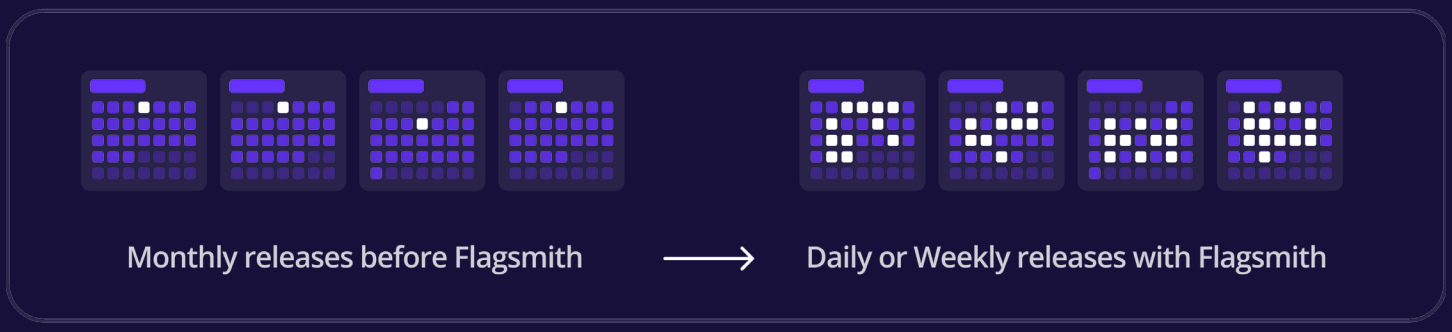
By separating deploy and release, feature flags enable individual developers and teams to simply push code when they're ready instead of waiting around for the next release window.

This prevents:

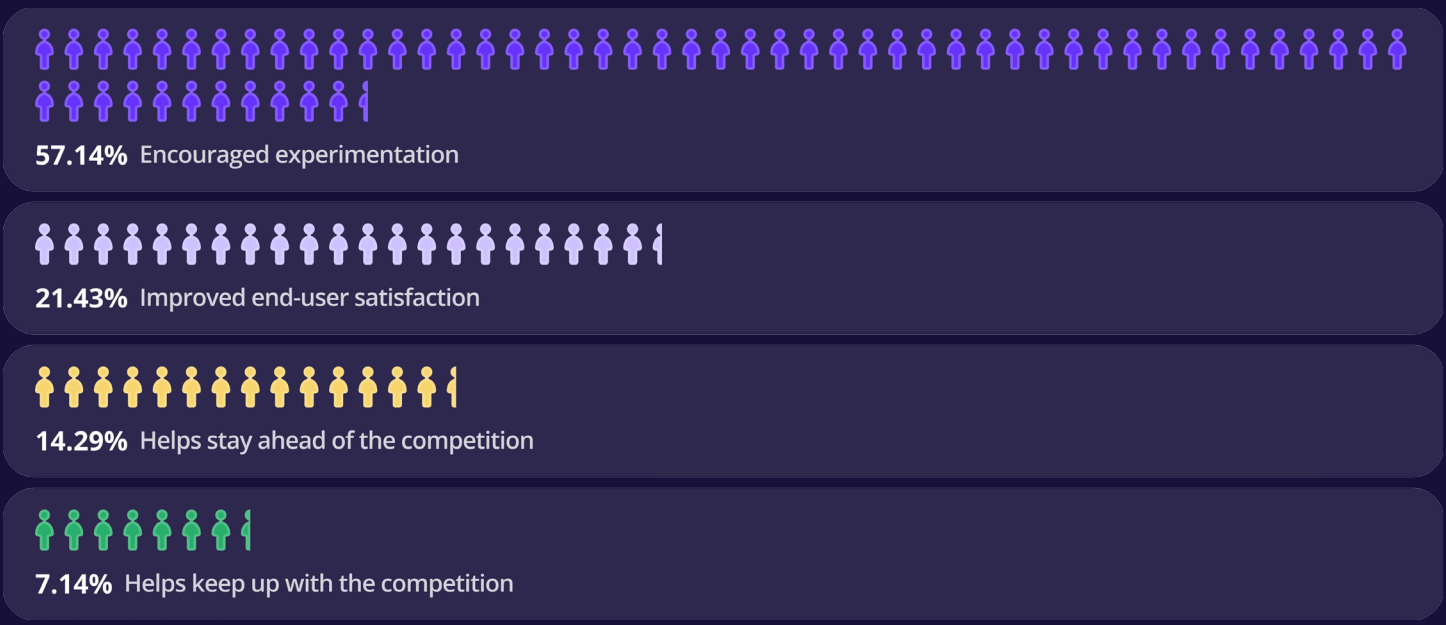
- A buildup of extra stress on the current release window, likely causing rushed code and potential bugs
- Putting extra stress on the next release by having a buildup of extra unreleased code on top of what you have planned
- Extra stress on developers as releases get bigger and bigger

