PagerDuty

Best Practices for Navigating the Cultural Shift to Service Ownership

Table of Contents

PagerDuty

Digital Transformation: To the Cloud and Beyond

Today, technical leaders across industries are accountable for delivering high-quality, always-on customer experiences through digital services—especially since downtime, outages, and slowdowns can negatively impact their company's revenue and brand. According to IDG's IDG's 2022 State of the CIO, 85% of IT leaders responding to the research are currently spending time on transformational activities..

If you're a technical leader tasked with digital transformation, you can't avoid the cloud. And with hybrid and multi-cloud environments becoming increasingly common, many organizations are faced with the challenge of dealing with more complexity in their digital operations infrastructure. A side effect of this is teams becoming decentralized into lines of business, each with their own toolchains and workflows, which can lead to difficulty in visibility and collaboration.

As systems and teams get more complex, there is no way to efficiently manage everything in a purely centralized fashion. This can be especially painful when it comes to incident response because when incidents happen, siloed systems and teams operating in traditional models can create a domino effect that can negatively impact the customer experience and put your business at risk.

The Case for Full-Service Ownership, a.k.a. "You Build It, You Own It"

With the increased reliance on digital services across all aspects of life today, there's more at stake when things go wrong. And things will go wrong—service failure is an inevitable part of operating with technology and systems. But how your company responds when that happens makes all the difference when it comes to the customer experience and, ultimately, your revenue.

In order to streamline the incident response process, you need several things:

- Visibility into the incident to properly acknowledge and triage.
- Intelligent routing and context, so the right teams are mobilized with the right information.
- Focused orchestration to make the incident response as swift as possible so the impact is contained.

Along with methodologies like agile and DevOps, service ownership has been a key opportunity for technical organizations that need to adapt to new cloud and hybrid environments.

What Is Service Ownership?

Service ownership is an operating model where the people closest to the technology from a design and implementation perspective are responsible for it for the entire product development lifecycle. "You code it, you own it" is a popular phrase associated with the methodology.

What is a service?

A service is a discrete piece of functionality that provides value and is wholly owned by a team.

Service ownership, in its simplest form, empowers engineers to be responsible for their code in production, which leads to higher-quality software. Realigning teams to this model gives engineers more control over their work, makes them accountable for the quality of their code, and results in faster, more orchestrated responses to incidents and downtime.

3 Key Benefits of Developers Owning Their Own Code

1. Developers who own their code have a greater impact on the end-user experience.
Service ownership gives developers a tighter connection with the purpose and impact that their work is making for the business and for the end customers. They get more autonomy and responsibility without feeling like they're being restricted, which in turn allows them to focus on building great products.

2. Developers who own their code are more accountable for it in production.
Owning code end-to-end from development, through testing, and into production means that when your code breaks, you're also responsible for fixing it. There is a quality control loop built in where there's a personal incentive to make sure it works—after all, they don't want to be woken up at 3 a.m. when their code fails. Mapping developers to services via service ownership keeps continuity that can outlast reorgs or turnover and backfills—it will always be clear who is responsible for what.

3. Developers who own their code can help drive down MTTR (mean time to resolution).
A key feature of service ownership often involves putting teams on call for the applications and services they own, making developers first in line to be called when things go wrong because they are best suited for fixing it. This is important because faster resolution isn't only good for minimizing impact to the customer and the business, it is also beneficial for the technical teams themselves: When it's clear who owns the code, fewer staff need to be brought in to troubleshoot an issue. Those 300-person conference calls trying to trouble-shoot who touched it last can be a thing of the past.

How Service Ownership Streamlines the Incident Response Lifecycle

Service ownership helps streamline the incident response lifecycle by:

- Empowering engineers to own their services in production, which reduces the number of handoffs and can significantly reduce MTTR when incidents occur.
- Placing subject matter experts with direct knowledge of the systems they support in the role of first responders, which helps decrease the inevitable chaos and panic that arise from uncertainty.

Navigating a Cultural Shift

To successfully adopt service ownership for your organization, it's not enough for engineers to simply take responsibility for their code—a cultural shift is also required.

It helps to start from a position of compassion, shared responsibility, and trust. Individuals need to be empowered to make decisions in an efficient manner and should not fear being blamed for an incident. For example, some may fear that if it's their code or a change made by them that caused downtime, they may be blamed for causing an incident.

Organization-wide buy-in is also required to make a cultural shift to blamelessness in order for service ownership to be truly embraced. Such a cultural change takes time and requires buy-in from the top down, so starting small and setting the example upfront that incidents are not causes for blame can help you set your team up for success.

A Note for Gaining Buy-In From the Greater Organization

For companies transitioning to service ownership from a traditional operating model, there will likely be resistance from a central operations team.

As with all change, there's a fear of what the tradeoff is. Here are few topics that we've seen early adopters of service ownership navigate with their internal partners.

