How Engineering Managers Can Bring Back the Joy of Building

Enabling A Culture of Experimentation with Feature Management

The joy of building can be liberating. It's like an open road, where all your carefully orchestrated actions gain traction and accelerate you towards your goal. As an engineering manager (EM), this is the experience you want for your people; you're accountable for mentoring, supporting, and empowering a team. But at the same time, the entire organization is counting on your leadership to take care of the mechanics that will get you to the destination: the best application and service possible.

You're constantly monitoring and mitigating risks, whether due to incidents in the system or user interface bugs causing customer churn. Meanwhile, you're trying to ramp up your team's speed and agility without running out of fuel. It's a lot of responsibility to take on, but it's also incredibly rewarding.

When competing pressures threaten to combust, the right tool can be the key to harnessing that energy into forward motion. Team health, business objectives, and risk mitigation might seem like separate priorities, but you can address them simultaneously by expanding your team's capacity to experiment. This whitepaper looks at why that is and how to reach maximum velocity.

Familiar Challenges Facing EMs

For starters, it's important to acknowledge the unique challenges that EMs are commonly called upon to navigate — ones that can have major implications for the health of their engineering teams and the quality of their work. These three common sources of stress for EMs may feel a little too familiar.

Challenge #1: Branch Management Complexity

Growth is supposed to be a good thing. But as the size of the engineering team increases, so does the complexity of the codebase. It only gets worse when multiple people are working on different product features, and all of these feature branches need to be merged before testing.

For EMs and engineers, this is as time consuming as it is frustrating. By the time an engineer is ready to test a change, there's bound to be a merge conflict because others on the team are checking in with changes of their own.

This all works against agility. Release trains are long term. Changes are large and complex, and they rely on manual testing by separate quality assurance (QA) teams. If something breaks, it's much more difficult for engineers to iterate and correct course. And it can take a long time to measure how changes affect the end user experience.

As any EM knows, engineers thrive on feedback. No one wants to wait months to see the fruits of their labor, but this is often the case when feature releases are infrequent.

So what can you do about it? The short answer is **trunk-based development,** which keeps the core code in a stable main branch. In this model, engineers can make and merge small changes constantly, several times a day, in a process of continuous integration.

This solves the issue of merge conflicts and creates a much faster path to production. But because it breaks up the process of adding a feature into a series of multiple commits, it creates a different kind of complexity. If you need to keep pushing partial features while also providing proper validation, a great way to control the code is through feature flags.

Deploying feature flags to enable trunk-based development brings a wide range of benefits, and here are three big ones for EMs:

01 It's a simple way to either **enable or disable newly-developed features,** providing peace of mind for leaders that any problems or defects can easily be rolled back.

02 Engineers don't have to worry about adding more branches because they can **commit new code straight to the main branch** and wrap it in an inactive code path.

03 Teams don't have to wait days, weeks, or months for feedback on the quality of the changes they commit; they can **fix issues and measure impact continuously.**

In short, this strategy for simplifying branch management can check all your boxes: boosting team health, reducing risk, and driving business results in a single, concentrated effort.

Challenge #2: Those Dreaded Weekend Deployments

Nearly every engineer and EM has experienced a deployment scheduled for Friday night or Saturday morning.

Releasing features during low-traffic periods is a necessary evil, especially in the era of cloud computing, to minimize the potential blast radius if there's a catastrophic software issue. But it also means being on call over the weekend, waiting on pins and needles for that critical moment when an outage or failure is detected.

There are all kinds of reasons why problems occur. There might be issues with the scalability of the feature. This can lead to downtime, security vulnerabilities introduced throughout the software supply chain, or a problematic difference between the environment where the feature was built and where it was released. There may just be bugs in the code. Teams know to expect the unexpected.

But the high stakes and demanding schedule exact a heavy toll on engineers. And for EMs, who are tasked with ensuring the productivity and positivity of their teams, it's a special kind of torture to watch them slowly burn out. At a time when organizations are trying harder than ever, and often failing harder than ever, to retain their top technical talent, it's critical to find a way to ease the pressure of feature releases.

EMs can take both hard (technical) and soft (cultural) approaches to ease the pressure and foster a work environment conducive to team health:

On the technical side, feature management makes it so that deployments are non-events. Code is only activated when a feature flag is being rolled out, which takes the risk out of large deployments and allows engineers to implement each new change independently. Additionally, stepping up the use of monitoring tools to screen for software errors enables teams to anticipate failures and outages.

On the cultural side, it's vital that business demands for velocity don't come at the expense of quality work. Engineers have to be able to invest the necessary time, rather than rushing to meet overly ambitious targets. Yet it's also true that slowing down can actually introduce errors, rather than eliminate them. Speed and safety shouldn't be a tradeoff. With the right tools, they can complement each other.

Feature flags can ease the pressure here too, enabling engineers to easily test changes with select users and quickly undo problematic ones. Best-in-class feature flag products even come with intelligent monitoring capabilities that notify engineers when a feature isn't performing as planned.

Challenge #3: Silos that Divide Devs from Ops

Of course, DevOps is the standard that digitally-enabled organizations aim to achieve. If the goal is to bring better applications to market faster, then does it really make sense to have one team build the software, and another to deploy and operate it? Bridging the divide between development and operations, and establishing a sense of shared responsibility, is a top business priority that EMs are often expected to run point on. But they need the resources to do it effectively.