What does this actually mean?

By decoupling deploy from release, developers can push their code when it's ready. Code quality goes up as developers push code they're confident in, and risk goes down. For a chunk of teams, this leads to more frequent releases. For the other chunk, regulations mean monthly releases are still standard (though many will still deploy more regularly, especially to pre-production environments).



What is the *number one* way feature flagging has contributed to feature innovation?



Using feature flags to stay ahead

In large enterprises, release cycles tend to be long and slow, whereas the market is always rapidly evolving. Apart from freeing up engineering time for writing code, working with feature flags allows you to balance rapid innovation with system stability. By decoupling deploy and release, your team can introduce new features behind flags and enable testing in production.

Experimentation

Feature flags give individual developers and teams the freedom to test and iterate new features with internal stakeholders, beta groups, and different segments of customers. This freedom is enabled without risk, because features are hidden behind flags. This allows teams to use data from real users to guide development strategy.

What is the *biggest impact* development teams have felt?

Feature flags allow for numerous other efficiency gains:

Reduce the number of development environments

Feature flags naturally lead towards a mono-environment methodology and cut down on the reliance on environments, letting developers release straight to production—or in the case of highly-regulated companies, pre-production. The money saved maintaining the environments can allow you to invest in the resilience of your production environment.

Combine features from previously disparate teams

Say you've got web and mobile teams that are running in isolation—working on the same features, but for different platforms. This lack of communication can lead to bugs and also a lack of parity. Having a single feature flag project can act as a talking point between these isolated teams.

Improve collaboration between technical teams and non-technical teams

In large organisations, a lot of engineering time is spent coordinating with non-technical team members like product managers (and vice-versa). This could look like product teams relying on engineers to manually adjust code to run a beta test or try targeting a specific user segment for a feature rollout. By giving non-technical team members the ability to manage feature releases, this massive bottleneck disappears, freeing up engineering time for building your product.

EASY REFERENCE GUIDE FOR FEATURE FLAG TYPES & USES

Short-lived feature flags

Short-lived feature flags enable you to integrate new code into your main codebase while keeping it inactive until release. For example, you can use a feature toggle if you want to hide a new transaction feature during testing. When you're done testing, switch it on and release it to all users before removing from your code, or setting a reminder to.

Long-lived feature flags

Long-lived feature flags remain in the codebase for extended periods—or permanently. They encompass kill switches, and support for multitenant architectures. You need to be careful and ensure they aren't removed since they're critical to the stability or usability of your application.

User-based feature flags

User-based feature flags control feature visibility based on user attributes or segments. Let's say you're releasing a new tax filing feature in a government portal. You can release it to only small business users or sole proprietors before releasing it to your entire user base.

System-based feature flags

System-based feature flags control functionality based on technical or environmental factors—not user attributes. They work well in cases where you use different deployment environments or system conditions. For example, a finance app could use these flags to deploy region-specific compliance features based on the server's geographic location.

Release Flags

Release flags can control the deployment of new features to production environments. They allow teams to deploy and activate code separately, ultimately reducing the risk associated with new releases. For instance, you can use them to gradually roll out certain features automatically when they hit certain thresholds.