Visibility and Control
Visibility and Control. When presented with the idea of service ownership, IT partners may have a knee-jerk reaction and think, "If I do something differently, I will suddenly lose visibility into X." Contrary to what they may think, when moving things to the cloud, central IT organizations often find that they end up with more visibility and pipeline control than the traditional model may have provided them. It also helps them keep up with the pace of change and offers some solutions for consolidating or reducing noise.

Productivity
How much time do your teams spend on manual tasks? How often do they have to dig through old or outdated documentation and figure out how to address incidents in an ad-hoc manner? Having tools that unify processes instead of passing playbooks back and forth will help streamline and reduce manual work, giving your teams time back to focus on innovation and driving the business forward.

Security and Governance
Every company is going to approach this a little differently, but it's possible to find a middle ground to build processes to address valid points and considerations. It's important to position this from the very start that these efforts require shared responsibility and trust. Change is always hard, but attitudes set the table for how to move the business forward.

Getting Started: 9 Best Practices for Activating Service Ownership

1. Stay agile

Agile methodologies can be essential when implementing a culture of service ownership. Work is often unpredictable and agile methodologies can help teams stay the course toward long-term goals, even when the unexpected occurs.

Agile-type workflows can help identify things that are going well and potential blockers. Adopting agile-type workflows is typically a part of most digital transformation initiatives.

2. Start small

Organizational transformation takes both significant effort and executive support. Gaining executive support begins with demonstrating value, and a useful method for demonstrating that value with mitigated risk is to start small. Choose a non-critical production system and begin the process of defining your service. Next, measure a baseline of performance for your current production system.

Additionally, allow your team the necessary time to learn the stepping stones they will use to master the mechanics of service ownership. One of the more immediate gains you should see is faster incident acknowledgement and, therefore, resolution.

3. Set yourself up for success

Greenfield projects eschew the past and start free from inherited technical and organizational debt. It's simpler to start fresh and design a world free from legacy restrictions. However, if your goal is to prove organizational value, it may better serve your long-term transformational initiative to select an achievable brownfield project.

Here's a simple test: Are you likely to hear the phrase, "That's never going to work here," in your organization if you propose something new? If so, consider selecting a more achievable brownfield project to start with.

4. Don't play the blame game

Mistakes will be made. Digital transformation and service ownership are about learning new models and adapting them to the needs of your organization. Individuals need to be empowered to make decisions and experiment, while not fearing blame or retribution when making the wrong choice.

A culture of psychological safety is one of the leading indicators that predict high-performance among software delivery teams. For in-depth tips on creating a culture of blamelessness, check out our Postmortem Ops Guide.

5. Practice, practice, practice

As with any new skill, practicing in low-risk settings builds the confidence needed when the real thing happens. For teams that are new to owning production services, simulating incidents is a great way for them to become accustomed to managing incidents when they occur. Chaos engineering can work wonders to help with building more reliable workflows.

6. Right-size your teams

In the early days of Amazon, Jeff Bezos set a rule: Teams shouldn't be larger than what two pizzas can feed, no matter how large a company gets. Setting this rule of small teams meant individuals spent less time providing status updates to each other and more time actually getting stuff done. It also allowed team members more time to focus on continuous improvement

7. Be clear when defining services

Services should be set up granularly enough to help identify the source of problems. If two microservices always behave as one, and fixing a problem on one means fixing it also on another, it might make sense to combine them. Document dependencies to help define roles and responsibilities so incidents don't fall through the cracks.

8. Have a plan for your monolith(s)

If you have a monolith, consider how you will address on-call responsibilities for it. Monoliths tend to be the cause of a lot of incidents, both actionable and non-actionable. If one team owns the monolith, they usually don't own any other services unless the on-call load for the monolith is low.

If multiple teams share responsibility for the monolith, consider gradually identifying different sources of functionality and routing those alerts to teams that have the right context. Each source of functionality can be represented as a different service for your documentation, enumerating runbooks and wikis, along with your on-call ownership in platforms like PagerDuty.

9. Don't forget to document for the masses

Service documentation is a shared responsibility among every engineer contributing to the code base. You wrote it, you document it.

Aside from writing code, there are key functional concerns for owning a service that extend well beyond checking code into a shared repo with a sparse README that maybe explains what the repo contains. Documentation provided should explain what the code does, in addition to a contract that other services can understand if they need to interact with the service.

Service Ownership Is a Shared Responsibility

At the end of the day, service ownership is a shared responsibility across cross-functional parts of your organization. Engineers who wrote code aren't the only people accountable for ownership of a service. It truly takes a village.

For even more information around service ownership and how to implement it at your organization, check out our Service Ownership Ops Guide.

To see how you can empower your developers to own their code, visit www.pagerduty.com or start a 14-day free trial today.