The reality is that siloes still exist. They exist at the testing stage, where many organizations still employ separate QA teams to manually trial new changes and features, even if they've adopted a modern DevOps model in everything else. And they exist in the recovery process, where operations teams notice a problem with an application or feature but lack the tools and knowledge to do anything about it.

Rather than rolling it back themselves, they have to pass the buck onto the engineers who originally built it because there's a lack of synergy across their teams and mandates. The result is bottlenecks and rough nights for the engineers.

What's the strategy to maintain flow? To make DevOps work for the organization, you can take these three steps to break down barriers and align teams:

01 Empower everyone on the team with the knowledge, skills, and tools they need to **identify the feature causing the issue** and roll it back in isolation without fear of side effects.

02 **Embed QA processes across the development** team rather than centralizing them in a separate department, thereby encouraging agile testing.

03 Create an environment of psychological safety so that people have an ethic of flexibility and experimentation, and the ability to **validate features in production in a controlled way**, free from finger-pointing and blame.

There are many ways for EMs and business leaders to tackle these initiatives individually, though in some cases they can also be addressed collectively.

A solution like feature flags hits multiple targets here, not only enabling operations personnel to reverse code changes by flicking a switch, but also distributing the ability to monitor, experiment, and QA test horizontally across the organization.

Another important change DevOps teams are making is broadening the focus beyond mean time between failures (MBTF) to include measuring mean time to recovery (MTTR) — also known as "mean time to repair". While there's value in tracking how long it's been since the last significant breakage, it's also important to measure how quickly the team was able to identify the problem, implement a solution, and bounce back.

Making MTTR a core metric defuses the intense pressure of trying to achieve perfect deployments and emphasizes continuous improvement. That's much better for the health of engineering and operations teams. But if you're aiming to optimize MTTR, then you must equip your team with the tools for a quick recovery.

The Formula for Energized Engineers

Whether solving for too many merge conflicts, too many silos, or too many weekends waiting on disaster, there's a way that EMs can make life better for engineering teams. The solution is to invest in the tools and processes that enable robust feature management.

On a practical level, this substantially lowers the risk of releasing features, encouraging peace of mind across the organization. But this focus doesn't just involve practical and procedural shifts; it also includes cultural ones. Extending the license to independently roll out, test, and roll back features gives engineers autonomy and a sense of ownership over — and therefore accountability for — the code. That's the essence of empowerment.

And it's an area where EMs can lead the charge. If done effectively, bringing feature management into the foreground has several major benefits. Here are three that matter to both the software builders and those who manage them:

Benefit #1: Confidence

Engineers want to build, test, and innovate to deliver features faster. The only obstacles in their way are outmoded processes, archaic metrics, and cultures that assign blame.

Experimentation-focused mindsets and behaviors embolden teams to move fast without breaking things. For one, setting more regular touchpoints can bolster the team's innovation and collaboration.

✓ Pre-mortems give your people the chance to **assess and anticipate risks proactively**

- ✓ Retrospectives open the floor to **team-wide dialogue** once a project concludes, encouraging **continuous learning**
- ✓ Individual and group health checks are valued **gestures of support**

Then, when you bring in user-friendly tools for feature management, and anyone can enable or disable changes, it becomes less about individual successes and failures, and more about accountability across the team.

Benefit #2: Excellence

Engineers and developers are doers. They want to focus on the applications and features they're building, not ancillary or administrative functions. When they can offload their more tedious monitoring and measurement tasks, they're freed from the high-risk, low-reward responsibilities that suck the joy out of their jobs.

Best-of-breed cloud solutions can support this through automations that help engineering and operations teams analyze usage patterns and data.

At this point, many engineers are familiar with the four key metrics outlined by the DevOps Research and Assessment (DORA) report:

01 **Deployment Frequency:** How often an organization successfully releases to production

02 **Lead Time for Changes:** The amount of time it takes a commit to get into production

03 **Change Failure Rate:** The percentage of deployments causing a failure in production

04 **Time to Restore Service:** How long it takes an organization to recover from a failure in production

Whether they're conscious of DORA's exact framework or not, these are the fundamental criteria that engineers evaluate themselves on.

Benefit #3: Satisfaction

Engineers thrive on having a purpose. They want their efforts to contribute to the success of the business and to help end users. This is why agile workflows and a culture of constant experimentation are so valuable both for team health and velocity:

- ✓ Continuous delivery brings frequent wins that **keep engineers engaged.**
- ✓ Coherent metrics and immediate feedback provide assurance that the work **meets stakeholder needs.**

These benefits – **confidence, excellence, and satisfaction** – are achievable for engineers, but they depend on their EMs to lay groundwork for a forward-thinking, fast-growing, and experiment-friendly organization.

With the right platform, you can engage modern feature management processes to expedite delivery, empowering DevOps with robust automation and analytics to monitor releases, measure performance, and rigorously test new features.

The result is a healthy culture of experimentation and a lively business. Engineers can get back to appreciating the joy of building — and doing so with higher productivity and less risk, while delivering code to customers faster. EMs can approach team, organizational, and security priorities as complements instead of competitors. With the right combination of tools and practices, you don't have to choose between reaching the destination and enjoying the journey.

split®

What a Relief. What a Release.

Learn how Split Feature Management and Experimentation can help your team reimagine software delivery.

Schedule a demo with us or visit **split.io** to learn more.