Experimentation Flags

Experimentation flags allow you to run A/B tests or any experiment in your application. They're meant to help you measure the impact of specific changes you make. As a result, you can make better decisions about which combination of variables works better for your user base. Let's say you want to understand which button colours increase clickthrough rates in your app. You can run two or three versions of the button colour and analyse which ones drive the most action.

Operational Flags

Operational flags manage system behaviour during runtime. So, they serve as an emergency control or a way to optimise performance in real time. If you run into production issues, these flags let you respond without redeployment or code changes. For instance, if your app experiences high traffic, you can add thresholds to switch off certain functions to reduce the traffic burden on the app.

How are large organisations bringing feature flags on at scale and fighting vendor lock-in?

EBAY CASE STUDY

Migrating to OpenFeature at scale: 0 to trillions of calls

With Chetan Kapoor & Justin Abrahms

This case study is a condensed version of a talk that Chetan and Justin gave at the OpenFeature North America Summit in 2024. You can watch their full talk here.

~2.1B	132M	190	$12B
Live Listings	Active Buyers Worldwide	Markets	Mobile Volume

Stats as of Q2 2024

Where it started

eBay was navigating challenges familiar to many large organisations—long development cycles, riskier-than-necessary deployments, and a siloed problem-solving culture. The statistics above offer a little insight into the scale of these challenges.

Meet the Velocity program: Faster delivery matters!

A 300-team cross-organisational effort staffed with forward-looking principal and senior principal architects and key product leaders, the Velocity program was built to succeed where other modernisation efforts had failed. The program's northstar mission was to make software and product delivery a competitive advantage for eBay.

In practice, this looked like a ground up reworking of how eBay thought about feature delivery. How to make it safer, easier, and faster, driven primarily by the DORA and SPACE methodologies and focusing on three main levers to find success:

1. Executive sponsorship (CTO and CPO sponsorship): Securing executive sponsorship enabled a mindset of, "go figure out the problem and then fix it"
2. Very clear directive: Making software delivery a competitive advantage (this was easy to apply when deciding if something was in or out of scope)
3. Budget and time to make this happen: Each engineering team was allocated a percentage of their total working time to use for the velocity initiative

Amongst other objectives, the program aimed to solve three key problems in eBay's developer ecosystem

1. Code was stuck in review—a slow review process meant developers were stuck waiting to merge code, so pull requests became larger and larger, ending in slow feature development and riskier, large-scale deployments

2. Multiple configuration platforms targeting different sections of eBay's stack. Mobile had their own feature flagging tool, there were a few different feature flagging tools being used by backend teams, and some teams were also making use of an experimentation framework to create feature flags

3. A culture of DIY rather than a platform mindset that could save teams from reinventing the wheel

Weighing up solutions: Build vs. buy

Looking at these challenges, the team saw the adoption of a unified feature flag platform—and similarly a unified way of using feature flags—as a possible solution to these problems. They weighed building their own vs. buying. In the end, the decision came down to their experimentation needs. eBay has a very mature experimentation engine that operates at scale, and leadership wanted a straightforward user experience that would seamlessly blend the two. So they chose to build and add feature flags to their in-house experimentation platform.

However, as the Chair of the Open Source program, Justin knew that building something bespoke instead of using an open-source solution felt risky in terms of future-proofing. So they adopted OpenFeature to provide an open-source facade layer. In the backend they would then use the internal solution, knowing that if this internal system didn't work out, they could always introduce a vendor later on without having to rewrite the code.

Building a solution: Enter, eBay's experimentation platform

The in-house eBay experimentation platform is used to A/B test most features before they are broadly rolled out to customers, fostering a "test and learn" culture that has served the company well.

Rolling out feature flags on a platform that already had a very high internal adoption rate seemed like the ideal strategy. "The idea of incorporating feature flags into this flow just felt right", says Chetan Kapoor, the Head of Product and Chief Evangelist for Feature Flags at eBay.

