TEXT ONLY

Software Managers' Guide to Operational Excellence

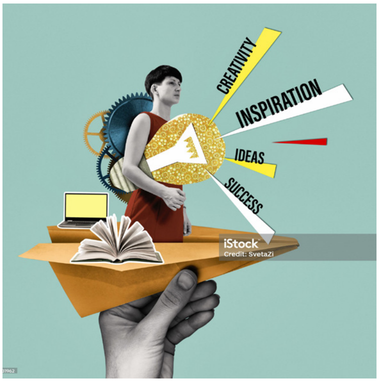
THE SECRET TO BEING A GREAT ENGINEERING LEADER? SETTING UP THE RIGHT CHECKS AND BALANCES.

KATE MATSUDAIRA

To be a good software leader, you have to give your teams as much autonomy as possible. However, you also have to be ultimately responsible (especially when things go wrong). One of the hardest things about being the manager is owning responsibility for everything but having no direct control.

The way great managers solve this is by setting up processes, tools, or mechanisms that provide insights. These allow them to ask the right questions (at the right time), and gently steer the team in the right direction.

Software engineering managers (or any senior technical leaders) have many responsibilities: the care and feeding of the team, delivering on business outcomes, and keeping the product/system/application up and running and in good order. Each of these areas can benefit from a systematic approach.



The one I present here is setting up checks and balances for the team's operational excellence.

Operational excellence is the ability to consistently deliver high-quality products and services to customers. It is essential for software engineering managers because it helps them ensure that their teams are able to meet the needs of their customers.

There are many benefits to operational excellence, including:

- Increased customer satisfaction
- Reduced costs
- Improved efficiency
- Increased innovation
- Improved employee morale

OPERATIONAL EXCELLENCE CHECKLIST

If you are taking on a new team or want to improve the way your current team works, this is a checklist and some best practices I have used in organizations I have led. Keep in mind this isn't meant to be comprehensive, and you should plan to adjust the list based on your team, your goals, and your timelines.

Verify launch plans

Most incidents occur from bad code pushes or other changes to the environment. As a leader, you should make sure you have visibility into launches and that the team doing the launch has done their homework. For example, consider these items:

- *Does the team have monitoring and dashboards?* It is not enough to have the instrumentation; you need to verify it

Problems and incidents will happen, but you don't want to be bitten by the same problem twice.

is working and know how to use it (and find it).

- *Runbook (or playbooks)?* Has the team planned through what to do when things go wrong? Is there enough documentation for someone less familiar with the project to build the code and deploy it? Make sure it is clear how to restart, reboot, clear the cache, warm up the cache, deploy clean, etc.
- *SLOs?* Service-level objectives are important to think about at the start so it is clear what the user flow should be and if the software is meeting that objective.
- *Disaster recovery plans?* What happens when everything goes wrong? Are there backups? How do you restore from the backup? Have you thought about failover and redundancy?
- *Dependencies?* What happens when dependencies fail? Do the clients degrade gracefully? What will the user/customer experience? Can the system operate if a downstream service is slow, unresponsive, or unavailable?
- *Load and performance testing?* Do you know what the limits are for the service? What will need to be done to add more capacity if required?
- *Compliance?* If you work in a high-compliance environment, have you met all the regulations and standards (potentially including things like a security and privacy review)?

Manage your incidents and closely track follow-up items

Problems and incidents *will* happen, but you don't want to be bitten by the same problem twice. Make sure you understand what the incident load looks like in your team, and how they are doing against closing open items from those incidents.

- *Set up an incident review process.* Whether you do retrospectives, root cause analysis, or other follow-ups, make sure that the team knows when someone is being paged and does the follow-up work to prevent those incidents in the future.
- *Measure alert volume.* How often is your team getting paged? How many alerts per incident? How often are incidents occurring after hours? If any of these numbers seem high, it might be time to invest in improving your alerts, or the underlying causes, to prevent burnout, alert fatigue, and/or outages.
- *Look at incident response.* How are incidents handled (jump on Zoom and observe)? Is the quality of the docs good enough? Are there any SMEs (subject matter experts) in areas that are single points of failure? Sometimes you might need to add a leader to these calls, if you don't already have one in place, and make sure the team stays focused on fixing the issue (rather than trying to understand the cause).
- *Track action items.* Are follow-up items handled with the right level of urgency? Make sure you have a regular (I like weekly) view into incident action items and how they are trending. You want to make sure that these are handled with the right priority.