The following steps were taken to enable feature flags on the platform:

1. Used OpenFeature SDK
   - Authored eBay-specific provider and hooks
   - Benefit: One API that eBay used for experimen tation and feature flags, so developers could get twice the value from one API, and were able to sunset 25+ legacy APIs
2. Added platform capabilities as they moved past MVP
   - Kill switch: allow site reliability to stop incidents with flags within 1 minute.
   - Audience-aware targeting: A granular way of doing customer-based targeting. A simple example of this would be, "only show this feature to users in the UK, using Google Chrome, with sneakers in their carts".
   - Latency guarantee: developers requested best-in-class evaluation latency.
3. Simplified onboarding journey
   - Removing blockers so that all engineers could and would use the platform for feature testing (improved internal documentation)
   - Hiring a Technical Program Manager to work on simplifying onboarding and proselytising across the company

Building a MVP and driving adoption at scale

Successfully building, iterating, and driving the adoption of a new technology across a company as large as eBay requires a high level of strategy as well as buy-in from key stakeholders and influencers at every level of the organisation.

Identifying initial influencers

To kick off the adoption programme, the team started with three key developer stakeholders. They leaned into education and topics like:

- What are feature flags

- Why should you care about them
- When should you use them—and when should you not
- Which stakeholders can answer your questions

Identify internal influencers and try to pilot with a diverse and excited group of developers.

Gradually, as the team collected and iterated on initial feedback from these developers, they prepared to scale it to other teams.

Scaling the pilot programme

They identified 21 different teams to run multiple pilots with to learn and create the roadmap for a feature flag MVP. They did three rounds of pilots over three quarters, focusing on building product-market fit internally:

Milestone 1: Make it possible for people to use flags

Milestone 2: Make it useful to use feature flags

Milestone 3: Make that process scaleable

Finding success & building the MVP

Once they had completed the final round of feedback, they started preparing the MVP, which, once ready, they had to roll out to everyone. No small feat at a company the size of eBay. The team driving adoption consisted of: Chetan Kapoor, Justin Abrahms, a program manager, a senior architect, and a lead engineer. Along with that dedicated team, they had a 5-step strategy:

1. All the different teams that needed to migrate needed a real budget. They secured 10% of the Velocity budget
2. Simple onboarding instructions and demos
3. Get a commitment: share your plan and publicly commit to it
4. Make sure the migration experience was self-serve (the dedicated team had a dashboard to monitor the migration progress, so they could see which teams were lagging behind—this also served as an easy way to troubleshoot and debug if people came with questions about the migration being complete, etc.
5. Internal marketing campaign via newsletters and escalation emails

How did it go?

Feature flags have fundamentally altered the software delivery landscape at eBay. The company has been able to shorten development cycles, reduce risk, and increase developer productivity. The engineering and product teams can now collaborate, test and iterate on new features in an agile way that supports the company's culture of experimentation and continuous improvements.

- Billions of calls made per day to the feature flag evaluation engine
- 2500+ experiments behind flags every year
- Rolled out to thousands of developers
- The 15-minute change propagation is now down to 1 minute
- The 25 millisecond evaluation latency (deciding if the feature should be shown or not) is down to <5 milliseconds

WRAPPING UP

Feature flags allow large enterprises operating in complex environments to move with an agility that was previously unattainable. By releasing development teams from deployments that are tightly coupled to releases, engineers, QA, and product teams, among others, can move at their own pace, increasing development velocity as well as shortening feedback loops.

For more reading, our guide *Flip the Switch: Unlock Modern Software Development with Feature Flags* takes readers through the ROI of feature flags for large enterprises, how to remove development bottlenecks, how feature flags fit into migration projects, and more.

" Without feature toggling, we'd run a deployment, do a check, and find something. Then we'd have to decide to live with it or roll it back. Rolling back meant a wasted night. With Flagsmith, we can turn things on or off individually on a feature level, so it's just really made things smoother for us."

- Senior Manager, Software Engineering, Fortune 500

Flagsmith

Release with confidence

Flagsmith is an open source feature flag software that gives developers peace of mind. We work with data-sensitive enterprises across the world, offering self-hosting and private cloud deployments, features for maximum security, and technical support to cover any needs. We also partner with OpenFeature to support open standards and prevent vendor lock-in.

Get in Touch

WE WORK WITH BANKS, HEALTHCARE, AND GOVERNMENT AGENCIES ACROSS THE WORLD

KB creditas Canva okta citi :ubank

Flagsmith.com