Manage on-call rotations

Another important part of your leadership role is architecting a sensible on-call rotation for your team(s). Grouping like services and expanding rotations can be helpful if it always takes the same two or three teams to solve an incident (for tightly coupled services). Some organizations also set up a front-end or mobile on-call rotation to respond quickly to urgent bugs or issues in clients. As a leader, you should think about the following aspects:

- How often is someone on call?
- How long does a rotation last?
- What happens when someone fails to answer a page?
- How many engineers are on call at any given time? If you have a primary and secondary, how do multiple on-call engineers divide their duties?
- Do the people on call have the skills and support to respond to issues when they arise?
- How painful is the on-call rotation (i.e., how often are people being paged)?
- When people are on call, are they also expected to contribute to their team's feature work, or are they free to work on incident action items or other high-priority bugs?
- Does everyone on call have all the permissions and tools they need (and do they know what to do with them)?
- How does the model select team members for each on-call rotation?

Manage your data

As a leader of the team, do you know how your software is performing? This is more than just uptime—you should be paying attention to all of your key user flows through the system, looking at throughput, latency, etc.

- How do you know that the business and services are

healthy? Do you have a dashboard? How often do you look at it?
- At any given time, do you know what feature flags are turned on in your environment?
- Do you know what marketing, sales, or other promotions are live?
- Are your systems elastic? When do you need to start working on capacity and throughput?

Besides the ability to handle incidents and outages, another important part of operational excellence is understanding the customer experience.

Track customer-reported issues

Besides the ability to handle incidents and outages, another important part of operational excellence is understanding the customer experience. In addition to system metrics, you should be paying attention to all the information you get from customers. For example:
- How often do customers report issues? How do you track them? Is this list shrinking or growing?
- How do you factor in and manage quality in the context of feature work?
- How often do services time out, error out, or result in crashes (e.g., on mobile)?

Manage failover and recovery

Do you have disaster recovery plans? Do your systems degrade gracefully? How long would it take to recover from a major service interruption, including your CSP (content security policy) building blocks?

Manage CI/CD, testing, and automation

This topic could be an article all to itself. Having good testing, automation, robust CI/CD (continuous integration/

continuous delivery) pipelines, etc., helps prevent problems. Ask yourself (or your engineers): How do you know the code you are pushing is high-quality? What would you need to do to answer that question with a yes?

- Do you feel confident about your deployments?
- Do you do canary releases with test and staging environments, or using feature flags to roll out iteratively?
- Do you have synthetics and RUM (real user monitoring) for your top user flows?
- Are all your systems instrumented with the right level of observability (i.e., profiling, add-ons for specific vendors, open-source packages, etc.)?
- Can you always roll back or failover to a known good instance?

KEYS TO SUCCESS

As you go through each of these points, you can find many ways to make adjustments and improve in each area. The first step is asking the right questions. The second step may include things such as:

- Investing in quality assurance
- Automating tasks
- Updating, standardizing, improving processes
- Measuring and tracking performance (and paying attention to the data)
- Creating a culture of continuous improvement

Operational excellence is a critical part of the success of any software engineering team. As a leader, you have a huge opportunity to improve the way your team is working. Best of luck and wishing you 100 percent uptime.

Kate Matsudaira *is VP of technology for SoFi's Money (checking and savings), credit card, Invest, insurance, At Work, and partnerships. Previously, she was a VP at Splunk, where she was responsible for the Observability product suite. She has also worked as an executive at Google and helped build several successful startups that were acquired by eBay, O'Reilly Media, and Limelight. She started her career as a software engineer and lead at Microsoft and Amazon. She is a keynote speaker and published author, and has been honored with recognitions such as the NCWIT Symons Innovator Award. She lives in Issaquah, WA (outside of Seattle), with her husband, Garrett; three boys; and three dogs.*

alternative artwork
for first page